SD Codes: Erasure Codes Designed for How Storage Systems Really Fail

James S. Plank
EECS Department
University of Tennessee
plank@cs.utk.edu

Mario Blaum
Independent Contractor to
IBM Almaden Research Center

James L. Hafner
IBM Research Division
Almaden Research Center

Appearing in FAST 2013:
11th USENIX Conference on File and Storage Technologies
February, 2013

http://web.eecs.utk.edu/~plank/plank/papers/FAST-2013-SD.html

SD Codes: Erasure Codes Designed for How Storage Systems Really Fail

James S. Plank
EECS Department
University of Tennessee

Mario Blaum
IBM Research Division
Almaden Research Center

James L. Hafner
IBM Research Division
Almaden Research Center

Abstract

Traditionally, when storage systems employ erasure codes, they are designed to tolerate the failures of entire disks. However, the most common types of failures are latent sector failures, which only affect individual disk sectors, and block failures which arise through wear on SSD's. This paper introduces SD codes, which are designed to tolerate combinations of disk and sector failures. As such, they consume far less storage resources than traditional erasure codes. We specify the codes with enough detail for the storage practitioner to employ them, discuss their practical properties, and detail an open-source implementation.

1 Introduction

Storage systems have grown to the point where failures are commonplace and must be tolerated to prevent data loss. All current storage systems that are composed of multiple components employ erasure codes to handle disk failures. Examples include commercial storage systems from Microsoft [10, 25], IBM [27], Netapp [12], HP [31], Cleversafe [45] and Panasas [55], plus numerous academic and non-commercial research projects [8, 11, 23, 28, 29, 46, 50, 51, 54, 57]. In all of these systems, the unit of failure is the disk. For example, a RAID-6 system dedicates two parity disks to tolerate the simultaneous failures of any two disks in the system [4, 12]. Larger systems dedicate more disks for coding to tolerate larger numbers of failures [10, 45, 54].

Recent research, however, studying the nature of failures in storage systems, has demonstrated that failures of entire disks are relatively rare. The much more common failure type is the *Latent Sector Error* or *Undetected Disk Error* where a sector on disk becomes corrupted, and this corruption is not detected until the sector in question is subsequently accessed for reading [2, 16, 22].

Additionally, storage systems are increasingly employing solid state devices as core components [3, 20, 26, 35]. These devices exhibit wear over time, which manifests in the form of blocks becoming unusable as a function of their number of overwrites, and in blocks being unable to hold their values for long durations.

To combat block failures, systems employ *scrubbing*, where sectors and blocks are proactively probed so that errors may be detected and recovered in a timely manner [1, 16, 37, 47, 48]. The recovery operation proceeds from erasure codes — a bad sector on a device holding data is reconstructed using the other data and coding devices. A bad sector on a coding device is re-encoded from the data devices.

Regardless of whether a system employs scrubbing, a potentially catastrophic situation occurs when a disk fails and a block failure is discovered on a non-failed device during reconstruction. It is exactly this situation which motivated companies to switch from RAID-5 to RAID-6 systems in the past decade [12, 16].

However, the RAID-6 solution to the problem is overkill: two entire disks worth of storage are dedicated to tolerate the failure of one disk and one sector. In effect, an entire disk is dedicated to tolerate the failure of one sector.

In this paper, we present an alternative erasure coding methodology. Instead of dedicating entire disks for fault-tolerance, we dedicate entire disks and individual sectors. For example, in the RAID-6 scenario above, instead of dedicating two disks for fault-tolerance, we dedicate one disk and one sector per stripe. The system is then fault-tolerant to the failure of any single disk and any single sector within a stripe.

We name the codes "SD" for "Sector-Disk" erasure codes. They have a general design, where a system composed of n disks dedicates m disks and s sectors per stripe to coding. The remaining sectors are dedicated to data. The codes are designed so that the simultaneous failures of any m disks and any s sectors per stripe may be tolerated without data loss.

In this paper, we present these codes for the storage practitioner and researcher. We do not dwell on the codes' theoretical aspects, but instead present them in such a way that they may be used in practical storage settings. We evaluate their practical properties, which we summarize briefly here. The main practical benefit is achieving an enriched level of fault-tolerance with a minimum of extra space. The CPU performance of the codes is less than standard Reed-Solomon codes, but well fast enough to make disk I/O the bottleneck rather than CPU. The main performance penalty of the codes is that $m+s$ sectors must be updated with every modification to a data block, making the codes more ideal for cloud, archival or append-only settings than for RAID systems that exhibit a lot of small updates. Finally, we have written an SD encoder and decoder in C, which we post as open source.

2 System Model and Nomenclature

We concentrate on a *stripe* of a storage system, as pictured in Figure 1. The stripe is composed of n disks, each of which holds r sectors. We may view the stripe as an $r \times n$ array of sectors; hence we call r the number of rows. Of the n disks, m are devoted exclusively to coding. In the remaining $n-m$ disks, s additional sectors are also devoted to coding. The placement of these sectors is arbitrary; by convention, we picture them evenly distributed in the bottom rows of the array.

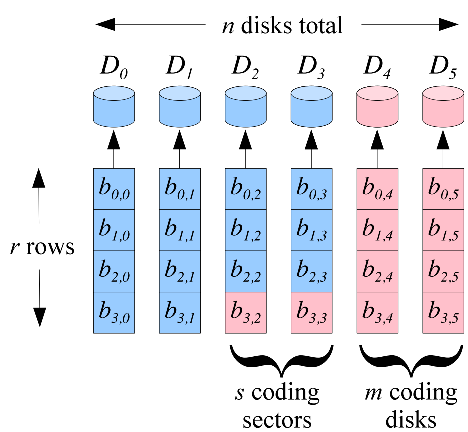

Figure 1: A stripe in a storage system with n total disks. Each disk is partitioned into r rows of sectors. Of the n disks, m are devoted exclusively to coding, and s additional sectors are devoted to coding.

While we refer to the basic blocks of the array as sectors, they may comprise multiple sectors. For the purposes of coding, we will also consider them to be w-bit symbols, where w is a parameter of the erasure code, typically 8, 16 or 32, so that sectors may be partitioned evenly into symbols. As such, we will use the terms "block," "sector" and "symbol" interchangeably.

Storage systems may be partitioned into many stripes, where the identities of the coding disks change from stripe to stripe to alleviate bottlenecks. The identities may be rotated as in RAID-5, or performed on an ad-hoc, per-file basis as in Panasas [55]. Blocks may be small, as in the block store of Cleversafe's distributed file system [45] or large as in Google FS [17]. As such, the mapping of erasure code symbols to disk blocks is beyond the scope of this work. It depends on many factors, including projected usage, system size and architecture, degree of correlated sector failures, and distribution of storage nodes. A thorough discussion of the performance implications of mapping erasure code symbols to disk blocks may be found in recent work by Khan *et al* [28].

3 Arithmetic for Coding

To perform encoding and decoding, each coding symbol is defined to be a linear combination of the data symbols. The arithmetic employed is Galois Field arithmetic over w-bit symbols, termed $GF(2^w)$. This is the standard arithmetic of Reed-Solomon coding which features a wealth of instructional literature and open source implementations [18, 33, 36, 41, 43, 44, 39]. Addition in $GF(2^w)$ is equivalent to the bitwise exclusive-or operation. Multiplication is more complex, but a recent open source implementation employs Intel's SIMD instructions to perform multiplication in $GF(2^8)$ and $GF(2^{16})$ at cache line speeds. $GF(2^{32})$ is only marginally slower [43].

Typical presentations of erasure codes based on Galois Field arithmetic use terms like "irreducible polynomials" and "primitive elements" to define the codes [5, 6, 34, 40]. In our work, we simply use numbers between 0 and (2^w-1) to represent w-bit symbols, and assume that the codes are implemented with a standard Galois Field arithmetic library such as those listed above. Our goal is to be practical rather than overly formal.

4 SD Code Specification

SD codes are defined by six parameters listed below:

Parameter	Description
n	The total number of disks
m	The number of coding disks
s	The number of coding sectors
r	The number of rows per stripe
$GF(2^w)$	The Galois field
$A=\{a_0,\ldots,a_{m+s-1}\}$	Coding coefficients

We label the disks D_0 through D_{n-1} and assume that disks D_{n-m} through D_{n-1} are the coding disks. There are nr blocks in a stripe, and we label them in two ways. The first assigns subscripts for the row and the disk, so that disk D_i holds blocks $b_{0,i}$ through $b_{r-1,i}$. This labeling is the one used in Figure 1. The second simply numbers the blocks consecutively, b_0 through b_{nr-1}. The mapping between the two is that block $b_{j,i}$ is also block b_{jn+i}. Figure 2 shows the same stripe as Figure 1, except the blocks are labeled with their single subscripts.

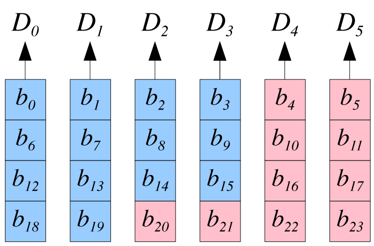

Figure 2: The same stripe as Figure 1, except blocks are labeled with single subscripts.

Instead of using a generator matrix as in Reed Solomon codes, we employ a set of $mr+s$ equations, each of which sums to zero. The first mr of these are labeled $C_{j,x}$, with $0 \le x < m$ and $0 \le j < r$. They are each the sum of exactly n blocks in a single row of the array:

$$C_{j,x}: \sum_{i=0}^{n-1} a_x^{jn+i} b_{j,i} \quad = \quad 0.$$

The remaining s equations are labeled S_x with $0 \le x < s$. They are the sum of all nr blocks:

$$S_x: \sum_{i=0}^{rn-1} a_{m+x}^{i} b_i \quad = \quad 0.$$

Intuitively, one can consider each block $b_{j,i}$ on a coding disk to be governed by $C_{j,x}$, and each additional coding block governed by a different S_x. However, the codes are not as straightforward as, for example, classic Reed-Solomon codes, where each coding block is the linear combination of the data blocks. Instead, unless m equals one, every equation contains multiple coding blocks, which means that encoding must be viewed as a special case of decoding.

A concrete example helps to illustrate and convey some intuition. Figure 3 shows the ten equations that result when the stripe of Figures 1 and 2 is encoded with $A = \{1,2,4,8\}$. The figure is partitioned into four smaller figures, which show the encoding with each a_i. The top two figures show the equations $C_{j,0}$ and $C_{j,1}$, which employ a_0 and a_1 respectively. Each equation is the sum of six blocks. The bottom two figures show S_0 and S_1, which are each the sum of all 24 blocks.

As mentioned above, encoding with these equations is not a straightforward activity, since each of the ten equations contains at least two coding blocks. Thus, encoding is viewed as a special case of decoding — when the two coding disks and two coding sectors fail.

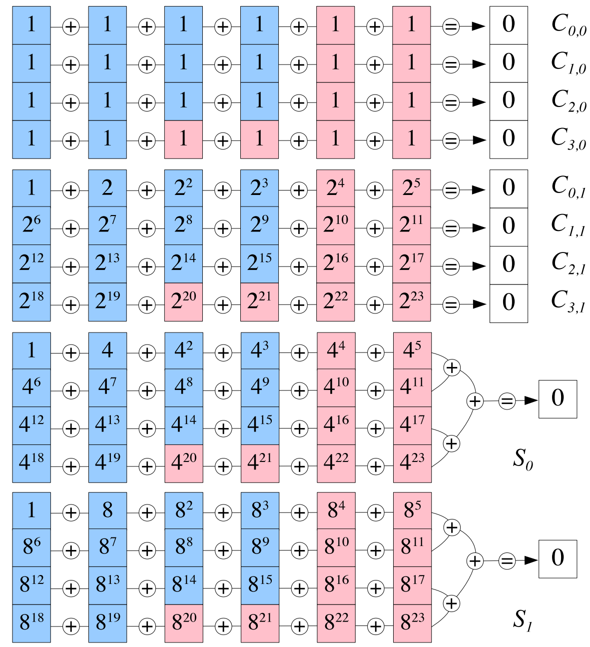

Figure 3: The ten equations to define the code when $n = 6$, $m = 2$, $s = 2$ and the values of a_i are 1, 2, 4 and 8.

The decoding process is straightforward linear algebra. When a collection of disks and sectors fail, their values of b_i are considered unknowns. The non-failed blocks are known values. Therefore, the equations become a linear system with unknown values, which may be solved using Gaussian Elimination.

For example, suppose we want to encode the system of Figure 3. To do that, we assume that disks 4 and 5 have failed, along with blocks b_{20} and b_{21}. The ten equations are rearranged so that the failed blocks are on the left and the nonfailed blocks are on the right. Since addition is equivalent to exclusive-or, we may simply add a term to both sides of the equation to move it from one side to another. For example, the four equations for $C_{j,1}$ become:

$$2^4b_4 + 2^5b_5 = b_0 + 2b_1 + 2^2b_2 + 2^3b_3$$
$$2^{10}b_{10} + 2^{11}b_{11} = 2^6b_6 + 2^7b_7 + 2^8b_8 + 2^9b_9$$
$$2^{16}b_{16} + 2^{17}b_{17} = 2^{12}b_{12} + 2^{13}b_{13} + 2^{14}b_{14} + 2^{15}b_{15}$$
$$2^{20}b_{20} + 2^{21}b_{21} + 2^{22}b_{22} + 2^{23}b_{23} = 2^{18}b_{18} + 2^{19}b_{19}$$

We are left with ten equations and ten unknowns, which we then solve with Gaussian Elimination or matrix inversion.

This method of decoding is a standard employment of a Parity Check Matrix [34, 40]. This matrix (conventionally labeled H) contains a row for every equation and a column for every block in the system. The element in row i column j is the coefficient of b_j in equation i. The vector $B = \{b_0, b_1, \ldots, b_{nr-1}\}$ is called the *codeword*, and the $mr+s$ equations are expressed quite succinctly by the equation $HB = 0$.

5 The SD Condition and Constructions

A code specified by the parameters above is SD if it decodes all combinations of m disks and s sectors (blocks). In other words, when the $mr+s$ decoding equations are created, the Gaussian Elimination proceeds successfully. Put another way, the sub-matrix of the Parity Check Matrix composed of the columns that correspond to failed blocks must be invertible.

Unlike Reed-Solomon codes, there is no general SD code construction for arbitrary n, m, s and r. However, for parameters that are likely in storage systems, we do have valid constructions of SD codes. In the absence of theory, verifying that a set of parameters generates a valid SD code requires enumerating failure scenarios and verifying that each scenario can be handled. There are:

$$\binom{n}{m}\binom{r(n-m)}{s}$$

failure scenarios, which means that the time to verify codes blows up exponentially. For example, verifying the SD code for $n = 24$, $m = 3$, $s = 3$, $r = 24$ and $w = 32$ took roughly three days on a standard processor.

There is some theory to help us in certain cases. Blaum *et al* have developed verification theorems for PMDS codes, which are a subset of SD codes [5]. PMDS codes provably tolerate more failures than SD codes: m arbitrary failures per row, plus any additional s failures in the stripe. For the purposes of most storage systems, we view the additional failure protection of PMDS codes as overkill; however, when their verification theorems apply, they are faster than enumerating failure scenarios. Therefore, we use them to verify codes, and when we fail to verify that codes are PMDS, we resort to the enumeration above.

As with Reed-Solomon codes, the choice of w affects code construction — larger values of w generate more codes. However, the rule of thumb is that larger values of w also result in slower CPU performance. Therefore, we start our search with $w = 8$, then move to $w = 16$ and then to $w = 32$. We focus on $4 \le n \le 24$, $1 \le m \le 3$, $1 \le s \le 3$ and $r \le 24$, which encompasses the majority of situations where SD codes will apply.

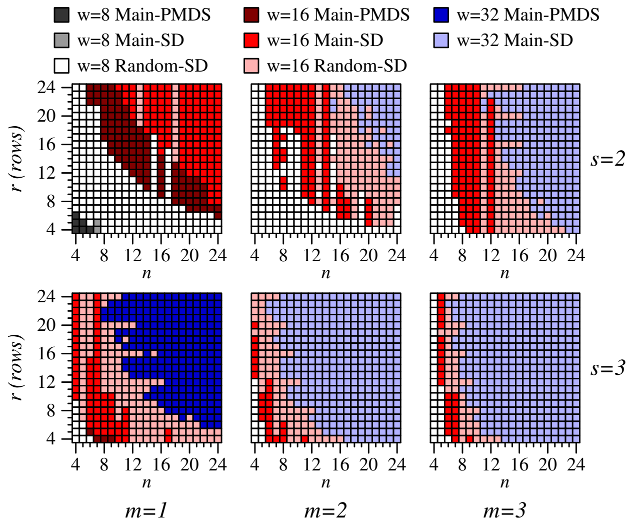

Figure 4: SD codes for values of n, m, s and r that would be common in disk systems.

We focus on two constructions. The first is when $a_i = 2^i$ in $GF(2^w)$. This is called "Code $C^{(1)}$ by Blaum *et al* [5], who have developed PMDS theorems for it. We will call it "the main construction." The second is when a_0 is equal to one, but the other a_i are chosen arbitrarily. We will call these "random constructions." Given values of n, m, r, s and w, our methodology for constructing SD codes as a follows. We first test whether the main construction is PMDS according to Blaum *et al*. If not, then we perform the enumeration to test whether the main construction code is SD, but not PMDS. If not, we then perform a Monte Carlo search to find a random SD construction. If the Monte Carlo search fails, we leave the search for the code to be an open problem. Although we have dedicated months of CPU time to the Monte Carlo searches, many SD constructions remain open. However, we digest our results below.

We start with $m = 1$ and $s = 1$, and we are protecting the system from one disk and one sector failure. We anticipate that a large number of storage systems will fall into this category, as it handles the common failure scenarios of RAID-6 with much less space overhead. The main construction is PMDS so long as $n \le 2^w$[5]. Therefore, $GF(2^8)$ may be used for all systems with at most 256 disks. When $m > 1$ and $s = 1$, the main construction is PMDS as long as $nr \le 2^w$. Therefore, $GF(2^8)$ may be used while $nr \le 256$, and $GF(2^{16})$ may be used otherwise.

For the remaining combinations of m and s, we digest the results of our search in Figure 4. There is only PMDS theory for $m = 1$; hence for $m = 2$ and $m = 3$, all of the codes are verified using enumeration. As would be expected, PMDS and SD codes for smaller values of n, m, s and r exist with $w = 8$. As the parameters grow, we cannot find codes for $w = 8$, and must shift to $w = 16$. As they grow further, we shift to $w = 32$. For all the parameters that we have tested, the main construction is SD for $w = 32$.

The bottom line is that for parameters that are useful in today's disk systems, there exist SD codes, often in $GF(2^8)$ or $GF(2^{16})$.

6 Practical Properties of SD Codes

The main property of SD codes that makes them attractive alternatives to standard erasure codes such as RAID-6 or Reed-Solomon codes is the fact that they tolerate combinations of disk and sector failures with much lower space overhead. Specifically, to tolerate m disk and s sector failures, a standard erasure code needs to devote $m+s$ disks to coding. An SD code devotes m disks, plus s sectors per stripe. This is a savings of $s(r-1)$ sectors per stripe, or $\frac{s(r-1)}{r}$ disks per system. The savings grow with r and are independent of the number of disks in the system (n) and the number of disks devoted to fault-tolerance (m). Figure 5 shows the significant savings as functions of s and r.

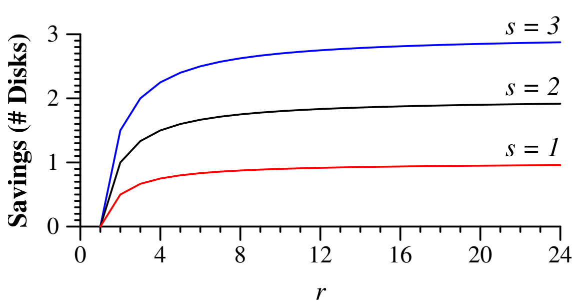

Figure 5: Space savings of SD codes over standard erasure codes as a function of s and r.

The update penalty of SD codes is high. This is the number of coding blocks that must be updated when a data block is modified. In a Reed-Solomon code, the update penalty achieves its minimum value of m, while other codes, such as EVENODD [4], RDP [12] and Cauchy Reed-Solomon [7] codes have higher update penalties. Because of the additional fault-tolerance, the update penalty of an SD code is much higher — assuming that $s \leq n-m$, the update penalty of our SD code construction is roughly $2m+s$. To see how this comes about, consider the modification of block b_0 in Figure 2. Obviously, blocks b_4, b_5, b_{20} and b_{21} must be updated. However, because of blocks b_{20} and b_{21}, blocks b_{22} and b_{23} must also be updated, yielding a total of $2m+s$.

Because of the high update penalty, these codes are appropriate for storage systems that limit the number of updates, such as archival systems which are modified infrequently [46, 51], cloud-based systems where stripes become immutable once they are erasure coded [10, 25], or systems which buffer update operations and convert them to full-stripe writes [15, 38, 49].

To assess the CPU overhead of encoding a stripe, we implemented encoders and decoders for both SD and Reed-Solomon codes. For Galois Field arithmetic, we employ an open source library that leverages Intel's SIMD instructions to achieve extremely fast performance for encoding and decoding regions of memory [43]. We ran tests on a single Intel Core i7 CPU running at 3.07 GHz. We test all values of n between 4 and 24, m between 1 and 3, and s between 1 and 3. We also test standard Reed-Solomon coding. For the Reed-Solomon codes, we test $GF(2^8)$, $GF(2^{16})$ and $GF(2^{32})$. For the SD codes, we set $r = 16$, using the codes from Figure 4.

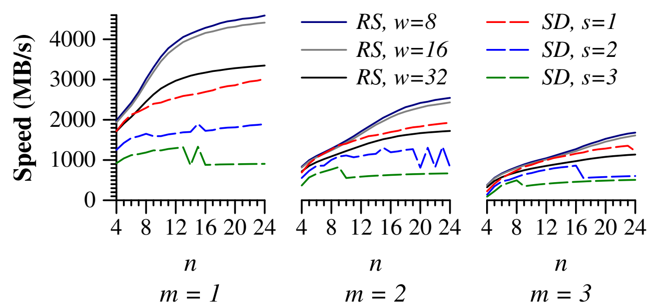

Figure 6: CPU performance of encoding with Reed-Solomon and SD codes.

The results are in Figure 6. Each data point is the average of ten runs. We plot the speed of encoding, which is measured as the amount of *data encoded per second*, using stripes whose sizes are roughly 32 MB. For example, when $n = 10$, $m = 2$ and $s = 2$, we employ a block size of 204 KB. That results in a stripe with 160 blocks (since $r = 16$) of which 126 hold data and 34 hold coding. That is a total of 31.875 MB, of which 25.10 MB is data. It takes 0.0225 seconds to encode the stripe, which is plotted as a rate of 25.10/0.0225 = 1116 MB/s.

The jagged lines in Figure 6 are a result of switching between $GF(2^{16})$ and $GF(2^{32})$, because the Monte Carlo search found codes in $GF(2^{16})$ for some values of n, but did not for others. Because the performance of $GF(2^8)$ is only marginally faster than $GF(2^{16})$, the effect of switching between $GF(2^8)$ and $GF(2^{16})$ is less noticeable.

While SD encoding is slower than Reed-Solomon coding, the speeds in Figure 6 are still much faster than writing to disk. As with other current erasure coding systems (e.g. [25, 28]), the CPU is not the bottleneck; performance is limited by I/O.

To evaluate decoding, we note first that the worst case of decoding SD codes is equivalent to encoding. That is because encoding is simply a special case of decoding that requires the maximum number of equations and terms. The more common decoding cases are faster. In particular, so long as there are at most m failed blocks per row of the stripe, we may decode exclusively from the $C_{j,x}$ equations. This is important, because the $C_{j,x}$ equations have only n terms, and therefore require less computation and I/O than the S_x equations. We do not evaluate this experimentally, but note that the decoding rate for f blocks using the $C_{j,x}$ equations will be equivalent to the Reed-Solomon encoding rate for $m = f$ in Figure 6.

7 Open Source Implementation

We have implemented an SD encoder and decoder in C and have posted it as open source under the New BSD License [42]. The programs allow a user to import data and/or coding information and then to perform either encoding or decoding using the techniques described above. The Galois Field arithmetic implementation leverages the Intel SIMD instructions for fast performance as described above in Section 6. The programs include all of the SD constructions described in Section 5.

Our implementation does not implement RAID or other storage methodologies. As such, we do not expect users to employ the implementation directly, but instead to use it as a blueprint for building their own SD encoded systems.

8 Related Work

The most recent work on erasure codes for storage systems has focused on improving the I/O performance of systems that tolerate multiple failures, when single failures occur [28, 53, 56], and on regenerating codes that replace lost data and coding blocks with reduced I/O for decoding [9, 14, 23, 52]. The focus of this work is on MDS codes in more classic erasure coding environments.

Non-MDS codes have been explored recently because of their reduced I/O costs and applicability to very large systems [19, 24, 32]. In particular, there are several non-MDS codes that organize blocks of a stripe into a matrix and encode rows (inter-disk) and columns (intra-disk) in an orthogonal manner. These include GRID codes [30], HoVeR codes [21] and Intradisk Redundancy [13]. Of these, only the latter code is like SD codes, specifically addressing the heterogeneous failure modes that current disk systems exhibit. The orthogonal nature of Intradisk Redundancy gives it a conceptual simplicity; however the failure coverage of SD codes is higher than Intradisk Redundancy and has greater storage efficiency as well.

In close relation to SD codes are PMDS codes from IBM [5] and Microsoft's LRC codes, which provide fault-tolerance in the Azure cloud storage system [25]. Both are similarly defined codes that achieve a higher level of fault tolerance, but have fewer known constructions. The enhanced fault tolerance of LRC codes is leveraged by Azure, because each code symbol is stored on a different disk, and therefore the "whole disk" failure mode of SD codes does not apply. However, since both LRC and PMDS codes have the SD property, they may be applied in the situations addressed in this paper.

LRC code constructions are limited to $m = 1$. There is quite a bit of theory for constructing PMDS codes which we leverage in our search for SD code constructions. If more theory is developed for PMDS or LRC codes, it can be applied to SD codes as in Section 5. Blaum *et al* were able to discover more PMDS codes for 16-bit symbols by using a variant of $GF(2^{16})$ that is a ring rather than a field. There may be more SD codes as well in this ring. The Galois Field libraries mentioned above do not support rings, so we did not employ them in our search.

9 Conclusion

We have presented a class of erasure codes designed for how today's storage systems actually fail. Rather than devote entire disks to coding, our codes devote entire disks and individual sectors in a stripe, and tolerate combinations of disk and sector failures. As such, they employ far less space for coding than traditional erasure coding solutions.

The codes are similar to Reed-Solomon codes in that they are based on invertible matrices and Galois Field arithmetic. Their constructions are composed of sets of equations that are solved using linear algebra for encoding and decoding. Their performance is not as fast as Reed-Solomon coding, but fast implementations of Galois Field arithmetic allow them to perform at speeds that are fast enough for today's storage systems.

We have written programs that encode and decode using our codes, which we will post as open source, so that storage practitioners may employ the codes without having to understand the mathematics behind them. We are enthused that instances of these codes, developed independently by Huang *et al* [25], are the basis of fault-tolerance in Microsoft's Azure cloud storage system. As such, we anticipate that these codes will have high applicability in large-scale storage installations.

10 Acknowledgements

This material is based upon work supported by the National Science Foundation under grant CSR-1016636 and REU supplement CSR-1128847. The authors would like to thank shepherd Sanjeev Kumar and the FAST referees for constructive comments on the paper, Cheng Huang for discussions about LRC, and undergraduates Adam Disney and Andrew LaPrise for helping search for code constructions.

References

[1] G. Amvrosiadis, A. Oprea, and B. Schroeder. Practical scrubbing: Getting to the bad sector at the right time. In *DSN-2012: The International Conference on Dependable Systems and Networks*, Boston, MA, June 2012. IEEE.

[2] L. N. Bairavasundaram, G. Goodson, B. Schroeder, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau. An analysis of data corruption in the storage stack. In *FAST-2008: 6th Usenix Conference on File and Storage Technologies*, San Jose, February 2008.

[3] M. Balakrishnan, A. Kadav, V. Prabhakaran, and D. Malkhi. Differential RAID: Rethinking RAID for SSD reliability. *ACM Transactions on Storage*, 6(2), July 2010.

[4] 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.

[5] M. Blaum, J. L. Hafner, and S. Hetzler. Partail-MDS codes and their application to RAID type of architectures. IBM Research Report RJ10498 (ALM1202-001), February 2012.

[6] M. Blaum and R. M. Roth. On lowest density MDS codes. *IEEE Transactions on Information Theory*, 45(1):46–59, January 1999.

[7] J. 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.

[8] K. Bowers, A. Juels, and A. Oprea. Hail: A high-availability and integrity layer for cloud storage. In *16th ACM Conference on Computer and Communications Security*, 2009.

[9] V. Cadambe, C. Huang, J. Li, and S. Mehrotra. Compound codes for optimal repair in MDS code based distributed storage systems. In *Asilomar Conference on Signals, Systems and Computers*, 2011.

[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 *23rd ACM Symposium on Operating Systems Principles*, October 2011.

[11] B. Chen, R. Curtmola, G. Ateniese, and R. Burns. Remote data checking for network coding-based distributed storage systems. In *Cloud Computing Security Workshop*, 2010.

[12] P. Corbett, B. English, A. Goel, T. Grcanac, S. Kleiman, J. Leong, and S. Sankar. Row diagonal parity for double disk failure correction. In *3rd Usenix Conference on File and Storage Technologies*, San Francisco, CA, March 2004.

[13] A. Dholakia, E. Eleftheriou, X. Y. Hu, I. Iliadis, J. Menon, and K. K. Rao. A new intra-disk redundancy scheme for high-reliability RAID storage systems in the presence of unrecoverable errors. *ACM Transactions on Storage*, 4(1):1–42, May 2008.

[14] A. G. Dimakis, K. Ramchandran, Y. Wu, and C. Suh. A survey on network codes for distributed storage. *Proceedings of the IEEE*, 99(3), March 2011.

[15] J. K. Edwards, D. Ellard, C. Everhart, R. Fair, E. Hamilton, A. Kahn, A. Kanevsky, J. Lentini, A. Prakash, K. A. Smith, and E. Zayas. FlexVol: Flexible, efficient file volume virtualization in WAFL. In *Usenix 2008 Annual Technical Conference*, pages 129–142, 2008.

[16] J. G. Elerath and M. Pecht. A highly accurate method for assessing reliability of redundant arrays of inexpensive disks. *IEEE Transactions on Computers*, 58(3):289–299, March 2009.

[17] S. Ghemawat, H. Gobioff, and S. T. Leung. The Google file system. In *19th ACM Symposium on Operating Systems Principles (SOSP '03)*, 2003.

[18] K. Greenan, E. Miller, and T. J. Schwartz. Optimizing Galois Field arithmetic for diverse processor architectures and applications. In *MASCOTS 2008: 16th IEEE Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, Baltimore, MD, September 2008.

[19] K. M. Greenan, X. Li, and J. J. Wylie. Flat XOR-based erasure codes in storage systems: Constructions, efficient recovery and tradeoffs. In *26th IEEE Symposium on Massive Storage Systems and Technologies (MSST2010)*, Nevada, May 2010.

[20] K. M. Greenan, D. D. Long, E. L. Miller, T. J. E. Schwarz, and A. Wildani. Building flexible, fault-tolerant flash-based storage systems. In *5th Workshop on Hot Topics in Dependability*, Lisbon, Portugal, June 2009.

[21] J. L. Hafner. HoVer erasure codes for disk arrays. In *DSN-2006: The International Conference on Dependable Systems and Networks*, Philadelphia, June 2006. IEEE.

[22] J. L. Hafner, V. Deenadhayalan, W. Belluomini, and K. Rao. Undetected disk errors in RAID arrays. *IBM Journal of Research & Development*, 52(4/5):418–425, July/September 2008.

[23] Y. Hu, H. C. H. Chen, P. P. C. Lee, and Y. Tang. NCCloud: Applying network coding for the storage repair in a cloud-of-clouds. In *FAST-2012: 10th Usenix Conference on File and Storage Technologies*, San Jose, February 2012.

[24] C. Huang, M. Chen, and J. Li. Pyramid codes: Flexible schemes to trade space for access efficienty in reliable data storage systems. In *NCA-07: 6th IEEE International Symposium on Network Computing Applications*, Cambridge, MA, July 2007.

[25] C. Huang, H. Simitci, Y. Xu, A. Ogus, B. Calder, P. Gopalan, J. Li, and S. Yekhanin. Erasure coding in Windows Azure storage. In *USENIX Annual Technical Conference*, Boston, June 2012.

[26] W. K. Josephson, L. A. Bongo, D. Flynn, and K. Li. DFS: A file system for virtualized flash storage. In *FAST-2010: 8th Usenix Conference on File and Storage Technologies*, San Jose, February 2010.

[27] D. Kenchammana-Hosekote, D. He, and J. L. Hafner. REO: A generic RAID engine and optimizer. In *FAST-2007: 5th Usenix Conference on File and Storage Technologies*, pages 261–276, San Jose, February 2007.

[28] O. Khan, R. Burns, J. S. Plank, W. Pierce, and C. Huang. Rethinking erasure codes for cloud file systems: Minimizing I/O for recovery and degraded reads. In *FAST-2012: 10th Usenix Conference on File and Storage Technologies*, San Jose, February 2012.

[29] H. Klein and J. Keller. Storage architecture with integrity, redundancy and encryption. In *23rd IEEE International Symposium on Parallel and Distributed Processing*, Rome, Italy, 2009.

[30] M. Li, J. Shu, and W. Zheng. GRID codes: Strip-based erasure codes with high fault tolerance for storage systems. *ACM Transactions on Storage*, 4(4), January 2009.

[31] X. Li, A. Marchant, M. A. Shah, K. Smathers, J. Tucek, M. Uysal, and J. J. Wylie. Efficient eventual consistency in Pahoehoe, an erasure-coded key-blob archive. In *DSN-10: International Conference on Dependable Systems and Networks*, Chicago, 2010. IEEE.

[32] M. Luby. LT codes. In *IEEE Symposium on Foundations of Computer Science*, 2002.

[33] J. Luo, K. D. Bowers, A. Oprea, and L. Xu. Efficient software implementations of large finite fields $GF(2^n)$ for secure storage applications. *ACM Transactions on Storage*, 8(2), February 2012.

[34] F. J. MacWilliams and N. J. A. Sloane. *The Theory of Error-Correcting Codes, Part I*. North-Holland Publishing Company, Amsterdam, New York, Oxford, 1977.

[35] Y. Oh, J. Choi, D. Lee, and S. H. Noh. Caching less for better performance: Balancing cache size and update cost of flash memory cache in hybrid storage systems. In *FAST-2012: 10th Usenix Conference on File and Storage Technologies*, San Jose, February 2012.

[36] Onion Networks. Java FEC Library v1.0.3. Open source code distribution: `http://onionnetworks.com/fec/javadoc/`, 2001.

[37] A. Oprea and A. Juels. A clean-slate look at disk scrubbing. In *FAST-2010: 8th Usenix Conference on File and Storage Technologies*, pages 57–70, San Jose, February 2010.

[38] J. K. Ousterhout and F. Douglis. Beating the I/O bottleneck: A case for log-structured file systems. *Operating Systems Review*, 23(1):11–27, January 1989.

[39] A. Partow. Schifra Reed-Solomon ECC Library. Open source code distribution: `http://www.schifra.com/downloads.html`, 2000-2007.

[40] W. W. Peterson and E. J. Weldon, Jr. *Error-Correcting Codes, Second Edition*. The MIT Press, Cambridge, Massachusetts, 1972.

[41] J. S. Plank. A tutorial on Reed-Solomon coding for fault-tolerance in RAID-like systems. *Software – Practice & Experience*, 27(9):995–1012, September 1997.

[42] J. S. Plank. Open source encoder and decoder for SD erasure codes. Technical Report UT-CS-13-704, University of Tennessee, January 2013.

[43] J. S. Plank, K. M. Greenan, and E. L. Miller. Screaming fast Galois Field arithmetic using Intel SIMD instructions. In *FAST-2013: 11th Usenix Conference on File and Storage Technologies*, San Jose, February 2013.

[44] J. S. Plank, J. Luo, C. D. Schuman, L. Xu, and Z. Wilcox-O'Hearn. A performance evaluation and examination of open-source erasure coding libraries for storage. In *FAST-2009: 7th Usenix Conference on File and Storage Technologies*, pages 253–265, February 2009.

[45] J. K. Resch and J. S. Plank. AONT-RS: blending security and performance in dispersed storage systems. In *FAST-2011: 9th Usenix Conference on File and Storage Technologies*, pages 191–202, San Jose, February 2011.

[46] S. Rhea, P. Eaton, D. Geels, H. Weatherspoon, B. Zhao, and J. Kubiatowitz. Pond: The OceanStore prototype. In *FAST-2003: 2nd Usenix Conference on File and Storage Technologies*, San Francisco, January 2003.

[47] B. Schroeder, S. Damouras, and P. Gill. Understanding latent sector errors and how to protect against them. In *FAST-2010: 8th Usenix Conference on File and Storage Technologies*, pages 71–84, San Jose, February 2010.

[48] B. Schroeder and G. Gibson. Disk failures in the real world: What does an MTTF of 1,000,000 mean to you? In *FAST-2007: 5th Usenix Conference on File and Storage Technologies*, San Jose, February 2007.

[49] M. Seltzer, K. Bostic, M.K. McKusick, and C. Staelin. An implementation of a log-structured file system for UNIX. In *Conference Proceedings, Usenix Winter 1993 Technical Conference*, pages 307–326, San Diego, CA, January 1993.

[50] M. W. Storer, K. M. Greenan, E. L. Miller, and K. Voruganti. Pergamum: Replacing tape with energy efficient, reliable, disk-based archival storage. In *FAST-2008: 6th Usenix Conference on File and Storage Technologies*, pages 1–16, San Jose, February 2008.

[51] M. W. Storer, K. M. Greenan, E. L. Miller, and K. Voruganti. POTSHARDS – a secure, long-term storage system. *ACM Transactions on Storage*, 5(2), June 2009.

[52] C. Suh and K. Ramchandran. Exact regeneration codes for distributed storage repair using interference alignment. In *IEEE International Symposium on Information Theory (ISIT)*, June 2010.

[53] Z. Wang, A. G. Dimakis, and J. Bruck. Rebuilding for array codes in distributed storage systems. In *GLOBECOM ACTEMT Workshop*, pages 1905–1909. IEEE, December 2010.

[54] B. Warner, Z. Wilcox-O'Hearn, and R. Kinninmont. Tahoe: A secure distributed filesystem. White paper from `http://allmydata.org/~warner/pycon-tahoe.html`, 2008.

[55] B. Welch, M Unangst, Z. Abbasi, G. Gibson, B. Mueller, J. Small, J. Zelenka, and B. Zhou. Scalable performance of the Panasas parallel file system. In *FAST-2008: 6th Usenix Conference on File and Storage Technologies*, pages 17–33, San Jose, February 2008.

[56] 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*, June 2010.

[57] L. Xu. Hydra: A platform for survivable and secure data storage systems. In *International Workshop on Storage Security and Survivability (StorageSS 2005)*, Fairfax, VA, November 2005.