

Sector-Disk (SD) Erasure Codes for Mixed Failure Modes in RAID Systems

James S. Plank
EECS Department
University of Tennessee

Mario Blaum
IBM Almaden Research Center, and
Universidad Complutense de Madrid

Technical Report CS-13-708
EECS Department
University of Tennessee
May, 2013

This paper has been submitted for publication.

Please see the following URL for its status:

<http://web.eecs.utk.edu/~plank/plank/papers/CS-13-708.html>.

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 [CWO⁺11, HSX⁺12], IBM [KHHH07], Netapp [CEG⁺04], HP [LMS⁺10], Cleversafe [RP11] and Panasas [WUA⁺08], plus numerous academic and non-commercial research projects [BJO09, CCAB10, HCLT12, KBP⁺12, KK09, REG⁺03, SGMV08, SGMV09, WWK08, Xu05]. 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 [BBBM95, CEG⁺04]. Larger systems dedicate more disks for coding to tolerate larger numbers of failures [CWO⁺11, RP11, WWK08].

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 [BGS⁺08, EP09, HDBR08].

Additionally, storage systems are increasingly employing solid state devices as core components [BKPM10, GLM⁺09, JBFL10, OCLN12]. 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.

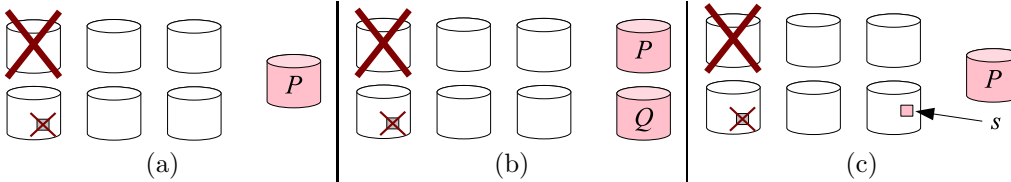


Figure 1: Practical motivation for SD codes. (a) In a RAID-5 disk array, there is one disk (P) devoted to fault-tolerance. The combination of a disk and sector failure results in catastrophic data loss. (b) RAID-6 solves the data loss problem by adding an extra disk (Q) to store redundancy. (c) An SD code solves the data loss problem as well, but only adds one extra sector per stripe (s) for redundancy.

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 [AOS12, EP09, OJ10, SDG10, SG07]. 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 [CEG⁺04, EP09].

We use this situation, where a disk and a sector have failed, as motivation in Figure 1. When RAID-5 is employed (Figure 1(a)), there is only one parity disk, and the failed sector, plus one sector on the failed disk, cannot be recovered. RAID-6 (Figure 1(b)) adds a second parity disk to the system, which can now tolerate the failure of any two disks. Therefore, it can recover from the loss of a disk and a sector. However, RAID-6 solution to the problem is overkill. While it does tolerate the failure of two disks, this is an exceptionally infrequent failure mode, and most systems use RAID-6 primarily to tolerate the scenario of Figure 1(b). In essence, RAID-6 dedicates an entire disk 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. In the scenario of Figure 1(c), 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 SD codes for the storage practitioner and researcher. We present their general design and how they may be implemented in RAID systems. There are a variety of SD constructions that apply to systems with different numbers of storage nodes, different stripe sizes and different fault-tolerance parameters. We present these for parameters that are relevant in today’s storage systems. There are some open problems in constructing SD codes, which we present so that others may build on our work.

We next evaluate the practical properties of SD codes, 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 to aid storage practitioners in implementing these codes in their storage systems.

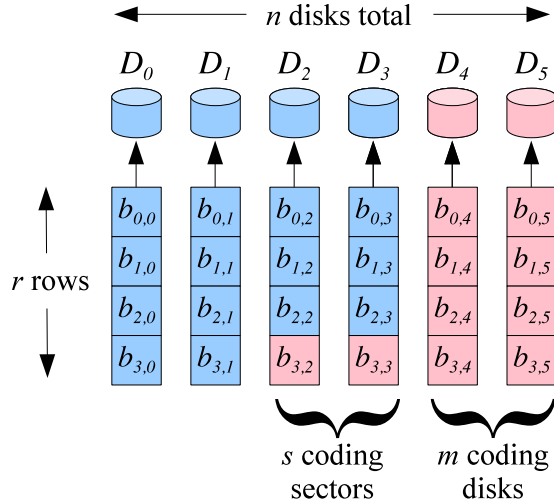


Figure 2: A stripe in a storage system with n total disks. Each disk is partitioned into r rows of sectors. Of the n disks, m are devoted exclusively to coding, and s additional sectors are devoted to coding.

2 System Model and Nomenclature

We concentrate on a *stripe* of a storage system, as pictured in Figure 2. 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.

While we refer to the basic blocks of the array as sectors, they may comprise multiple sectors. For the purposes of coding, we will 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 [WUA⁺08]. Blocks may be small, as in the block store of Cleversafe’s distributed file system [RP11] or large as in Google FS [GGL03]. 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* [KBP⁺12].

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 $GF(2^w)$. Galois Field arithmetic is important because it defines addition, multiplication and division over closed sets of numbers so that every number has a unique multiplicative inverse. It is the standard arithmetic of Reed-Solomon coding, which features a wealth of instructional literature and open source implementations [GMS08, LBOX12, Oni01, Pla97, PGM13, PLS⁺09, Par07]. Addition in $GF(2^w)$ is equivalent to the bitwise exclusive-or operation. Multiplication is more complex, but a recent open source im-

plementation 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 [PGM13].

Typical presentations of erasure codes based on Galois Field arithmetic use terms like “irreducible polynomials” and “primitive elements” to define the codes [BHH12, BR99, MS77, PW72]. 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. (For reproducibility of our results, we employ the same primitive polynomials in our implementations as the software libraries cited above. In hexadecimal, these are 0x11d in $GF(2^8)$, 0x1100b in $GF(2^{16})$ and 0x100400007 in $GF(2^{32})$.)

One feature of Galois Fields that we exploit is that every non-zero number in the field is equal to the value two raised to some power. For example, in standard implementations of $GF(2^8)$, three is equal to 2^{25} and seven is equal to 2^{198} . There are $2^w - 1$ non-zero values in $GF(2^w)$, so the exponents cycle: $2^{w-1} = 1$, and if $x\%(2^w - 1) = y\%(2^w - 1)$, then $2^x = 2^y$ in $GF(2^w)$. This also allows us to employ negative exponents — for example, in $GF(2^8)$, $2^{-1} = 2^{254}$.

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
$A = \{a_{i,j} 0 \leq i < m + s \text{ and } 0 \leq j < nr\}$	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 2. 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 3 shows the same stripe as Figure 2, except the blocks are labeled with their single subscripts.

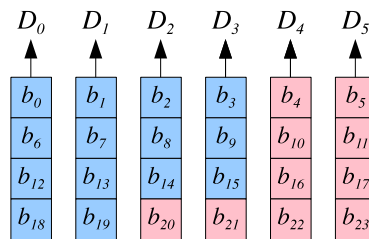


Figure 3: The same stripe as Figure 2, except blocks are labeled with single subscripts.

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 $C_{j,z}$, with $0 \leq z < m$ and $0 \leq j < r$. They are each the sum of exactly n blocks in a single row of the array:

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

We call these equations, “Local Parity Equations,” because they are localized to the blocks in each row of the stripe.

The remaining s equations are labeled S_z with $0 \leq z < s$. They are the sum of all nr blocks:

$$S_z : \sum_{i=0}^{rn-1} a_{m+z,i} b_i = 0.$$

We call these equations “Global Parity Equations,” because they involve all of the blocks in the stripe.

Intuitively, one can consider each block $b_{j,i}$ on a coding disk to be governed by $C_{j,z}$, and each additional coding block governed by a different S_z . However, the codes are not as straightforward as, for example, classic Reed-Solomon codes, where each coding block is specified as a linear combination of the data blocks. Instead, unless m equals one, every equation contains multiple coding blocks. Thus, the equations must be manipulated to calculate the coding blocks. In other words, encoding must be viewed as a special case of decoding, where only the coding blocks have failed.

A concrete example helps to illustrate and convey some intuition. Figure 4 shows the ten equations that result when the stripe of Figures 2 and 3 is encoded with A such that $a_{i,j} = 2^{ij}$. The figure is partitioned into four smaller figures, which show the encoding with each $a_{i,j}$. The left two figures show the equations $C_{j,0}$ and $C_{j,1}$, which employ $a_{0,0}$ through $a_{0,nr-1}$, and $a_{1,0}$ through $a_{1,nr-1}$ respectively. Each equation is the sum of six blocks. The right two figures show S_0 and S_1 , each of which is 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.

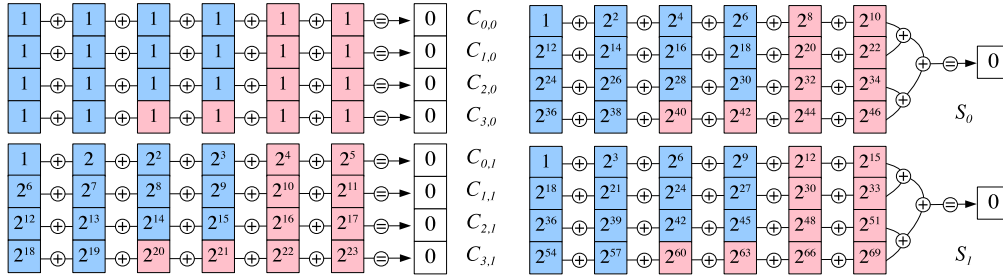


Figure 4: The ten equations to define the code when $n = 6$, $m = 2$, $s = 2$ and $a_{i,j} = 2^{ij}$.

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 4. 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:

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

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 [MS77, PW72, PH13]. 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 $B = \{b_0, b_1, \dots, b_{nr-1}\}$ is called the *codeword*, and the $mr + s$ equations are expressed quite succinctly by the equation $HB = 0$.

5 Defining Fault-Tolerance: MDS, PMDS and SD

We define a hierarchy of fault-tolerance classes for erasure codes that apply to stripes of blocks, such as those depicted in Figures 2 and 3. Each class is a subset of each successive class. For example, if a code is MDS, it is also PMDS and SD. The classes are defined as follows:

- **MDS (Maximum Distance Separable)**: Given a stripe with k data blocks and c coding blocks, an MDS erasure code tolerates the failure of any c of the $k + c$ blocks. The well-known Reed-Solomon codes are canonical examples of MDS codes [MS77, PW72]. We can apply MDS codes to our erasure-coding scenario; however in this case, *every* coding block has to be a function of *every* data block. In other words, every equation would have to be a Global Parity Equation, and encoding/decoding would be exorbitantly expensive. For that reason, we do not consider MDS codes in our evaluation. We mention them here for completeness.
- **PMDS (Partial MDS)**: Given a stripe defined by the parameters n , m , s and r as above in Section 4, a PMDS code tolerates the failure of any m blocks per row, and any additional s blocks in the stripe. PMDS codes are *maximally fault-tolerant* for codes that are defined with mr Local Parity Equations and s Global Parity Equations [GHSY12]. However, PMDS codes make no distinction for blocks that fail together because they are on the same disk.
- **SD (Sector-Disk)**: Given a stripe defined by the parameters n , m , s and r , an SD code tolerates the failures of any m disks (columns of blocks), plus any additional s sectors in the stripe.

For a given code construction, the brute force way to determine whether it is PMDS or SD is to enumerate all failure scenarios and test to make sure that decoding is possible. For SD codes, the total number of scenarios is:

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

Although that is exponential, for smaller values of n , m , s and r , it is computable in a reasonable amount of time. For PMDS, the number of scenarios does not have a simple closed-form equation; however it is much larger than SD. For that reason, we do not test for the PMDS property in a brute force manner. We have written a highly optimized brute-force verifier for SD codes, which enumerates all failure scenarios and incrementally verifies decodability. We have used it to verify all code constructions that we present below in section 6. The largest of these is the code for $n = 24$, $m = 3$, $s = 3$ and $r = 24$ in $GF(2^{32})$. The verification of the 4.9×10^{10} failure scenarios took roughly five and a half hours on a commodity microprocessor.

m	X	Y
1	$\{0, 1, 2\}$	$Y = \{0, 1, -1\}$
2	$\{0, 0, 3, 2\}$	$\{0, 1, -1, 2\}$
3	$\{0, 0, 0, 0, 1\}$	$\{0, 1, -1, 2, -2\}$

Figure 5: SD code constructions for $s = 2$ and $m \in \{1, 2, 3\}$. These constructions are SD as long as $nr < 2^w$.

6 Code Constructions

A major challenge of this work is to derive constructions that generate SD codes. In other words, for given values of n , m , s , r and $GF(2^w)$, our challenge is to define coding coefficients that yield SD codes. Our constructions define the $A_{i,j}$ coefficients using two sets of numbers, $X = \{x_0, \dots, x_{m+s-1}\}$ and $Y = \{y_0, \dots, y_{m+s-1}\}$. These define the coding coefficients in the following manner:

$$a_{i,j} = 2^{x_i(\frac{i}{n})n + y_i(j\%n)}. \quad (1)$$

In the exponent in Equation 1, the division is integer division, and “%” is the modulo operator. For example, the construction employed in Figure 4 may be described by $X = \{0, 1, 2, 3\}$ and $Y = \{0, 1, 2, 3\}$.

In our previous work [PBH13], we limited ourselves to codes where $x_i = y_i$. In other words, we defined codes by $m + s$ coefficients, z_0, \dots, z_{m+s-1} , and $a_{i,j} = z_i^j$. Since every non-zero $z_i \in GF(2^w)$ is equal to 2^{x_i} for some $0 \leq x_i < 2^w - 1$, our old constructions may be represented by $x_i = y_i$ such that 2^{x_i} equals z_i . We call those codes “FAST” codes. The simplest of these is the one in Figure 4, where $x_i = y_i = i$. We call this the “Main Construction.”

In subsequent work, we have discovered that certain codes, where $x_i \neq y_i$, have richer properties than the FAST codes. This is the reason why we describe the codes with two sets of numbers rather than one. Below we detail constructions for which we have proven properties. We do not present the proofs here to save on space and improve readability. Instead, we provide citations to the papers in which the proofs appear.

6.1 $s = 1$

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. Blaum, Hafner and Hetzler prove that the Main Construction is PMDS so long as $n \leq 2^w$ [BHH12]. Therefore, the Main Construction in $GF(2^8)$ may be used for all systems with at most 256 disks.

They also show that when $m > 1$ and $s = 1$, the Main Construction is PMDS as long as $nr \leq 2^w$. Therefore, $GF(2^8)$ may be used while $nr \leq 256$, and $GF(2^{16})$ may be used otherwise.

6.2 $s = 2$.

When $s = 2$, we have derived constructions that are SD for $m \in \{1, 2, 3\}$ so long as $nr < 2^w$. We define them in Figure 5. We have proved the SD property for $m = 1$ and $m = 2$ [BP13]. We have not proved it for $m = 3$; however, we have verified it for all n and r in $GF(2^8)$, and for all $n \leq 24$ and $r \leq 24$ in $GF(2^{16})$.

Because these constructions can be confusing, we give a concrete example in Figures 6 and 7. These show the parity check matrix for the code when $n = 5$, $m = 2$, $s = 2$ and $r = 3$. From Table 5, $X = \{0, 0, 3, 2\}$ and $Y = \{0, 1, -1, 2\}$. We show two representations of the parity check matrix — using powers of two in Figure 6, and as numbers in $GF(2^8)$ in Figure 7.

$C_{0,0}$:	2^0	2^0	2^0	2^0	2^0	0	0	0	0	0	0	0	0	0	0
$C_{1,0}$:	0	0	0	0	0	2^0	2^0	2^0	2^0	2^0	0	0	0	0	0
$C_{2,0}$:	0	0	0	0	0	0	0	0	0	0	2^0	2^0	2^0	2^0	2^0
$C_{0,1}$:	2^0	2^1	2^2	2^3	2^4	0	0	0	0	0	0	0	0	0	0
$C_{1,1}$:	0	0	0	0	0	2^0	2^1	2^2	2^3	2^4	0	0	0	0	0
$C_{2,1}$:	0	0	0	0	0	0	0	0	0	0	2^0	2^1	2^2	2^3	2^4
S_0 :	2^0	2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{30}	2^{29}	2^{28}	2^{27}	2^{26}
S_1 :	2^0	2^2	2^4	2^6	2^8	2^{10}	2^{12}	2^{14}	2^{16}	2^{18}	2^{20}	2^{22}	2^{24}	2^{26}	2^{28}

Figure 6: Parity check matrix for the code for $n = 5$, $m = 2$, $s = 2$, $r = 3$, where $X = \{0, 0, 3, 2\}$ and $Y = \{0, 1, -1, 2\}$. This code is SD when $nr < 2^w$.

$C_{0,0}$:	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0
$C_{1,0}$:	0	0	0	0	0	1	1	1	1	1	0	0	0	0	0
$C_{2,0}$:	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1
$C_{0,1}$:	1	2	4	8	16	0	0	0	0	0	0	0	0	0	0
$C_{1,1}$:	0	0	0	0	0	1	2	4	8	16	0	0	0	0	0
$C_{2,1}$:	0	0	0	0	0	0	0	0	0	0	1	2	4	8	16
S_0 :	1	142	71	173	216	38	19	135	205	232	96	48	24	12	6
S_1 :	1	4	16	64	29	116	205	19	76	45	180	234	143	6	24

Figure 7: The same matrix as in Figure 6, with the values shown as numbers in $GF(2^8)$. Because $nr < 2^8$, this code is SD.

6.3 $s = 3$.

For $s \geq 3$, we have not been able to derive or prove any general SD constructions. Instead, we have performed a pragmatic search for SD codes for parameters that are relevant in today's storage systems. We define these to be n and $r \leq 24$, and $m \leq 3$. Within this space, we first verified that the Main Construction is indeed SD in $GF(2^{32})$. Next, we enumerated the $\binom{255}{3} \approx 2,700,000$ FAST constructions for $m = 1$ in $GF(2^8)$ where $x_0 = y_0 = 0$. That led to a fairly low number of SD codes, and no single construction was SD for all the successful cases.

We then performed three large-scale Monte Carlo searches to discover SD codes for $m \in \{1, 2, 3\}$ in $GF(2^8)$ and $GF(2^{16})$. In the first, we generated random FAST constructions, where $x_0 = y_0 = 0$, and in the second, we generated random values for both X and Y . For each construction, we tested whether the code is SD for each value of n from 4 to 24 and $r = 4$. For each value of n , so long as $r \leq 24$ and the code is SD for $r - 1$, we test to see whether the code is SD for r . If the code is not SD for $r - 1$, it is impossible for the code to be SD for r [BP13]. Thus, we do not spend extra time testing for SD codes that cannot be SD.

The third search arose when the second search generated a code for $n = 7$, $r = 4$, $m = 1$ and $w = 8$. There is no FAST construction in our enumeration that generates an SD code for these parameters, which demonstrates that there are SD codes that cannot be generated by FAST constructions. Additionally, we noticed that in this code, each element of X and Y is a multiple of 15. This spurred an additional enumeration and a third search. The enumeration was of all codes where $m = 1$, $x_0 = y_0 = 0$, and the remaining elements of X and Y are multiples of 15. There are roughly $17^6 \approx 24,000,000$ of these.

The third search Monte Carlo search generated random X and Y whose values are multiples of 15 for $m \in \{2, 3\}$ in $GF(2^8)$, and whose values are multiples of 255 for $m \in \{1, 2, 3\}$ in $GF(2^{16})$. The intuition is that both values are equal to $2^{\frac{w}{2}} - 1$, and while we cannot give a reason why these coefficients are effective, they are indeed effective.

The searches have been performed on over 60 machines at the University of Tennessee for multiple months. On the whole, we have tested over 10,000,000 random constructions, and although the discovery of new codes has slowed, we continue to test more in all three searches.

We present the results of our searches and enumerations in $GF(2^8)$ in Figure 8. The FAST and

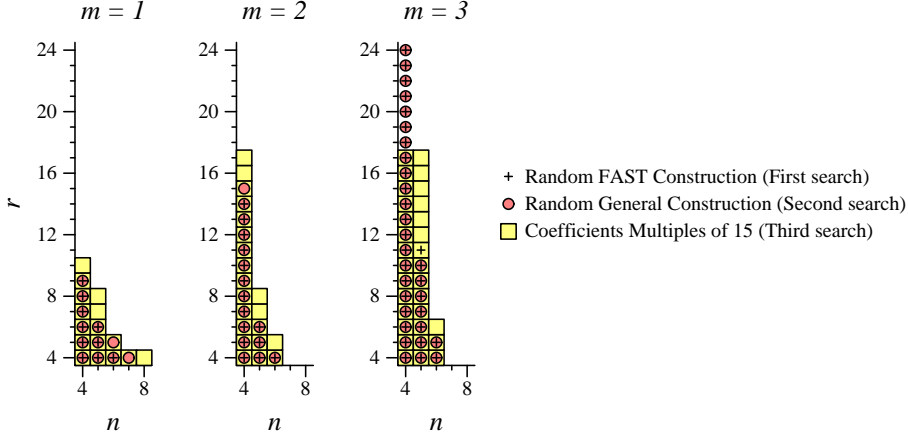


Figure 8: SD Codes for $s = 3$ in $GF(2^8)$, discovered through three Monte Carlo searches and two enumerations.

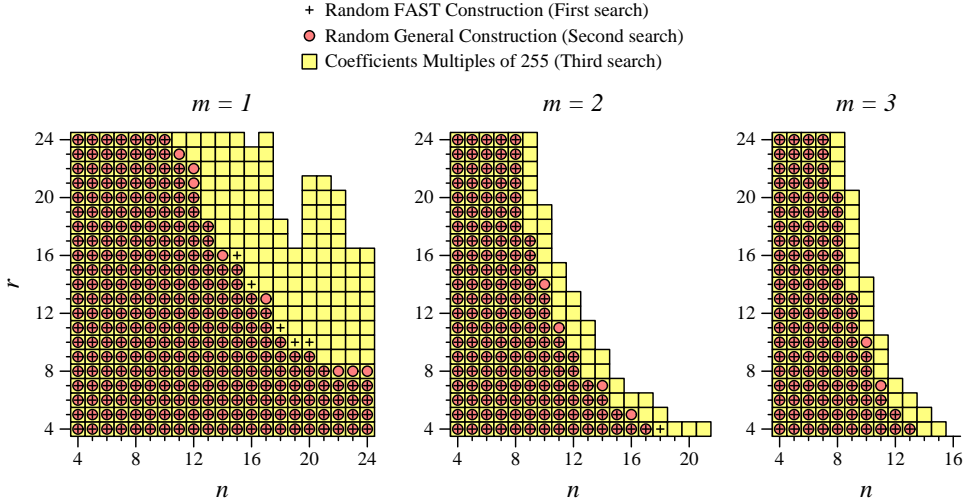


Figure 9: SD Codes for $s = 3$ in $GF(2^{16})$, discovered through three Monte Carlo searches.

multiples of 15 for $m = 1$ are results of enumerations, and the rest are results of Monte Carlo searches. With the exception of $m = 3$, the multiples of 15 generate the widest variety of SD codes. With $m = 3$, the FAST and General Constructions generate codes for all $r \leq 24$ when $n = 4$. There is no one set of coefficients that generates all the SD codes in Figure 8 for any of the constructions. In all cases, the number of SD codes is very limited for $w = 8$.

For $w = 16$, we present the results of our searches in Figure 9. There are more SD codes in these cases, and the third search yielded stunningly more codes than the other two searches. It should be noted that in cases where there is no SD code for given values of n and r , but there is an SD code for $n' > n$ and r , then one may construct an SD code for n and r from the larger code by shortening it. For example, there is no code in Figure 9 for $n = 16$ and $r = 24$; however there is a code for $n = 17$ and $r = 24$. Therefore, one may use this larger code for $n = 16$ by assuming there is a data disk in the $n = 17$ code whose values are all zeros.

As in $GF(2^8)$, there is no set of coefficients responsible for all of the points in any graph. In Figure 10, we present the coefficients that generated the most SD codes for each value of m . It should be noted

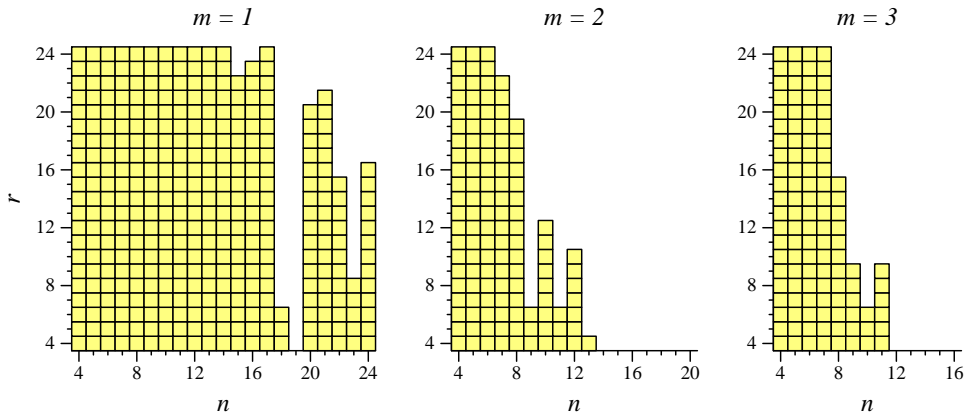


Figure 10: The best sets of coefficients for SD codes in $GF(2^{16})$, $s = 3$. For $m = 1$, these coefficients are $X = \{0, 24480, 28560, 32640\}$, $Y = \{0, 29835, 17850, 35700\}$. For $m = 2$, these coefficients are $X = \{0, 37995, 16575, 20910, 44370\}$, $Y = \{0, 10710, 38505, 23970, 586\}$. For $m = 3$, these coefficients are $X = \{0, 19635, 41820, 28815, 38250, 15300\}$, $Y = \{0, 2295, 3825, 24480, 32895, 48195\}$.

that for $m = 1$, all of the successful cases of Figure 9 with the exception of $n = 22$, $17 \leq r \leq 20$ can be achieved with the code in Figure 10 through shortening.

We include the coefficients that generate all the data points in Figures 8 and 9 as part of our open source software.

The bottom line of this section is as follows. For all values of n , m , s and r that we have considered, there exist SD constructions for these values. In the majority of these cases, these codes are in $GF(2^8)$ and $GF(2^{16})$, which means that their performance will be very good. The remaining cases are handled by the Main Construction in $GF(2^{32})$, which is less efficient; however, as we show below in Section 7, their performance is still adequate for today's storage systems.

7 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 11 shows the significant savings as functions 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 [BBBM95], RDP [CEG⁺04] and Cauchy Reed-Solomon [BKK⁺95] 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 3. 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 [REG⁺03, SGMV09], cloud-based systems where stripes become immutable once they are erasure coded [CWO⁺11, HSX⁺12], or systems which buffer update operations and convert them to full-stripe writes [EEE⁺08, OD89,

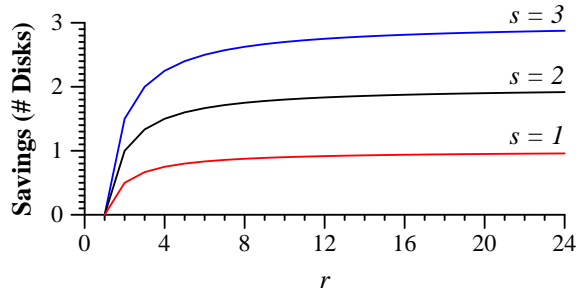


Figure 11: Space savings of SD codes over standard erasure codes as a function of s and r .

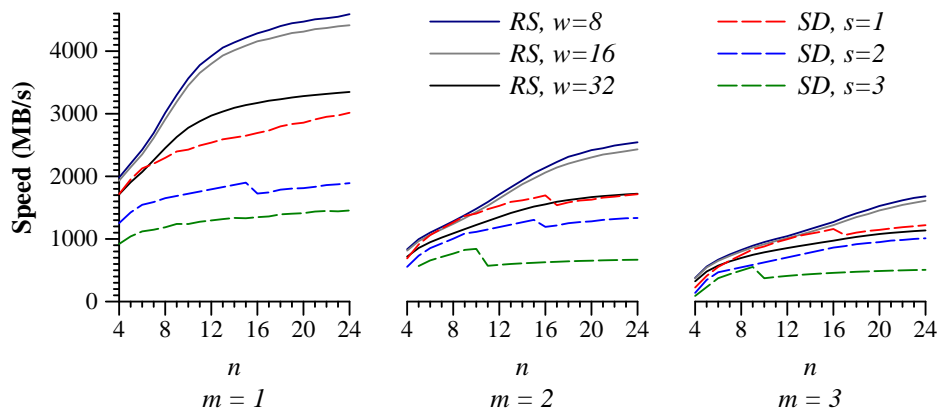


Figure 12: CPU performance of encoding with Reed-Solomon and SD codes.

SBMS93].

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 “GF-Complete,” an open source library that leverages Intel’s SIMD instructions to achieve extremely fast performance for encoding and decoding regions of memory [PGM13]. 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 $r = 16$ and use the best constructions from Section 6.

The results are in Figure 12. 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 0.0217 seconds to encode the stripe, which is plotted as a rate of $25.10/0.0226 = 1111$ MB/s.

The sharp drops in some of the curves in Figure 12 are a result of having to increase w in order to obtain an SD code as n increases. For example, when $m = 1$ and $s = 2$, one must use $GF(2^{16})$ as opposed to $GF(2^8)$ when $n \geq 16$. Accordingly, the curve for $s = 2$ in the leftmost graph of Figure 12 exhibits a sharp drop at $n = 16$. A similar drop occurs in the curve for $s = 3$ in the middle graph at $n = 11$, because this is the point at which one must shift from $GF(2^{16})$ to $GF(2^{32})$.

While SD encoding is slower than Reed-Solomon coding, the speeds in Figure 12 are still much faster than writing to disk. As with other current erasure coding systems (e.g. [HSX⁺12, KBP⁺12]), 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 $C_{j,z}$ equations. This is important, because the $C_{j,z}$ 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 $C_{j,z}$ equations will be equivalent to the Reed-Solomon encoding rate for $m = f$ in Figure 12.

8 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 at https://bitbucket.org/jimplank/sd_codes. 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 7. The programs include all of the SD constructions described in Section 6.

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.

9 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 [KBP⁺12, WDB10, XXLC10], and on regenerating codes that replace lost data and coding blocks with reduced I/O for decoding [CHLM11, DRWS11, HCLT12, SR10]. 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 [GLW10, HCL07, Lub02]. 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 [LSZ09], HoVeR codes [Haf06] and Intradisk Redundancy [DEH⁺08]. 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 they have greater storage efficiency as well.

In close relation to SD codes are PMDS codes from IBM [BHH12] and LRC codes from Microsoft, which provide fault-tolerance in the Azure cloud storage system [HSX⁺12]. 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 6. 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.

Another similar code is the “diff-MDS” code [LMMS⁺11]; however, the focus of this code is correcting bit flips in main memory instead of erasures in storage systems.

10 Open Problems and Further Relaxation

As detailed in Section 6, we have only been able to prove that general constructions are SD for $s = 1$, and for $s = 2$, $m \leq 2$. We surmise that the construction given for $s = 2$, $m = 3$ has the same properties as the others, but its proof is an open problem. Deriving general constructions for larger m and s remain interesting open problems.

As future work, we have begun to explore a further relaxation of the SD property, in hopes that we may derive more constructions for $GF(2^8)$ and $GF(2^{16})$. With this relaxation, we introduce yet another parameter $0 < e \leq s$, and the relaxed SD property specifies that in addition to the m disk failures, the code must tolerate up to s sector failures subject to the limitation that no more than e of these sector failures occur in the same row.

With relaxed SD codes, there are fewer failure scenarios to consider, and we have derived relaxed SD codes where $e = 1$ in some cases where there is no SD code for that set of parameters. It is a subject of future work to explore the benefits and restrictions of relaxed SD codes in relation to SD codes.

11 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 today's 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 post as open source, so that storage practitioners may employ the codes without having to understand the mathematics behind them. We are enthused that codes with similar properties, developed independently by Huang *et al* [HSX⁺12], 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.

12 Acknowledgements

This work is supported by the National Science Foundation, under grants CSR-1016636 and REU Supplement CSR-1128847, and by an IBM Faculty Award.

The authors would like to thank Jim Hafner helping with the early work on this research project, Adam Disney and Andrew LaPrise for helping search for SD codes, Cheng Huang for discussions on the relationship between SD and LRC codes, and Keith Smith and Y. Y. Zhou for recommending the paper to ACM TOS.

Author's addresses: J. S. Plank, EECS Department, University of Tennessee, Knoxville, TN 37991; M. Blaum, IBM Research Division, IBM Almaden Research Center, 650 Harry Road, San Jose, CA 95120.

References

- [AOS12] 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.

- [BBBM95] 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.
- [BGS⁺08] 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.
- [BHH12] M. Blaum, J. L. Hafner, and S. Hetzler. Partial-MDS codes and their application to RAID type of architectures. IBM Research Report RJ10498 (ALM1202-001), February 2012.
- [BJO09] 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.
- [BKK⁺95] 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.
- [BKPM10] M. Balakrishnan, A. Kadav, V. Prabhakaran, and D. Malkhi. Differential RAID: Rethinking RAID for SSD reliability. *ACM Transactions on Storage*, 6(2), July 2010.
- [BP13] M. Blaum and J. S. Plank. Construction of two SD codes. Technical Report arXiv:1305.1221 [cs.IT], arXiv.org, May 2013.
- [BR99] M. Blaum and R. M. Roth. On lowest density MDS codes. *IEEE Transactions on Information Theory*, 45(1):46–59, January 1999.
- [CCAB10] 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.
- [CEG⁺04] 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.
- [CHLM11] 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.
- [CWO⁺11] 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.
- [DEH⁺08] 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.
- [DRWS11] 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.
- [EEE⁺08] 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.

- [EP09] 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.
- [GGL03] S. Ghemawat, H. Gobioff, and S. T. Leung. The Google file system. In *19th ACM Symposium on Operating Systems Principles (SOSP '03)*, 2003.
- [GHSY12] P. Gopalan, C. Huang, H. Simitci, and S. Yekhanin. On the locality of codeword symbols. volume 58, November 2012.
- [GLM⁺09] 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.
- [GLW10] 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.
- [GMS08] 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.
- [Haf06] J. L. Hafner. HoVer erasure codes for disk arrays. In *DSN-2006: The International Conference on Dependable Systems and Networks*, Philadelphia, June 2006. IEEE.
- [HCL07] C. Huang, M. Chen, and J. Li. Pyramid codes: Flexible schemes to trade space for access efficiency in reliable data storage systems. In *NCA-07: 6th IEEE International Symposium on Network Computing Applications*, Cambridge, MA, July 2007.
- [HCLT12] 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.
- [HDBR08] 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.
- [HSX⁺12] 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.
- [JBFL10] 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.
- [KBP⁺12] 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.
- [KHHH07] 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.
- [KK09] 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.

- [LBOX12] 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.
- [LMMS⁺11] L. A. Lastras-Montaño, P. J. Meaney, E. Stephens, B. M. Trager, J. O'Connor, and L. C. Alves. A new class of array codes for memory storage. In *Information Theory and Applications Workshop (ITA)*, La Jolla, CA, February 2011.
- [LMS⁺10] 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.
- [LSZ09] 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.
- [Lub02] M. Luby. LT codes. In *IEEE Symposium on Foundations of Computer Science*, 2002.
- [MS77] 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.
- [OCLN12] 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.
- [OD89] J. K. Ousterhout and F. Douglass. Beating the I/O bottleneck: A case for log-structured file systems. *Operating Systems Review*, 23(1):11–27, January 1989.
- [OJ10] 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.
- [Oni01] Onion Networks. Java FEC Library v1.0.3. Open source code distribution: <http://onionnetworks.com/fec/javadoc/>, 2001.
- [Par07] A. Partow. Schifra Reed-Solomon ECC Library. Open source code distribution: <http://www.schifra.com/downloads.html>, 2000-2007.
- [PBH13] J. S. Plank, M. Blaum, and J. L. Hafner. SD codes: Erasure codes designed for how storage systems really fail. In *FAST-2013: 11th Usenix Conference on File and Storage Technologies*, San Jose, February 2013.
- [PGM13] 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.
- [PH13] J. S. Plank and C. Huang. Tutorial: Erasure coding for storage applications. Slides presented at FAST-2013: 11th Usenix Conference on File and Storage Technologies, February 2013.
- [Pla97] 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.
- [PLS⁺09] 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.

- [PW72] W. W. Peterson and E. J. Weldon, Jr. *Error-Correcting Codes, Second Edition*. The MIT Press, Cambridge, Massachusetts, 1972.
- [REG⁺03] 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.
- [RP11] 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.
- [SBMS93] 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.
- [SDG10] 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.
- [SG07] 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.
- [SGMV08] 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.
- [SGMV09] 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.
- [SR10] 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.
- [WDB10] 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.
- [WUA⁺08] 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.
- [WWK08] 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.
- [Xu05] 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.
- [XXLC10] 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.