Optimizing Cauchy Reed-Solomon Codes for Fault-Tolerant Storage Applications

James S. Plank

plank@cs.utk.edu

Technical Report CS-05-569
Department of Computer Science
University of Tennessee

December, 2005.

This paper has been submitted for publication.
See the web link below for current publication status.

http://www.cs.utk.edu/˜plank/plank/papers/CS-05-659.html

Optimizing Cauchy Reed-Solomon Codes for Fault-Tolerant Storage Applications

James S. Plank*
Department of Computer Science
University of Tennessee
Knoxville, TN 37996.

Abstract

In the past few years, all manner of storage systems, ranging from disk array systems to distributed and wide-area systems, have started to grapple with the reality of tolerating multiple simultaneous failures of storage nodes. Unlike the single failure case, which is optimally handled with RAID Level-5 parity, the multiple failure case is more difficult because optimal general purpose strategies are not yet known.

Erasure Coding is the field of research that deals with these strategies, and this field has blossomed in recent years. Despite this research, the decades-old strategy of Reed-Solomon coding remains the only space-optimal (MDS) code for all but the smallest storage systems. The best performing implementations of Reed-Solomon coding employ a variant called *Cauchy Reed-Solomon coding*, developed in the mid 1990's [BKK$^+$95].

In this paper, we present an improvement to Cauchy Reed-Solomon coding that is based on optimizing the Cauchy distribution matrix. We detail an algorithm for generating good matrices and then evaluate the performance of encoding using all manners of Reed-Solomon coding, plus the best MDS codes from the literature. The improvements over the original Cauchy Reed-Solomon codes are as much as 83% in realistic scenarios, and average roughly 10% over all cases that we tested.

1 Introduction

Erasure codes have profound uses in settings that involve multiple storage nodes. These include disk array systems, wide-area storage platforms, peer-to-peer storage platforms, and grid storage platforms. An erasure code may be defined as follows.

> We are given n storage nodes with B bytes of data each. To these, we add m storage nodes, also with B bytes of storage capacity. Any of these nodes may fail, which results in its storage being inaccessible. Node failures are recognized by the storage system and are termed *erasures*.
>
> An erasure code defines how to encode the Bn bytes of data on the collection of $n+m$ nodes such that upon failure of up to m nodes from the collection, the Bn bytes of data may be recalculated from the non-failed nodes.

Erasure codes have been employed for fault-tolerance and improved performance in single-site [FMS$^+$04, GWGR04, Wil06], archival [RWE$^+$01], wide-area [ASP$^+$02, CP05, XC05] and peer-to-peer storage systems [ZL02, Li04, LCL04, DLLF05]. They have additional uses in content distribution systems [BLMR98, Mit04]. As the number of components in these systems grow and as they continue to employ failure-prone interconnection networks, the need for erasure codes will continue to grow in the future.

There are three dimensions of performance of an erasure code:

1. **Space overhead.** Space overhead may be evaluated in one of two ways – either by the number of coding nodes required to achieve a baseline of fault-tolerance [Haf05a, Haf05b], or by the average number of failures tolerated by a given number of coding nodes [LMS$^+$97, WK03, PT04]. Regardless of the evaluation methodolgy, space optimality may be achieved when the number of coding nodes is equal to the number of failures tolerated. These codes are called *Maximum Distance Separable (MDS)* codes and are clearly desirable.

2. **Encoding performance.** This is the time complexity of creating the m coding nodes from the n

[*] plank@cs.utk.edu, 865-974-4397, fax: 865-974.4404. Approximate word count: 5500. This work has been cleared through the author's institution. This material is based upon work supported by the National Science Foundation under grants CNS-0437508, EIA-0224441 and ACI-0204007.

data nodes. A related metric is the **update performance** of a code, which is the number of coding nodes that must be updated when a data node is updated.

3. **Decoding performance.** This is the time complexity of recreating data from the surviving data and coding nodes.

In this paper, we focus solely on the encoding (and update) performance of MDS codes. Decoding performance is a far more complex problem, and will be addressed in future work.

The Current State of the Art

We focus first on MDS codes. When $n = 1$, replication is a trivially time optimal MDS code. When $n > 1$, any MDS code must perform at least $n - 1$ arithmetic operations per coding block, and any update to a piece of data must require at least m operations, one per coding node [XB99]. The typical operation is bitwise exclusive-or (XOR), which is extremely fast on most machines. A second operation is Galois Field multiplication, which is more expensive than XOR.

When $m = 1$, RAID Level-5 parity [CLG$^+$94] is an MDS code that performs $n - 1$ XORs for encoding. Thus, it is time optimal and is pervasive as the main coding technique in disk array systems. In 1995, a new parity-based code called EVENODD coding was presented as the first MDS code for $m = 2$ that relies solely on parity operations [BBBM95]. This code was recently extrapolated to $m = 3$ in the STAR code [HX05]. Neither code is time-optimal; however, both are close.

In 1999, the X-Code [XB99] was presented as a time optimal MDS code for $m = 2$, $n + 2$ prime. This code has additional significance as the first *vertical* parity code, requiring all storage nodes to hold both data and coding information, as opposed to *horizontal* parity codes which partition the storage nodes into exclusively holding either data or coding information.

Apart from these codes, the only MDS codes are Reed-Solomon codes, which have existed for decades [MS77, PW72, WB94]. Reed-Solomon codes are very powerful as they can be defined for any value of n and m. However, they have a drawback of requiring n Galois Field multiplications per coding block, and since coding blocks are typically smaller than a machine's word size, they can require $2n$ to $8n$ multiplications per machine word. Thus, Reed-Solomon codes are expensive. However, they remain the only MDS coding alternative in a large number of storage applications [LS00, RWE$^+$01, CP05].

In 1995, Blomer *et al* presented two important performance improvements to Reed-Solomon codes, termed *Cauchy Reed-Solomon (CRS)* coding [BKK$^+$95]. The first improvement converts all encoding operations to XORs, so that encoding takes $O(n\log_2(m+n))$ XORs per coding block. The second improvement is the use of a Cauchy distribution matrix rather than the standard Vandermonde distribution matrix [PD05], which improves the performance of matrix inversion for decoding. To date, CRS coding is the state of the art for general MDS erasure coding.

Since MDS codes can be expensive, recent research has relaxed space optimality in order to improve performance. Following a landmark paper in 1997 [LMS$^+$97], Low-Density Parity-Check (LDPC) codes have been developed as important alternatives to MDS codes. Tornado codes [LMS$^+$97, BLM99], IRA codes [JKM00], LT codes [Lub02] and Raptor codes [Sho03] all encode with a constant number of XORs per coding block, which is a factor of n *better* than the time optimality criterion defined above. This comes at a cost in space, however. Specifically, given m coding nodes, LDPC codes can only tolerate an average of m/f failures, where f is an *overhead factor*, whose minimum (and optimal) value is one.

While LDPC codes are asymptotically MDS (i.e. $f \to 1$ as $n \to \infty$), they have significant space overhead penalties for the values of n and m that many storage applications require. For example, when $n = 50$ and $m = 50$, the best known LDPC code tolerates an average of 36.025 node failures [CP05]. When the ratio of networking performance to CPU speed is high enough, LDPC codes outperform their MDS alternatives. However, when that ratio is lower, MDS codes perform better [PT04, CP05].

A second class of non-MDS codes are the recently-developed HoVer and WEAVER codes [Haf05a, Haf05b]. HoVer codes are a combination of horizontal and vertical codes for small m that have time-optimal characteristics, but are not MDS. WEAVER codes are vertical codes that are also time-optimal, and tolerate larger numbers of failures (up to 12). However, they are only MDS in limited cases ($n = 2, m = 2$ and $n = 3, m = 3$).

The Contribution of This Paper

This paper improves the encoding performance of Cauchy Reed-Solomon codes, and by so doing improves the state of the art in MDS erasure codes. As mentioned above, CRS coding employs a Cauchy distribution matrix to perform encoding (and upon failure, decoding). Any Cauchy matrix will suffice, and the number of Cauchy matrices for given values of n and m is exponential in n and m. The original work on CRS coding treats all Cauchy matrices as equivalent

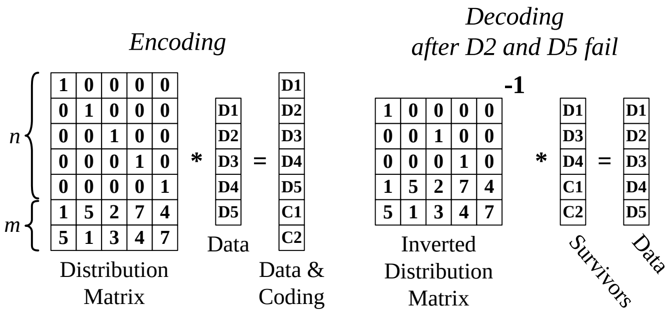
	0	1	2	3	4	5	6	7
0	0	1	2	3	4	5	6	7
1	1	0	3	2	5	4	7	6
2	2	3	0	1	6	7	4	5
3	3	2	1	0	7	6	5	4
4	4	5	6	7	0	1	2	3
5	5	4	7	6	1	0	3	2
6	6	7	4	5	2	3	0	1
7	7	6	5	4	3	2	1	0

Addition

	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6	7
2	0	2	4	6	3	1	7	5
3	0	3	6	5	7	4	1	2
4	0	4	3	7	6	2	5	1
5	0	5	1	4	2	7	3	6
6	0	6	7	1	5	3	2	4
7	0	7	5	2	1	6	4	3

Multiplication

Figure 1: Addition and multiplication tables for $GF(2^3)$.



$$\begin{bmatrix}1&0&0&0&0\\0&1&0&0&0\\0&0&1&0&0\\0&0&0&1&0\\0&0&0&0&1\\1&5&2&7&4\\5&1&3&4&7\end{bmatrix} * \begin{bmatrix}D1\\D2\\D3\\D4\\D5\end{bmatrix} = \begin{bmatrix}D1\\D2\\D3\\D4\\D5\\C1\\C2\end{bmatrix}$$

$$\begin{bmatrix}1&0&0&0&0\\0&0&1&0&0\\0&0&0&1&0\\1&5&2&7&4\\5&1&3&4&7\end{bmatrix}^{-1} * \begin{bmatrix}D1\\D3\\D4\\C1\\C2\end{bmatrix} = \begin{bmatrix}D1\\D2\\D3\\D4\\D5\end{bmatrix}$$

Figure 2: An example of Reed-Solomon coding and decoding with $n = 5$ and $m = 2$ over $GF(2^3)$.

and specifies an arbitrary construction. The authors quantify the matrix's impact on performance as a factor of $O(\log_2(m + n))$ [BKK+95]. While this is true, big-O notation treats constant factors as equal, and in these applications, constant factors can have a significant performance impact. In this paper, we show that two Cauchy matrices for the same values of n and m can differ in performance by over 8Moreover, we give an algorithm for constructing Cauchy matrices that have excellent performance.

Additionally, we compare the performance of our Reed-Solomon coding to Cauchy Reed-Solomon coding as originally described [BKK+95], classical "Vandermonde" Reed-Solomon coding [Pla97], and the parity-based MDS codes [BBBM95, XB99, HX05, Haf05b]. As such, this paper provides a useful reference for the performance of various MDS codes.

2 Cauchy Reed-Solomon Coding

To understand the performance improvements in this paper, we must describe CRS coding in detail. See [BKK+95] for further details. To help with example calculations, in Figure 1 we give addition and multiplication tables for the Galois Field $GF(2^3)$. Note, addition is simply XOR. Multiplication is more complex, but for small fields such as $GF(2^3)$, a multiplication table suffices.

All Reed-Solomon coding employs the same methodology [Pla97]. There are n data words, which are represented in a column vector $D =< D_1, \ldots, D_n >$. D is multiplied by an $(n + m) \times n$ *distribution matrix*, whose first n rows are the identity matrix. The product is an $n + m$-element column vector $D|C$, where $C =< C_1, \ldots, C_m >$ represents the coding words. Thus, each row of the distribution matrix represents a storage device, which holds either data or coding words.

Decoding proceeds by deleting the rows of the distribution matrix that correspond to node failures, inverting the resulting matrix, and multiplying it by the surviving words in order to recalculate the lost data. The process is depicted in Figure 2.

The distribution matrix must have the property that all $n \times n$ submatrices are invertible. The classic definition of Reed-Solomon coding derives the distribution matrix from an $(n + m) \times n$ Vandermonde matrix over the Galois Field $GF(2^w)$, where $n + m \leq 2^w$ [PD05]. The word sizes are thus 2^w bits. Typical values of w are 4, 8 and 16, since these values allow one to break up 32 and 64 bit machine-sized words evenly into coding words.

CRS coding modifies this scheme in two ways. First, instead of using a Vandermonde matrix, CRS coding employs an $m \times n$ Cauchy matrix, again over $GF(2^w)$, where $n + m \leq 2^w$. However, with CRS, w can be selected to be as small as possible, rather than be limited to 4, 8 or 16. An $m \times n$ Cauchy matrix is defined as follows. Let $X = \{x_1, \ldots, x_m\}$ and $Y = \{y_1, \ldots, y_n\}$ be defined such that each x_i and y_i is a distinct element of $GF(2^w)$, and $X \cap Y = \emptyset$. Then the Cauchy matrix defined by X and Y has $1/(x_i + y_j)$ in element i, j. For example, in Figure 2, the last two rows of the distribution matrix make up the Cauchy matrix over $GF(2^3)$, where $X = \{1, 2\}$ and $Y = \{0, 3, 4, 5, 6\}$.

The distribution matrix composed of the identity matrix in the first n rows, and a Cauchy matrix in the remaining m rows has the desired property that all $n \times n$ submatrices are invertible. It has an additional property that these submatrices may be inverted in $O(n^2)$ Galois Field operations [Rab89].

The second modification of CRS is to use projections that convert the operations over $GF(2^w)$ into XORs. These work as follows. Each element e of $GF(2^w)$ may be represented by a $1 \times w$ column vector of bits, $V(e)$. This vector is equivalent to the standard binary representation of the element. Each element e of $GF(2^w)$ may also be represented by a $w \times w$ matrix of bits, $M(e)$, where the i-th column of $M(e)$ is equal to the column

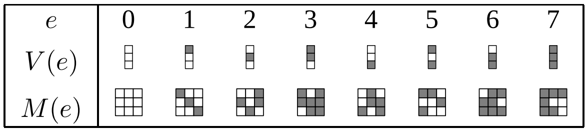
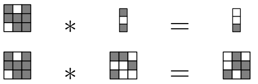


Figure 3: Vector and matrix representation of the elements of $GF(2^3)$.

vector $V(e2^{i-1})$. Continuing our example from Figure 2, in Figure 3, we show $V(e)$ and $M(e)$ for each element e over $GF(2^3)$.

An important property of these projections is that using standard bit arithmetic (addition is XOR, multiplication is bitwise-and), $M(e_1) * V(e_2) = V(e_1e_2)$, and $M(e_1) * M(e_2) = M(e_1e_2)$. For example, in $GF(2^3)$:

$$3 \quad * \quad 5 \quad = \quad 4$$



The distribution matrix is now converted to a $w(n + m) \times wn$ binary matrix by substituting $M(e)$ for e in the distribution matrix over $GF(2^w)$. Moreover, instead of partitioning the data into words of size 2^w, we partition each storage device's entire data space into w *packets*, where each packet's size (B/w bytes) must be a multiple of the machine's word size. Continuing our example from Figure 2, suppose each device holds 3 GB. Then, each device D_i and C_i will be partitioned into three 1 GB packets, $D_{i,1}$, $D_{i,2}$, and $D_{i,3}$ (or $C_{i,1}$, $C_{i,2}$, and $C_{i,3}$). Then the encoding and decoding processes may now be implemented over $GF(2)$ (bit arithmetic) rather than $GF(2^3)$. This is depicted in Figure 4, for the example scenario of Figure 2 (we omit the identity matrix to save space).

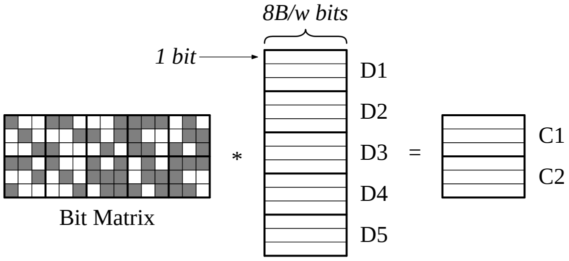


Figure 4: The Reed-Solomon coding scenario of Figure 2, converted to use bit arithmetic.

Since the arithmetic of this new matrix is over $GF(2)$, we can calculate the coding packets using only XOR operations. Specifically, to calculate $C_{i,j}$, we take the XOR of all packets $D_{i',j'}$ such that the bit corresponding to $D_{i',j'}$ in $C_{i,j}$'s row of the matrix is one. For example, in Figure 4:

$$\begin{aligned} C_{1,1} &= D_{1,1} \oplus D_{2,1} \oplus D_{2,2} \oplus D_{3,3} \oplus \\ &\quad D_{4,1} \oplus D_{4,2} \oplus D_{4,3} \oplus D_{5,2}. \end{aligned}$$

Let o be the average number of ones per row in the distribution matrix. Then the number of XORs to produce a word in each coding packet is equal to $o - 1$.

For example, in the distribution matrix of Figure 4, there are 47 ones. Since there are six rows, $o = 47/6$, and thus the average number of XORs per coding word is $o - 1 = 47/6 - 1 = 6.83$. Compared to standard Reed-Solomon coding, where each coding word would require 4 XORs plus 20 multiplications over $GF(2^8)$, (or 40 multiplications over $GF(2^4)$), this is an improvement indeed, and is why, for example, OceanStore [RWE$^+$01] uses Cauchy Reed-Solomon coding for their erasure coding.

3 All Cauchy Matrices Are Not Equal

In [BKK$^+$95], the performance of CRS is reported to be $O(n\log(n + m))$ per coding word. This is because o is $O(w)$, and w is $O(\log(n+m))$. Since all Cauchy matrices have the property that o is $O(w)$, the authors give an arbitrary Cauchy matrix construction: X equals the first m elements of $GF(2^w)$ and Y equals the next n elements. For our example scenario where $n = 5$ and $m = 2$, this yields the matrix in Figure 5, which has 54 ones, as opposed to the 47 ones when $X = \{1, 2\}$ and $Y = \{0, 3, 4, 5, 6\}$.

5	6	7	2	3
6	5	2	7	4

Figure 5: The Cauchy matrix defined in [BKK$^+$95] for $n = 5$ and $m = 2$: $X = \{0, 1\}$ and $Y = \{2, 3, 4, 5, 6\}$.

The impact on performance is significant. In this example, the matrix in Figure 5 requires $54/6 - 1 = 8$ XORs per word, or a 17% decrease in performance over the matrix in Figure 2. This observation fuels the exploration in the remainder of the paper.

4 Enumerating Optimal Cauchy Matrices

The simplest way to discover optimal Cauchy Matrices is to enumerate them exhaustively. Given n, m, and w,

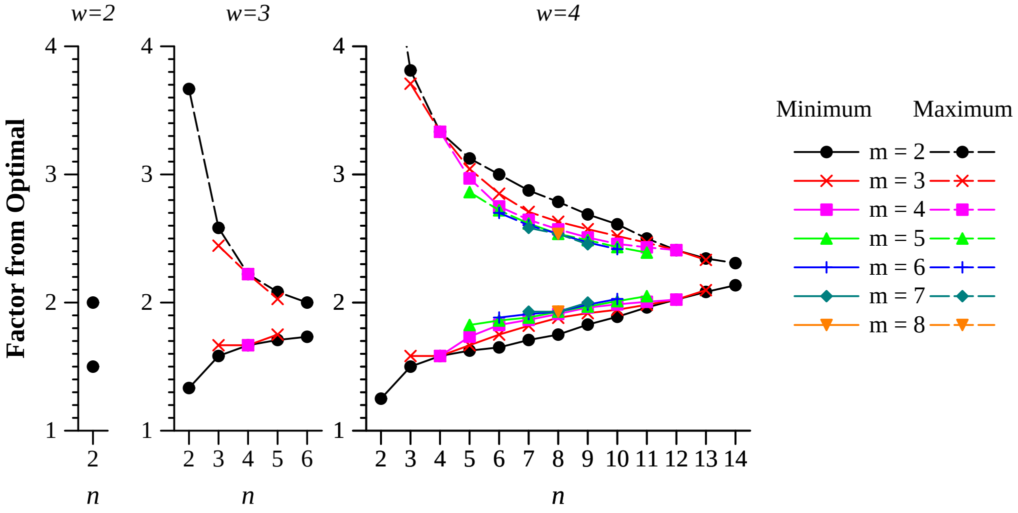


Figure 6: Minimum and maximum Cauchy matrices for $w \leq 4$ (i.e., $n+m \leq 16$).

the number of ways to partition the 2^w elements into the sets X and Y is:

$$\binom{2^w}{n+m}\binom{n+m}{n},$$

which is clearly exponential in n and m. However, for $w \leq 4$, and in 107 of the 225 possible combinations of n and m when $w = 5$, we have enumerated all Cauchy matrices, and determined the best and worst distribution matrices.[1] We plot the results for $w \leq 4$ in Figure 6. Instead of plotting number of ones, or number of XORs, we plot the factor from optimal coding, where optimal is defined as $n - 1$ XORs per coding word [XB99, Haf05a]. Thus, for example, our exploration shows that the Cauchy matrix of Figure 2 indeed has the minimal number of ones. Since that matrix requires 6.83 XORs per coding word, and optimal coding would require 4, its factor is $6.83/4 = 1.71$, which is plotted in the rightmost graph of Figure 6 at $n = 5$, $m = 2$.

There are three features of Figure 6 worth mentioning. First, there is a significant difference in the performance of the minimum and maximum Cauchy matrices for these values. This difference is most pronounced when n is small, because that is when there is a greater variety of possible values in the Cauchy matrix. Second, the performance of CRS coding gets worse as n grows. This is to be expected, again because as the Cauchy matrix grows, it must contain more values from $GF(2^w)$. The elements of $GF(2^w)$ vary in their number of ones, from exactly w (element 1) to close to w^2. Therefore, small matrices can be populated with elements that have have $O(w)$ ones. The larger matrices must include elements with $O(w^2)$ ones, and thus they perform worse.

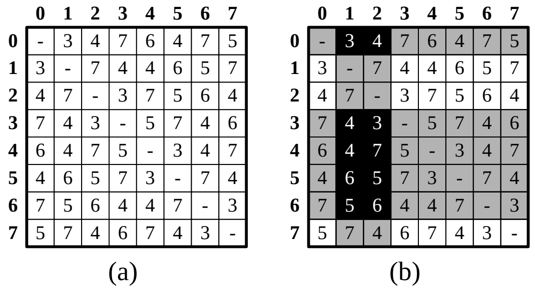
The third feature is perhaps unexpected. This is that for the same values of n and m, the best matrices for $w = 4$ perform *better* than those for $w = 3$ and $w = 2$. For example, consider $n = 2, m = 2$. When $w = 2$, the best matrix has 10 ones, which means $10/4 - 1 = 1.5$ XORs per coding word. When $w = 3$, the best matrix has 14 ones, which means 1.33 XORs per coding word, and when $w = 4$, the best matrix has 18 ones, which means 1.25 XORs per coding word. We will explore this phenomenon further in Section 7.

5 Generating Good Cauchy Matrices for Larger w

For larger w, it is impractical to use exhaustive search to find optimal Cauchy matrices. Therefore, we have developed the following algorithm to construct good Cauchy matrices. We call the matrices that it produces GC matrices (for "Good Cauchy"), and parameterize GC with n, m, and w. The $GC(n, m, w)$ matrices where $n = m$ are optimal in all cases that we have corroborated by enumeration. When $n \neq m$, some $GC(n, m, w)$ matrices are slightly worse than optimal. We measure this effect below.

To construct a $GC(m, n, w)$ matrix, we first construct a $2^w \times 2^w$ matrix $ONES(w)$. $ONES(w)_{i,j}$ contains the number of ones in the bit matrix $M(1/(i+j))$. Obviously, $ONES(w)_{i,i}$ is always undefined. The matrix $ONES(3)$ is shown in Figure 7(a).

[1] While this is roughly half of the combinations of n and m for $w = 5$, it is only 3.7% of the work required to calculate all of the combinations of n and m. We are continuing to enumerate optimal matrices for the remainder of these cases.



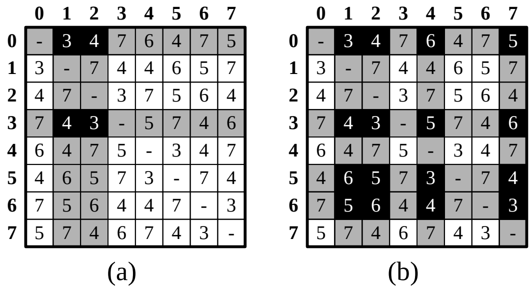
(a)

	0	1	2	3	4	5	6	7
0	-	3	4	7	6	4	7	5
1	3	-	7	4	4	6	5	7
2	4	7	-	3	7	5	6	4
3	7	4	3	-	5	7	4	6
4	6	4	7	5	-	3	4	7
5	4	6	5	7	3	-	7	4
6	7	5	6	4	4	7	-	3
7	5	7	4	6	7	4	3	-

(b)

	0	1	2	3	4	5	6	7
0	-	3	4	7	6	4	7	5
1	3	-	7	4	4	6	5	7
2	4	7	-	3	7	5	6	4
3	7	4	3	-	5	7	4	6
4	6	4	7	5	-	3	4	7
5	4	6	5	7	3	-	7	4
6	7	5	6	4	4	7	-	3
7	5	7	4	6	7	4	3	-

Figure 7: (a): $ONES(3)$. (b): The optimal Cauchy matrix for $n = 5$, $m = 2$, $w = 3$.



(a)

	0	1	2	3	4	5	6	7
0	-	3	4	7	6	4	7	5
1	3	-	7	4	4	6	5	7
2	4	7	-	3	7	5	6	4
3	7	4	3	-	5	7	4	6
4	6	4	7	5	-	3	4	7
5	4	6	5	7	3	-	7	4
6	7	5	6	4	4	7	-	3
7	5	7	4	6	7	4	3	-

(b)

	0	1	2	3	4	5	6	7
0	-	3	4	7	6	4	7	5
1	3	-	7	4	4	6	5	7
2	4	7	-	3	7	5	6	4
3	7	4	3	-	5	7	4	6
4	6	4	7	5	-	3	4	7
5	4	6	5	7	3	-	7	4
6	7	5	6	4	4	7	-	3
7	5	7	4	6	7	4	3	-

Figure 8: Balanced Cauchy matrices for $w = 3$. (a): $n = m = 2$, (b): $n = m = 4$.

We may define a Cauchy matrix by selecting m columns, $X_1, \ldots X_m$, and n rows, $Y_1, \ldots Y_n$, of $ONES(w)$, such that no $X_j = Y_i$. We define the weight, $W(w, X, Y)$ of a Cauchy matrix to be:

$$W(w, X, Y) = \sum_{i=1}^{n} \sum_{j=1}^{m} ONES(w)_{Y_i, X_j}.$$

The weight is equal to the number of ones in the Cauchy distribution matrix, and thus may be used to measure the encoding performance of the matrix. For example, in Figure 7(b), we show the Cauchy matrix of Figure 2, where $X = \{1, 2\}$ is represented by the shaded columns, and $Y = \{0, 3, 4, 5, 6\}$ is represented by the shaded rows. The weight of this Cauchy matrix is equal to the sum of the black squares, 47, which indeed is the number of ones in the matrix.

Our goal, therefore is to define X and Y such that $W(w, X, Y)$ is minimal or close to minimal. First, note that $ONES(w)$ has an extremely high degree of symmetry. There are only 2^w values in $ONES(w)$, which correspond to the number of ones in $M(1/e)$ for each element $e \in GF(2^w)$. Each of these values occurs exactly once in each row of $ONES(w)$ and in each column of $ONES(w)$. Moreover, when $n = m$ is a power of two, it is possible to choose X and Y such that

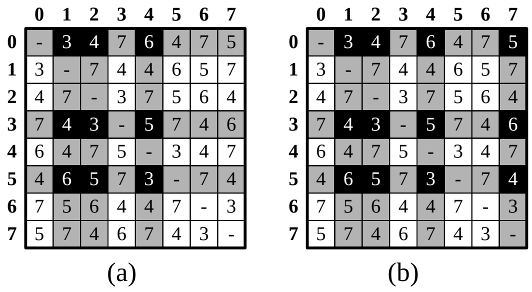
$$W(w, n, m) = n \sum_{i=1}^{n} ONES(w)_{Y_i, X_1}.$$

In other words, for each column X_j of $ONES(w)$, the values where X_j intersects Y are the same as the values where the first column X_1 intersections Y. They are simply permuted. We call such a Cauchy matrix a *balanced* Cauchy matrix. We show two such matrices for $w = 3$ in Figure 8.

We now define $GC(n, n, w)$ where n is a power of two. Obviously, $2n$ must be less than or equal to 2^w. For $w \geq 2$, $GC(2, 2, w)$ is the minimum-weight balanced Cauchy matrix with $X = \{1, 2\}$. For example, $GC(2, 2, 3)$ is pictured in Figure 8(a), and has a weight of 14.

For $n > 2$, we construct $GC(n, n, w)$ to be the minimum-weight balanced Cauchy matrix which contains $GC(n/2, n/2, w)$. For example, $GC(4, 4, w)$ is pictured in Figure 8(b). That $GC(n, n, w)$ always exists is a simple proof, based on the symmetry of $ONES(w)$, which we omit for brevity.

We now define $GC(n, n, w)$ where n is not a power of two to be the minimum weight submatrix of $GC(n + 1, n + 1, w)$. Thus, we construct $GC(n, n, w)$ by constructing $GC(n+1, n+1, w)$, and deleting the row and column that results in a minimal weight matrix. Since n is not a power of two, we know that there is a value of $n' > n$ such that n' is a power of two and $2n' \leq 2^w$. Thus, $GC(n', n', w)$ exists, and it is possible to construct $GC(n, n, w)$ by constructing $GC(n', n', w)$, and iteratively deleting rows and columns until there are n rows and columns left. For example, $GC(3, 3, 3)$ is pictured in Figure 9(a), and is constructed by deleting row 6 and column 7 from $GC(4, 4, 3)$ (Figure 8(b)).



(a)

	0	1	2	3	4	5	6	7
0	-	3	4	7	6	4	7	5
1	3	-	7	4	4	6	5	7
2	4	7	-	3	7	5	6	4
3	7	4	3	-	5	7	4	6
4	6	4	7	5	-	3	4	7
5	4	6	5	7	3	-	7	4
6	7	5	6	4	4	7	-	3
7	5	7	4	6	7	4	3	-

(b)

	0	1	2	3	4	5	6	7
0	-	3	4	7	6	4	7	5
1	3	-	7	4	4	6	5	7
2	4	7	-	3	7	5	6	4
3	7	4	3	-	5	7	4	6
4	6	4	7	5	-	3	4	7
5	4	6	5	7	3	-	7	4
6	7	5	6	4	4	7	-	3
7	5	7	4	6	7	4	3	-

Figure 9: (a): $GC(3, 3, 3)$, (b): $GC(3, 4, 3)$.

Finally, we define $GC(n, m, w)$, where $n \neq m$ to be the minimum weight supermatrix of $GC(\min(n, m), \min(n, m), w)$. Suppose $n > m$. One constructs $GC(n, m, w)$ by first constructing $GC(m, m, w)$, then sorting the weights of the rows that can potentially be added to $GC(m, m, w)$ to create $GC(n, m, w)$, and then adding the $n - m$ smallest of these rows. The construction when $m > n$ is analogous, except columns are added rather than rows. For example, $GC(3, 4, 3)$ is pictured in Figure 9(b), and is constructed by adding column 7 (rather than column 6) to $GC(3, 3, 3)$.

The running time complexity of constructing $GC(n, m, w)$ is a very detailed calculation, which is outside the scope of this paper. However $O(2^{2w+1})$ is a succinct upper bound. While that is exponential in n and m (since $n + m \leq 2^w$), it grows much more slowly that the number of possible Cauchy matrices, detailed in Section 4, and allows us to construct good matrices for larger values of n and m.

6 Performance of GC Matrices

Our first evaluation of GC matrices is to compare them to the best Cauchy matrices generated from our exhaustive search. All GC matrices for $w \leq 3$ are optimal Cauchy matrices. For $w = 4$ and $w = 5$, the GC matrices are all optimal when $n = m$. Overall, in the 166 cases where we were able to determine the optimal Cauchy matrix, 53 of them matched the GC matrix. In the other 113 cases, the maximum performance difference was for $GC(10, 2, 5)$, which differed by 7.9% from the optimal matrix in the number of XORs per coding word. On average, the performance difference between the GC matrix and the optimal matrix over all cases was 1.78%.

In terms of their performance as n and m grow, we present two studies – one for small m, and one where n and m both grow. In both studies, we compare the following MDS coding techniques:

- **WEAVER**: There are two MDS WEAVER codes [Haf05b] — one for $m = 2, n = 2$, and one for $m = 3, n = 3$. Both are optimal in performance.

- **X-Code**: The optimal X-Code [XB99] is defined for $m = 2$ and $n + 2$ prime .

- **EVENODD**: This is defined for $m = 2$ and all n [BBBM95]. Its performance is slightly worse than optimal.

- **STAR**: This is an extrapolation of EVENODD coding for $m = 3$ [HX05].

- **CRS Coding (GC)**: This uses the matrices defined above for all values of w between 2 and 10, and selects the one that performs the best.

- **CRS Coding (Original)**: This uses the original matrix construction as defined in [BKK$^+$95], where X consists of the first m elements in the field, and Y consists of the next n elements.

- **CRS Coding (BC)**: This uses BC, or "Bad Cauchy" matrices, by employing the GC algorithm, but starting with columns 1 and 3, and finding maximum weight matrices rather than minimum weight matrices.

- **Standard RS Coding**: This uses distribution matrices based on the Vandermonde matrix, and arithmetic over $GF(2^w)$ as outlined in [PD05, Pla97, Riz97].

The metric for comparison is the factor of optimal coding, as in Figure 6. For the XOR-based codes (all but standard Reed-Solomon coding), this is the number of XORs per coding word, divided by $n - 1$. As noted above, the X-Code and the two WEAVER codes attain this bound.

Standard Reed-Solomon coding uses Galois Field multiplication in addition to XOR. To enable a comparison of it to the XOR-based codes, we measured the bandwidth of XOR operations ($B_{\oplus}$), and of multiplication in $GF(2^8)$ (B_*), which covers values of $n + m \leq 256$. For maximum performance, we implemented multiplication using a 256×256 multiplication table. This is faster than either using log and anti-log tables (as in [Pla97]), or than simulating polynomial arithmetic over $GF(2)$ using XOR and bit-shifting [PW72]. This was done on a Dell Precision Workstation with a 3.40 GHz Intel Pentium 4 processor. The measurements are below:

$B_{\oplus}$	B_*
2992 MB/s	787.9 MB/s

Note, we measure both in terms of their bandwidth (megabytes per second), which accounts for the fact that XORs may be done over 32-bit words, while multiplication over $GF(2^8)$ operates on 8-bit quantities.

Reed-Solomon coding requires n multiplications and $n - 1$ XORs per coding word. Thus, we calculate the factor of Reed-Solomon coding as:

$$\frac{\left(\frac{n-1}{B_{\oplus}} + \frac{n}{B_*}\right)}{\left(\frac{n-1}{B_{\oplus}}\right)}.$$

The results for small m are in Figure 10. Handling small numbers of failures is the most common case for disk controllers and medium-scale storage systems. The most glaring feature of these graphs is that the special-purpose codes (EVENODD, STAR, X-Code, WEAVER) drastically outperform the Reed-Solomon

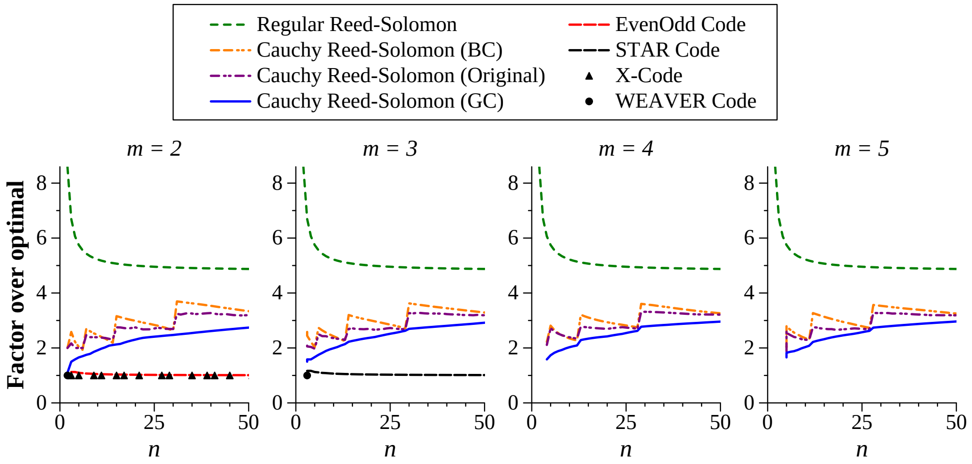


Figure 10: Performance comparison of MDS codes for $2 \leq m \leq 5$.

codes. Thus, in applications which need reslience to two and three failures, these should be used in all cases. Note, this is not a new result; it simply reaffirms the original research on special-purpose codes, and fuels the search for good MDS codes for higher values of m.

Focusing solely on the Reed-Solomon codes, we draw a few conclusions from Figure 10. First, Cauchy Reed-Solomon coding in all cases outperforms standard Reed-Solomon coding. Although it appears that the two techniques will converge as n grows larger, it must be noted that when $n + m$ becomes greater than 256, standard Reed-Solomon coding must use multiplication over $GF(2^{16})$, which is much slower than over $GF(2^8)$ (we measured B_* = 148.5 MB/sec).

Second, not only do the GC matrices outperform the other constructions, but their performance decreases gradually as n increases, rather than exhibiting jumps at the points where $n + m$ crosses a power of two. We illuminate this as follows. If one holds n and m constant and increases w, the range of weights of Cauchy matrices (and therefore factors over optimal) increases drastically; however, the minimum factors stay roughly the same. For example, in Table 1, we show the weights of the GC and BC matrices for $m = 3$, $n = 29$, and w ranging from 5 to 10. Note then when $w = 5$, the difference between the GC and BC matrices is slight, whereas when $w = 10$, the difference is more than a factor of two. This means that when a value of w becomes unusable (for example, when $m = 3$ and $n = 30$, one cannot use $w = 5$), there is far less of a performance penalty in using the GC matrix for the next value of w than using the BC matrix. The "Original" matrices split the difference between the two.

Our second case study is for larger values of n and m, which is applicable to wide-area storage systems and content distribution systems. In Figure 11, we show the performance of the Reed-Solomon codes for three rates $R = \frac{n}{n+m}$: $\frac{1}{2}$ $(m = n)$, $\frac{2}{3}$ $(2m = n)$, and $\frac{4}{5}$ $(4m = n)$. These are rates that are popular in coding studies and implementations [LMS+97, WK03, PT04]. For example, OceanStore has employed (n, m) pairs of $(64, 16)$, $(32, 32)$ and $(16, 16)$ in various installations. For these values of n and m, Reed-Solomon coding is the only MDS coding technique.

	$GC(3, 29, w)$		$BC(3, 29, w)$	
	Weight	Factor	Weight	Factor
5	1118	2.63	1154	2.71
6	1370	2.68	1854	3.64
7	1666	2.80	2680	4.52
8	2162	3.18	3470	5.13
9	2303	3.01	4579	6.02
10	2750	3.24	5749	6.81

Table 1: Weights and factors of GC and BC matrices for $m = 3$, $n = 29$, and w ranging from 5 to 10.

The only real difference between Figures 10 and 11 is that the GC matrices for $R = \frac{1}{2}$ exhibit minor performance jumps when $n + m$ crosses a power of two.

With respect solely to Cauchy Reed-Solomon coding, the GC matrices are a significant improvement over the others in nearly all cases. This effect is summarized in Table 2, which shows the maximum, minimum and average performance improvement of GC matrices versus the original constructions. The greatest improvements come for small values of n, which are the most frequently implemented cases. The smallest improvements typically come when $n+m$ equals a power of two. In terms of averages, the GC matrices show roughly a 10% improvement over the original constructions in all

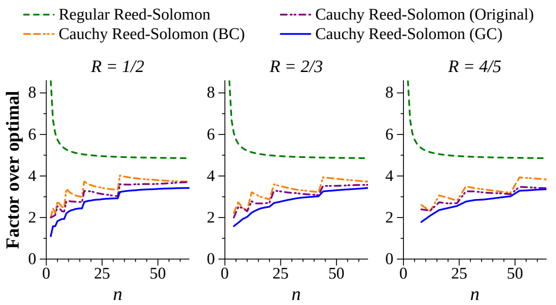


Figure 11: Performance of Reed-Solomon coding for higher values of n and m

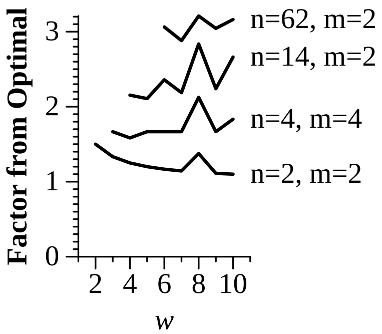


Figure 12: Four cases where the best performance occurs at a non-minimum value of w.

cases.

Test	Maximum	Minimum	Average
$m = 2$	81.8% ($n = 2$)	6.1% ($n = 100$)	17.3%
$m = 3$	42.9% ($n = 6$)	1.2% ($n = 61$)	11.3%
$m = 4$	56.8% ($n = 5$)	1.8% ($n = 60$)	10.8%
$m = 5$	51.4% ($n = 5$)	1.2% ($n = 59$)	9.4%
$R = \frac{1}{2}$	81.8% ($n = 2$)	3.7% ($n = 32$)	12.9%
$R = \frac{2}{3}$	42.9% ($n = 6$)	1.7% ($n = 42$)	11.3%
$R = \frac{4}{5}$	34.0% ($n = 8$)	1.6% ($n = 64$)	10.1%

Table 2: The improvement in performance using GC matrices, as opposed to the original Cauchy construction.

7 Larger w Can Perform Better Than Smaller w

Recall Figure 6 from Section 4. This figure illustrates a point that at first seems counter-intuitive: the factor for $m = 2$, $n = 2$ improves when w grows from two to three to four. Probing further, we find that in 52 of the cases that we studied, the minimum factor occurs at a value of w that is not the smallest possible. Most are when $m = 2$. We illustrate four such cases in Figure 12. In each of the four cases, the factor does not decrease or increase consistently, but instead jumps around as w increases.

We make two remarks about this phenomenon. First, it should not be counter-intuitive. For example, consider $m = 2, n = 2$, and the Cauchy matrix where $X = \{1, 2\}$ and $Y = \{0, 3\}$. This Cauchy matrix has two distinct elements: 1 and 1/2. The element 1 has exactly w ones. The element 1/2 has $w + z - 2$ ones, where z is the number of non-zero coefficients in the primitive polynomial for $GF(2^w)$ [PW72]. For example, for $2 \leq w \leq 7$ and $9 \leq w \leq 11$, z is three. For $w = 8$ and $12 \leq w \leq 14$, z is five. Thus, the weight of the Cauchy matrix is $4w + 2z - 4$, and its factor over optimal is $\frac{4w+2z-4}{2w} - 1 = 1 + (z - 2)/w$, which clearly approaches one as w grows. Additionally, this explains why the factor goes up when $w = 8$ in Figure 12 (since $z = 5$ for $w = 8$, and $z = 3$ for the other w's).

w	Average Weight
6	11.700000 = 1.95 * 6
7	12.700000 = 1.81 * 7
8	17.550000 = 2.19 * 8
9	14.900000 = 1.66 * 9

Table 3: The average weight of the 20 minimum-weight elements for $GF(2^6)$ through $GF(2^9)$.

Put another way, while an average element of $GF(2^w)$ does indeed have $O(w^2)$ ones, the elements with the fewest ones have $O(w)$ ones, which means that if Cauchy matrices can be made from them, they should perform well. It is interesting to note the average weight of the 20 minimum-weight elements of $GF(2^w)$ for $6 \leq w \leq 9$. These are tabulated in Table 3. Because of the extra elements in the primitive polynomial, the average weight of the 20 minimum-weight elements for $w = 8$ is proportionally higher than the others, and this effect is reflected in the poor factors for $w = 8$ in Figure 12.

8 Available Resources

In [Pla05], we enumerate the optimal and GC matrices generated for this paper. We do this as a service to the community so that researchers and systems programmers who want to use the best variants of Cauchy Reed-Solomon codes may do so.

9 Conclusions and Future Work

In this paper, we have shown that the construction of the distribution matrix in Cauchy Reed-Solomon coding impacts the encoding performance. In particular, our desire is to construct Cauchy matrices with a minimal number of ones. We have enumerated optimal matrices for small cases, and given an algorithm for constructing good matrices in larger cases. The performance difference between good and bad matrices is significant, averaging roughly 10% across all cases, with a maximum of 83% in the best case. The work is significant, because for $m > 3$, these are the best MDS codes currently known.

Additionally, we have put the performance of Cauchy Reed-Solomon coding into perspective, comparing its performance to standard Reed-Solomon coding, and to special-purpose MDS algorithms for small numbers of failures. The main conclusion to draw here is that the special-purpose algorithms vastly outperform Reed-Solomon coding, and that more research should be performed on broadening these algorithms for larger numbers of failures. The recent work on HoVer [Haf05a] and WEAVER [Haf05b] codes are promising in this direction.

In this work, we have not studied decoding performance, nor have we included non-MDS codes for comparison. Both are topics for the future.

Finally, we note the rather counter-intuitive result that Cauchy Reed-Solomon coding can perform *better* for larger values of w while holding the other parameters constant. This is because larger Galois Fields may have more elements with proportionally fewer ones than smaller Galois Fields. It is a subject of future work to explore this phenomenon and construct Cauchy matrices for large fields that perform well. $w = 28$ and $w = 29$ are interesting candidates here, as they both have primitive polynomials with only three non-zero coefficients.

References

[ASP+02] S. Atchley, S. Soltesz, J. S. Plank, M. Beck, and T. Moore. Fault-tolerance in the network storage stack. In *IEEE Workshop on Fault-Tolerant Parallel and Distributed Systems*, Ft. Lauderdale, FL, April 2002.

[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.

[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.

[BLM99] J. W. Byers, M. Luby, and M. Mitzenmacher. Accessing multiple mirror sites in parallel: Using tornado codes to speed up downloads. In *IEEE INFOCOM*, pages 275–283, New York, NY, March 1999.

[BLMR98] J. Byers, M. Luby, M. Mitzenmacher, and A. Rege. A digital fountain approach to reliable distribution of bulk data. In *ACM SIGCOMM '98*, pages 56–67, Vancouver, August 1998.

[CLG+94] P. M. Chen, E. K. Lee, G. A. Gibson, R. H. Katz, and D. A. Patterson. RAID: High-performance, reliable secondary storage. *ACM Computing Surveys*, 26(2):145–185, June 1994.

[CP05] R. L. Collins and J. S. Plank. Assessing the performance of erasure codes in the wide-area. In *DSN-05: International Conference on Dependable Systems and Networks*, Yokohama, Japan, 2005. IEEE.

[DLLF05] L. Dairaine, J. Lacan, L. Lancérica, and J. Fimes. Content-access QoS in peer-to-peer networks using a fast MDS erasure code. *Computer Communications*, 28(15):1778–1790, September 2005.

[FMS+04] S. Frolund, A. Merchant, Y. Saito, S. Spence, and A. Veitch. A decentralized algorithm for erasure-coded virtual disks. In *DSN-04: International Conference on Dependable Systems and Networks*, Florence, Italy, 2004. IEEE.

[GWGR04] G. R. Goodson, J. J. Wylie, G. R. Ganger, and M. K. Reiter. Efficient byzantine-tolerant erasure-coded storage. In *DSN-04: International Conference on Dependable Systems and Networks*, Florence, Italy, 2004. IEEE.

[Haf05a] J. L. Hafner. HoVer erasure codes for disk arrays. Technical Report RJ10352 (A0507-015), IBM Research Division, July 2005.

[Haf05b] J. L. Hafner. WEAVER Codes: Highly fault tolerant erasure codes for storage systems. In *FAST-2005: 4th Usenix Conference on File and Storage Technologies*, San Francisco, December 2005.

[HX05] C. Huang and L. Xu. STAR: An efficient coding scheme for correcting triple storage node failures. In *FAST-2005: 4th Usenix Conference on File and Storage Technologies*, San Francisco, December 2005.

[JKM00] H. Jin, A. Khandekar, and R. McEliece. Irregular repeat-accumulate codes. In *2nd International Symposium on Turbo codes and Related Topics,*, Brest, France, September 2000.

[LCL04] W. K. Lin, D. M. Chiu, and Y. B. Lee. Erasure code replication revisited. In *PTP04: 4th International Conference on Peer-to-Peer Computing*. IEEE, 2004.

[Li04] J. Li. PeerStreaming: A practical receiver-driven peer-to-peer media streaming system. Technical Report MSR-TR-2004-101, Microsoft Research, September 2004.

[LMS+97] M. Luby, M. Mitzenmacher, A. Shokrollahi, D. Spielman, and V. Stemann. Practical loss-resilient codes. In *29th Annual ACM Symposium on Theory of Computing*, pages 150–159, El Paso, TX, 1997. ACM.

[LS00] W. Litwin and T. Schwarz. Lh*rs: a high-availability scalable distributed data structure using Reed Solomon codes. In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*, pages 237–248, 2000.

[Lub02] M. Luby. LT codes. In *IEEE Symposium on Foundations of Computer Science*, 2002.

[Mit04] M. Mitzenmacher. Digital fountains: A survey and look forward,. In *2004 IEEE Information Theory Workshop*, San Antonio, October 2004.

[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.

[PD05] J. S. Plank and Y. Ding. Note: Correction to the 1997 tutorial on reed-solomon coding. *Software – Practice & Experience*, 35(2):189–194, February 2005.

[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.

[Pla05] J. S. Plank. Enumeration of optimal and good Cauchy matrices for Reed-Solomon coding. Technical Report CS-05-570, University of Tennessee, December 2005.

[PT04] J. S. Plank and M. G. Thomason. A practical analysis of low-density parity-check erasure codes for wide-area storage applications. In *DSN-2004: The International Conference on Dependable Systems and Networks*, pages 115–124, Florence, Italy, June 2004. IEEE.

[PW72] W. W. Peterson and E. J. Weldon, Jr. *Error-Correcting Codes, Second Edition*. The MIT Press, Cambridge, Massachusetts, 1972.

[Rab89] M. O. Rabin. Efficient dispersal of information for security, load balancing, and fault tolerance. *Journal of the Association for Computing Machinery*, 36(2):335–348, April 1989.

[Riz97] L. Rizzo. Effective erasure codes for reliable computer communication protocols. *ACM SIGCOMM Computer Communication Review*, 27(2):24–36, 1997.

[RWE$^+$01] S. Rhea, C. Wells, P. Eaton, D. Geels, B. Zhao, H. Weatherspoon, and J. Kubiatowicz. Maintenance-free global data storage. *IEEE Internet Computing*, 5(5):40–49, 2001.

[Sho03] A. Shokrollahi. Raptor codes. Technical Report DR2003-06-001, Digital Fountain, 2003.

[WB94] S. B. Wicker and V. K. Bhargava. *Reed-Solomon Codes and Their Applications*. IEEE Press, New York, 1994.

[Wil06] W. Wilcke *et al*. The IBM intelligent brick project – petabytes and beyond. *IBM Journal of Research and Development*, to appear, April 2006.

[WK03] S. B. Wicker and S. Kim. *Fundamentals of Codes, Graphs, and Iterative Decoding*. Kluwer Academic Publishers, Norwell, MA, 2003.

[XB99] L. Xu and J. Bruck. X-Code: MDS array codes with optimal encoding. *IEEE Transactions on Information Theory*, 45(1):272–276, January 1999.

[XC05] H. Xia and A. A. Chien. RobuSTore: Robust performance for distributed storage systems. Technical Report CS2005-0838, University of California at San Diego, October 2005.

[ZL02] Z. Zhang and Q. Lian. Reperasure: Replication protocol using erasure-code in peer-to-peer storage network. In *21st IEEE Symposium on Reliable Distributed Systems (SRDS'02)*, pages 330–339, October 2002.