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

James S. Plank*
Department of Computer Science
University of Tennessee
Knoxville, TN 37996
plank@cs.utk.edu

Lihao Xu
Department of Computer Science
Wayne State University
Detroit, MI 48202
lihao@cs.wayne.edu

Abstract

In the past few years, all manner of storage applications, 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 Reed-Solomon erasure code 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 [4].*

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 implementations Reed-Solomon codes, 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 applications that involve distributed or networked storage. 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 stored data 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 recovered from the non-failed nodes.

Erasure codes have been employed for fault-tolerance and improved performance in single-site [12, 14, 31], archival [13, 28], wide-area [1, 7, 32] and peer-to-peer storage systems [35, 18, 19, 9]. They have additional uses in content distribution systems [5, 23] and applications with fault-tolerant data structures [20]. 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. **Space overhead** is defined in one of two ways – either by the number of (redundant) coding nodes required to achieve a baseline of fault-tolerance [16, 15], or by the average number of failures tolerated by a given number of coding nodes [21, 30, 27]. Regardless of the evaluation methodology, space optimality may be achieved when the number of coding nodes is equal to the number of failures that may be tolerated. These codes are called *Maximum Distance Separable* codes [22], denoted as "$(n+m,m)$ MDS codes." Clearly, they are desirable. **Encoding performance** is the time/computation complexity of creating the m coding nodes from the n data nodes. A related metric is the **update performance** of a code, which is the number of blocks on coding nodes that must be updated when a data node block is updated. Finally, **decoding performance** is the time/computation complexity of recovering data from the surviving data and coding nodes.

*This material is based upon work supported by the National Science Foundation under grants CNS-0437508, CNS-0549202, IIS-0541527, EIA-0224441 and ACI-0204007.

Fifth IEEE International Symposium on Network Computing and Applications (NCA'06)
0-7695-2640-3/06 $20.00 © 2006 IEEE

In this paper we focus solely on MDS codes, because of their space optimality. Moreover, discussions are limited to the encoding (and update) performance. 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. There are two trivial classes of MDS codes: the *repetition* code (where $n = 1$) and the *single parity* code (where $m = 1$). These are used in RAID-1 (or mirroring) and RAID-5 [6] respectively. Obviously both codes achieve their optimal time complexity.

It is well known in coding theory that besides the above two classes of codes, there are *no* other *binary* MDS codes [22, Ch.11]. In other words, any ($n + m$, m) MDS code where $m \geq 2$ must define its codeword symbols over a finite field other than (binary) $GF(2)$. It is also well known that a dual code of an ($n + m$,m) MDS code is also MDS [22, Ch.11]. In other words, an ($n + m$,n) MDS code can be easily constructed from an ($n + m$,m) MDS code. Thus one only needs to focus on ($n + m$, m) MDS codes with $m \leq n$.

Most research effort on time-efficient MDS codes has been on *array codes*, a class of 2-dimensional MDS codes whose encoding and decoding operations can be performed using binary XORs, which can be computed very efficiently in hardware and/or software. See [17] for a complete list of references on array codes. Most of these array codes are for $m = 2$, including EVENODD [2], X-Code [34], B-Code [33], and RDP [8]. The latter codes achieve optimal performance along all dimensions. For $m = 3$, there are a generalized EVENODD code [3], its recent variation [17], and an array code derived from the $(n + 3, 3)$ Reed-Solomon code [10]. None of these achieves optimal time complexity, though all are quite close. For $m \geq 4$, some MDS array codes do exist [11], but their time complexity is not optimal either.

Apart from these codes, the only known MDS codes are the Reed-Solomon codes, which have existed for decades [22] and are widely used in communication and storage systems. 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 [20, 28, 7].

In 1995, Blomer *et al* presented *Cauchy Reed-Solomon (CRS)* coding [4] which improved the performance of Reed-Solomon codes. 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 the computation performance. Low-Density Parity-Check (LDPC) codes have received much recent attention as high-performance alternatives to MDS codes. See [27, 25] for tutorial material on LDPC codes and for citations. While LDPC codes are asymptotically MDS, they have significant space overhead penalties for the values of n and m that many storage applications require. When the ratio of networking performance to CPU speed is high enough, LDPC codes do outperform their MDS alternatives. However, when that ratio is lower, MDS codes perform better [27, 7].

A second class of non-MDS codes are the recently-developed HoVer and WEAVER codes [15, 16]. Both are array codes for small m that have time-optimal characteristics, but are only MDS in two cases: ($n = 2, m = 2$ and $n = 3, m = 3$).

The Contribution of This Paper

As described above, currently known time-efficient MDS codes are limited to small m (in fact, mostly $m = 2$ and $m = 3$). As storage systems increase in size, (i.e. as the number of storage nodes grows) so does the probability of concurrent multiple-node failures. This calls for MDS codes with large m, which at present means Reed-Solomon codes. 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.

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 and specifies an arbitrary construction. The authors quantify the matrix's impact on performance as a factor of $O(\log_2(m + n))$ [4]. 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 81%. Moreover, 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 [4], classical "Vandermonde" Reed-Solomon coding [24], and the parity-based MDS codes [2, 34, 17, 15, 10, 11]. As such, this paper provides a useful reference for the performance of various

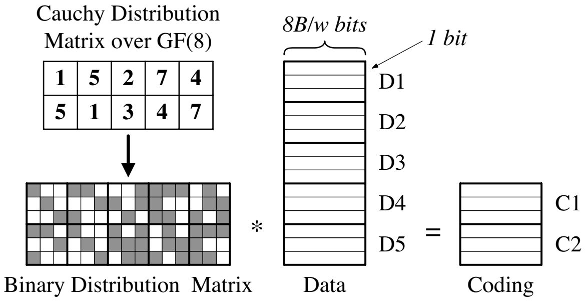


Figure 1: Cauchy Reed-Solomon coding example with $n = 5, m = 2$, and $w = 3$. The Cauchy distribution matrix is selected with $X = \{1, 2\}$ and $Y = \{0, 3, 4, 5, 6\}$.

MDS codes.

2 Cauchy Reed-Solomon Coding

The mechanics of Reed-Solomon Coding, both regular and Cauchy variants, are well understood and presented in tutorial form in a variety of sources [4, 24, 26, 29]. The presentation in [25] is most germaine to this paper and is recommended for readers who desire to implement this scheme.

Figure 1 summarizes the mechanics of CRS encoding, using an example with $n = 5$ and $m = 2$. A *word size* $w \geq \log_2(n + m)$ is selected ($w = 3$ in Figure 1), and the data/coding devices are each partitioned into w *packets* of size B/w bytes. The wn data packets are arranged as $8B/w$ column vectors of one bit each.

A *Cauchy distribution matrix* is defined over the Galois Field $GF(2^w)$ in the following way. 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, Figure 1 displays the Cauchy distribution matrix over over $GF(2^3)$, where $X = \{1, 2\}$ and $Y = \{0, 3, 4, 5, 6\}$.

The $m \times n$ Cauchy distribution matrix over $GF(2^w)$ is next converted into a $wm \times wm$ *binary distribution matrix* using a projection defined by $GF(2^w)$'s primitive polynomial [4]. With this matrix, addition is XOR and multiplication is binary AND, and thus instead of encoding using standard matrix multiplication, one may create a coding packet as the XOR of all data packets whose corresponding columns of the binary distribution matrix have ones in the coding packet's row. Note that this is a big improvement over standard Reed-Solomon coding, because the Galois Field arithmetic over portions of 32 or 64 bit words is replaced by XOR's, each of which may be executed by one machine instruction.

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 1, 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 $GF(2^8)$, this is an improvement indeed, and is why, for example, OceanStore [28] uses Cauchy Reed-Solomon coding instead of standard Reed-Solomon coding.

All Cauchy Matrices Are Not Equal

In [4], 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 a matrix which has 54 ones, as opposed to the 47 ones when $X = \{1, 2\}$ and $Y = \{0, 3, 4, 5, 6\}$. The impact on performance is significant: $54/6 - 1 = 8$ XORs per word, or a 17% decrease in performance over the matrix in Figure 1. This observation fuels the exploration in the remainder of the paper.

3 Enumerating Cauchy Matrices

The simplest way to discover optimal Cauchy Matrices is to enumerate them. Given n, m, and w, 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 2. Instead of plotting the number of XORs, we plot the factor over optimal coding, where optimal is defined as $n - 1$ XORs per coding word [34, 16]. Thus, for example, our exploration shows that the Cauchy matrix of Figure 1 indeed has the minimal number of ones. Since 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 2 at $n = 5$, $m = 2$.

There are three features of Figure 2 worth mentioning. First, there is a significant difference in the performance of the minimum and maximum Cauchy matrices

[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.

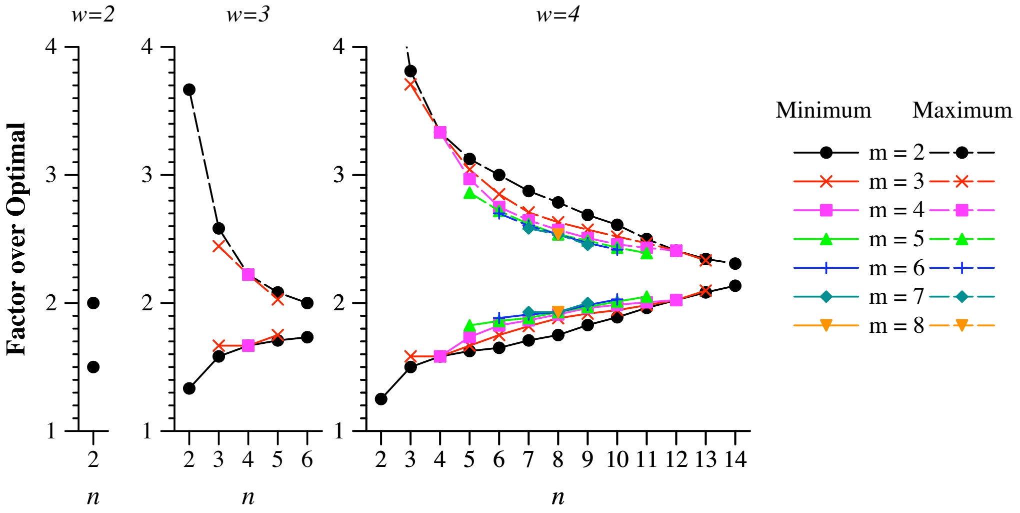


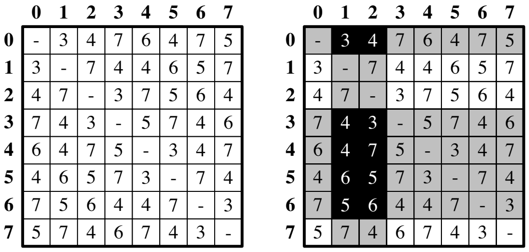
Figure 2: Minimum and maximum Cauchy matrices for $w \leq 4$ (i.e., $n + m \leq 16$).

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 explore this phenomenon further in [25].

4 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 $n \neq m$,



	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	-

(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)

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

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 3(a).

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:

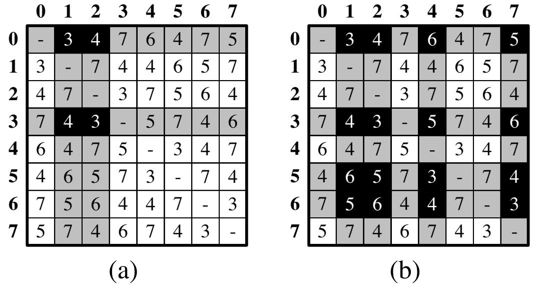
$$W(w, X, Y) = \sum_{i=1}^{n} \sum_{j=1}^{m} ONES(w)_{Y_i, X_j}.$$

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 3(b), we show the Cauchy matrix of Figure 1, 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 4.



	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	-

(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	-

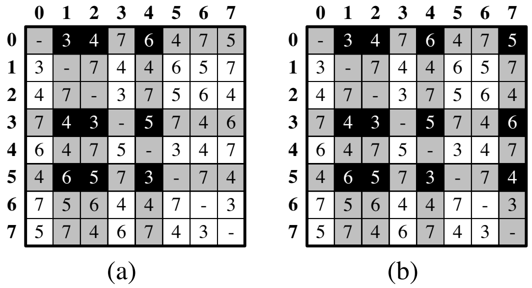
(b)

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

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 4(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 4(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 5(a), and is constructed by deleting row 6 and column 7 from $GC(4, 4, 3)$ (Figure 4(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	-

(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)

Figure 5: (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 5(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 3, and allows us to construct good matrices for larger values of n and m.

5 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

Fifth IEEE International Symposium on Network Computing and Applications (NCA'06)
0-7695-2640-3/06 $20.00 © 2006 IEEE

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 [15] — one for $m = 2, n = 2$, and one for $m = 3, n = 3$. Both perform optimally.
- **X-Code**: The optimal X-Code [34] is defined for $m = 2$ and $n + 2$ prime .
- **EVENODD**: This is defined for $m = 2$ and all n [2]. Its performance is slightly worse than optimal.
- **STAR**: This is an extrapolation of EVENODD coding for $m = 3$ [17].
- **FENG**: These are the recently-defined MDS array codes by Feng *et al* [10, 11].
- **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 [4], 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 [26, 24, 29].

The metric for comparison is the factor over optimal coding, as in Figure 2. 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 [24]), or than simulating polynomial arithmetic over $GF(2)$ using XOR and bit-shifting [22]. This was done on a Dell Precision Workstation with a 3.40 GHz Intel Pentium 4 processor. The measurements are: $B_\oplus$ = 2992 MB/s and B_* = 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:

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

The results for small m are in Figure 6. 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, X-Code, etc.) drastically outperform the Reed-Solomon codes. Thus, in applications which need resilience 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 6. 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 7, we show the performance of the Reed-Solomon codes for three

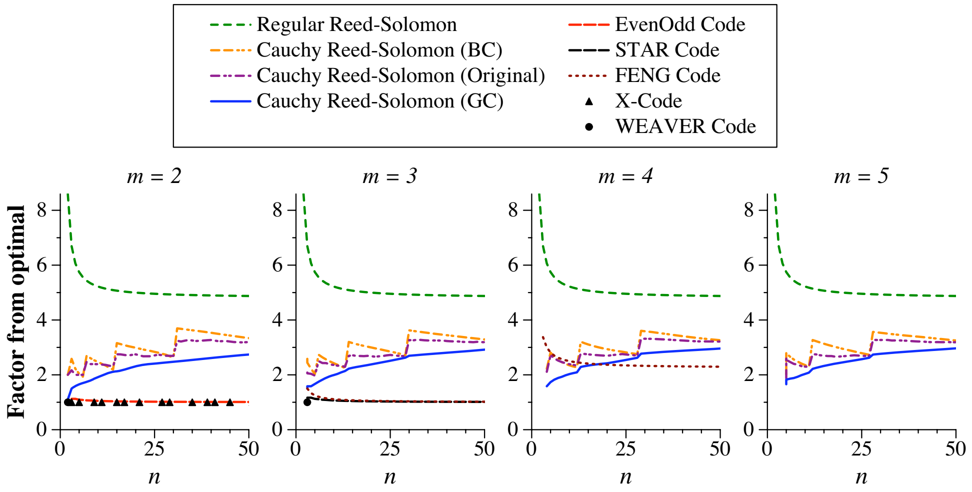


Figure 6: Performance comparison of MDS codes for $2 \le m \le 5$.

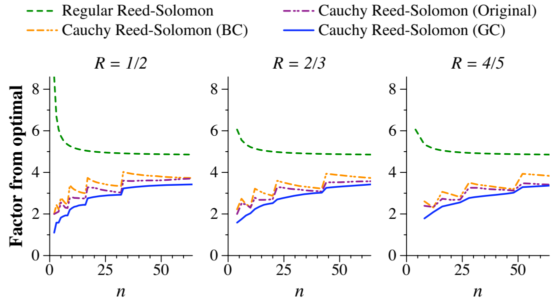


Figure 7: Performance of Reed-Solomon coding for higher values of n and m

	$GC(3, 29, w)$		$BC(3, 29, w)$	
	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.

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 [21, 30, 27]. 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.

The only real difference between Figures 6 and 7 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 cases.

6 Additional Resources/Results

For brevity, we omit some further explorations on how codes with larger w can perform better than those with smaller w. The reader is referred to [25], which also includes the the optimal and GC matrices generated for this paper, so that those who need to implement this technique may do so easily.

7 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 maxi-

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 of GC matrices, compared to the original Cauchy constructions.

mum 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 codes, and that more research should be performed on broadening these algorithms for larger numbers of failures. The recent work by Hafner [15, 16] and Feng [10, 11] 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

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

[2] M. Blaum, J. Brady, J. Bruck, and J. Menon. EVENODD: An efficient scheme for tolerating double disk failures in RAID architectures. *IEEE Trans. Comp.*, 44(2):192–202, 1995.

[3] M. Blaum, J. Bruck, and A. Vardy. MDS array codes with independent parity symbols. *IEEE Trans. Inf. Thy*, 42(2):529–542, 1996.

[4] 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.

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

[6] 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.

[7] R. L. Collins and J. S. Plank. Assessing the performance of erasure codes in the wide-area. *DSN-05: Int. Conf. on Dependable Sys. and Networks*, Yokohama, 2005.

[8] P. Corbett *et al.* Row diagonal parity for double disk failure correction. *4th Usenix Conf. on File and Storage Tech.*, San Francisco, 2004.

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

[10] G. Feng, R. Deng, F. Bao, and J. Shen. New efficient MDS array codes for RAID part I: Reed-Solomon-like codes for tolerating three disk failures. *IEEE Trans. Comp.*, 54(9):1071–1080, 2005.

[11] G. Feng, R. Deng, F. Bao, and J. Shen. New efficient MDS array codes for RAID part II: Rabin-like codes for tolerating multiple ($\geq$ 4) disk failures. *IEEE Trans. Comp.*, 54(12):1473–1483, 2005.

[12] S. Frolund, A. Merchant, Y. Saito, S. Spence, and A. Veitch. A decentralized algorithm for erasure-coded virtual disks. *DSN-04: Int. Conf. on Dependable Sys. and Networks*, Florence, 2004.

[13] A. Goldberg and P. N. Yianilos. Towards an archival intermemory. *ADL-98: IEEE Adv. in Dig. Libr.*, Santa Barbara, 1998, pp. 147–156.

[14] G. R. Goodson, J. J. Wylie, G. R. Ganger, and M. K. Reiter. Efficient byzantine-tolerant erasure-coded storage. *DSN-04: Int. Conf. on Dependable Sys. and Networks*, Florence, 2004.

[15] J. L. Hafner. WEAVER Codes: Highly fault tolerant erasure codes for storage systems. *FAST-2005: 4th Usenix Conf. on File and Storage Tech.*, San Francisco, 2005, pp. 211–224.

[16] J. L. Hafner. HoVer erasure codes for disk arrays. *DSN-06: Int. Conf. on Dependable Sys. and Networks*, Philadelphia, 2006.

[17] C. Huang and L. Xu. STAR: An efficient coding scheme for correcting triple storage node failures. *FAST-2005: 4th Usenix Conf. on File and Storage Tech.*, San Francisco, 2005, pp. 197–210.

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

[19] W. K. Lin, D. M. Chiu, and Y. B. Lee. Erasure code replication revisited. *PTP04: 4th Int. Conf. on Peer-to-Peer Computing*. 2004.

[20] W. Litwin and T. Schwarz. Lh*rs: a high-availability scalable distributed data structure using Reed Solomon codes. *2000 ACM SIGMOD Int. Conf. on Management of Data*, 2000, pp. 237–248.

[21] M. Luby, M. Mitzenmacher, A. Shokrollahi, D. Spielman, and V. Stemann. Practical loss-resilient codes. *29th Annual ACM Symp. on Theory of Computing*, El Paso, TX, 1997, pp. 150–159.

[22] 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.

[23] M. Mitzenmacher. Digital fountains: A survey and look forward,. *2004 IEEE Inf. Theory Workshop*, San Antonio, 2004.

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

[25] J. S. Plank. Optimizing Cauchy Reed-Solomon codes for fault-tolerant storage applications. Technical Report CS-05-569, Univ. Tennessee, December 2005.

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

[27] J. S. Plank and M. G. Thomason. A practical analysis of low-density parity-check erasure codes for wide-area storage applications. *DSN-04: Int. Conf. on Dependable Sys. and Networks*, Florence, 2004, pp. 115-124.

[28] 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.

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

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

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

[32] H. Xia and A. A. Chien. RobuSTore: Robust performance for distributed storage systems. Technical Report CS2005-0838, Univ. Calif. San Diego, October 2005.

[33] L. Xu, V. Bohossian, J. Bruck, and D. Wagner. Low density MDS codes and factors of complete graphs. *IEEE Trans. Inf. Thy*, 45(6):1817–1826, 1999.

[34] L. Xu and J. Bruck. X-Code: MDS array codes with optimal encoding. *IEEE Trans. Inf. Thy*, 45(1):272–276, 1999.

[35] Z. Zhang and Q. Lian. Reperasure: Replication protocol using erasure-code in peer-to-peer storage network. *21st IEEE Symp. Reliable Distributed Systems*, 2002, pp. 330–339.