Array Codes for Random and Clustered Disk Failures

Yuval Cassuto and Jehoshua Bruck
California Institute of Technology
Electrical Engineering Department
MC 136-93
Pasadena, CA 91125, U.S.A.
E-mail: {ycassuto,bruck}@paradise.caltech.edu

Abstract

RC (Random/Clustered) code is a new efficient array-code family for recovering from 4 disk failures. Most such failures can be corrected and, in particular, all 4 failures that fall into at most two clusters. These include a single cluster of 4 failures, two clusters of 2 failures each and two clusters, one with 3 failures and the other with 1. In addition, all but a vanishing number of failures that fall into three clusters are corrected. Motivations to address clustered failure correction are provided. Our new RC codes are significantly more efficient in all practical implementation factors than the best known 4 erasure correcting MDS code. These include: small-write update-complexity, full-disk update-complexity, decoding time and number of supported disks in the array.

I. INTRODUCTION

Protecting disk-arrays and other dynamic storage systems against device failures has long become an essential part of their design. Implemented solutions to data survivability in the presence of failed hardware have progressed considerably in the last twenty years. In the dawn of failure protected storage systems, relatively simple schemes were implemented. In RAID [1] arrays, a redundant disk is used to store parity bits of the information disks which helps recovering from any single disk failure. Simple data replication and data striping are also commonly used to avoid data loss. Meanwhile, storage requirements are growing rapidly and at the same time, device reliability was reduced to control the implementation cost. Consequently, recovering from only a single failure has become inadequate while data replication is turning infeasible. Schemes that are based on the Reed-Solomon codes [2], can recover from more failures, and with a good resiliency-redundancy trade-off, but they require complex decoding algorithms (in either space or time) and also many parity writes are needed for small data updates. These shortcomings left such schemes out of reach of many storage applications. In the literature of array codes a column serves as an abstraction to a disk or other physical storage unit. A distinctive example in the class of array codes is the EVENODD code [4] and its relatives (e.g [5]), that can recover from any two failures with optimal redundancy (MDS), simple decoding, and fairly low update-complexity. EVENODD codes for more than two failures exist [6], but their decoding becomes more complex for growing numbers of failures, and their update complexity grows as fast as $2^r - 1$, for $r$ correctable failures. A high update complexity limits the system performance as it imposes excess disk I/O operations, even if no failures occur. High update complexity also implies high wear of parity disks whose direct consequence is the shortening of disk lifetimes.

The primary incentive to move to higher order failure resilience in disk arrays is to combat “catastrophic” events, in which multiple disks fail simultaneously. For such events, the assumption that device failures occur independently of each other is no longer true, and many failure mechanisms cause multiple disk failures that are highly correlated. Since adjacent disks in the array share cabling, cooling, power and mechanical stress, failure combinations that are clustered are more likely than completely isolated multiple failures. Consequently, array codes that have excellent clustered-failure correctability, good random-failure correctability, and low update complexity are desirable for high order failure protection of disk arrays.

1This work was supported in part by the Caltech Lee Center for Advanced Networking and by NSF grant ANI-0322475
This paper pursues the first attempt to improve the performance of array-codes by prioritizing the correctable failures based on their relative locations in the array. A related work in [7], suggests a trade-off between storage space and access efficiency. The main contribution of this paper is the construction of an array-code family, called RC codes (for Random/Clustered), that corrects essentially all clustered failures and 7 out of 8 non-clustered failure combinations. The RC codes are better than EVENODD in all implementation complexity parameters. They improve the encoding and decoding complexities by 25% and the small-write update complexity by 28.57%. They also support twice as many disks compared to EVENODD codes.

II. DEFINITIONS AND NOTATIONS

A. ARRAY-CODES

Throughout the paper we use common coding theory terminology that provides a good abstraction for failure-resilient disk arrays. A length $n$ array code consists of $n$ columns. A column is a model for, depending on the exact application, either a whole disk or a different physical data unit (such as a sector) in the disk array. In the codes discussed here, there are $k$ columns that store uncoded information bits and $r$ columns that store redundant parity bits (thus $n = k + r$). This array structure has the advantage that information can be read off a disk directly without decoding, unless it suffered a failure, in which case a decoding process is invoked. An array-code that admits this structure is called strongly systematic. A column erasure occurs when, for some physical reason, the contents of a particular column cannot be used by the decoder. An erasure is a model for a device failure whereby all the data on a disk (or other physical unit) is known to have become unusable. We say that an array with given column erasures is correctable by the array code if there exists a decoding algorithm that, independent on the specific array contents, can reconstruct the original array from unerased columns only. An array-code is called MDS (Maximum Distance Separable) if it has $r$ redundant columns and it can correct all possible combinations of $r$ column erasures. MDS codes obviously have the strongest conceivable erasure correction capability for a given redundancy, since the $k$ information columns can be recovered from any $k$ columns. Beyond space efficiency of the code, one should also consider its I/O efficiency. I/O efficiency of a disk array is determined by the small-write and full-column update complexities of the array code used. The small-write update complexity is defined as the number of parity-bit updates required for a single information bit update, averaged over all information bits. The full-column update complexity is the number of parity columns that have to be modified per a single full-column update. Another crucial performance marker of an array code is its erasure-decoding complexity, defined as the number of bit operations (additions, shifts) required to recover the erased columns from the surviving ones. Unless noted otherwise, $p$ will refer to a general prime number $p$.

B. RANDOM/CLUSTERED ERASURE CORRECTION

To describe column erasure combinations whose only restriction is the number of erased columns, it is customary to use the somewhat misleading term random erasures. An array is said to recover from $\rho$ random erasures if it can correct all combinations of $\rho$ erased columns. The random erasure model is most natural when storage nodes are known to, or more commonly, assumed to behave uniformly and independent of each other. Indeed, almost all array-code constructions discussed in the literature aim at correcting random erasures. Refinement of the erasure model is possible by adding restrictions on the relative locations of the erased columns. This paper considers clustered erasures, where the $\rho$ erasures fall into a limited ($\rho$) number of clusters. We now turn to some definitions related to the clustered erasure model. In words, a cluster is a contiguous block of columns. In an array-code with columns numbered $\{0, 1, 2, \ldots, n - 1\}$, a cluster is formally defined as a set of $\sigma$ columns such that the difference between the highest numbered column and the lowest numbered one is exactly $\sigma - 1$. For example, $\{2, 3, 4, 5\}$ is a cluster with $\sigma = 4$. Random erasures are a special case of clustered erasures where the $\rho$ erasures fall (trivially) into at most $\rho$ clusters. The other extreme is the column burst model where all erased columns need to fall into a single cluster. These two well-studied extreme cases open our presentation and later the RC codes are shown to be very effective for all intermediate cases of clustered erasures. An illustration of the column-clustering definitions is given in Figure 1.

III. PRELIMINARIES AND RELEVANT KNOWN RESULTS

The purpose of this section is to discuss relevant known results in sufficient detail to prepare for the presentation of the new RC codes in the next section. Some algebraic tools used later for RC codes have been used before for other codes, so this section also serves to elucidate those tools prior to their use by the actual code construction.
A. Codes for erasures in a single cluster

Assume that our design goal is a disk array that will sustain any erasure of $\rho$ columns in a single cluster, without requiring any random erasure correction capability. One option is to take a code that corrects any $\rho$ random column erasures that, in particular, corrects any clustered $\rho$ erasures. However, as can be expected, this may not be the best approach since correcting all random erasures is a far stronger requirement that excludes much simpler and more efficient constructs. As this section shows, a simple and well known technique called interleaving can achieve that task optimally both with respect to the required redundancy and in terms of the code update complexity.

Let $C_P$ be an array-code with $n'$ columns, out of which $k' = n' - 1$ are information columns. The remaining column holds the bit-wise parity of the $k'$ information columns. Define the code $C_P^{\rho}$ as the length $n = \rho n'$ code that is obtained by the interleaving of $\rho$ codewords of $C_P$. In other words, if $C^{(1)}, C^{(2)}, \ldots, C^{(\rho)}$ are $\rho$ codewords of $C_P$, then the corresponding code word of $C_P^{\rho}$ will be

$$C^{(1)}_1 C^{(2)}_1 \ldots C^{(\rho)}_1 C^{(1)}_2 C^{(2)}_2 \ldots C^{(\rho)}_2 C^{(1)}_3 \ldots$$

**Proposition 1:** The code $C_P^{\rho}$ corrects any $\rho$ erasures in a single cluster.

**Proof:** Any erasure that is confined to at most $\rho$ consecutive columns erases at most one column of each constituent $C_P$ code. These single erasures are correctable by the individual $C_P$ codes. It is clear that the code $C_P^{\rho}$ has optimal redundancy since it satisfies $\rho = r$ and $\rho$ is a well known and obvious lower bound on the redundancy $r$. For any $\rho$, the code $C_P^{\rho}$ has update complexity (both small-write and full-column) of 1, which is optimal since a lower update complexity would imply at least one code bit that is independent of all other bits, and erasure of that bit would not be correctable.

B. Codes for random erasures

As mentioned in sub-section III-A, array-codes that correct any $\rho$ random erasures also correct any $\rho$ clustered erasures. In this section we seek to survey a family of random erasure correcting codes: the EVENODD [4],[8] codes. These codes enjoy several properties that make them most appealing for implementation in disk arrays. The purpose of
presenting the codes here is twofold. First, in the absence of prior codes that combine clustered and random correction capabilities, EVENODD will be used as the current state-of-the-art for comparison with our construction. Second, the analysis of the new RC codes herein is made simpler by building on properties previously shown for EVENODD.

1) EVENODD for correcting 2 Random disk erasures: An EVENODD code [4] takes \( p \) data columns, each of size \( p - 1 \) and adds to them 2 parity columns of the same size. The encoded array is therefore of size \( (p - 1) \times (p + 2) \). EVENODD can correct any 2 column erasures so it is clearly optimal in terms of added redundancy (MDS). Other properties of EVENODD are that it is strongly systematic, it has low small-write update-complexity that is approximately one above optimal, and optimal full-column update complexity. In addition, it can be encoded and decoded using simple XOR and shift operations. The simplest way to define the EVENODD code is through its encoding rules. Given a \( (p - 1) \times p \) information bit array, parity column \( P \) and parity column \( Q \) are filled using the rules shown in Figure 2, for the case \( p = 5 \). An imaginary row, shown unshaded, is added to the array for the sake of presenting the structure of the encoding function. Parity column \( P \) is simply the bit-wise even parity of the information columns (4 parity groups are marked using the 4 different icons in Figure 2a). Parity column \( Q \) is the slope-1 diagonal parities of the information bits (whose groups are marked by icons in Figure 2b). Whether the parity of these diagonal groups is set to be even or odd is decided by the parity of the information bits that lie on the diagonal that was left blank in Figure 2b.

![Fig. 2. Encoding of EVENODD code (a) horizontal parity \( P \) (b) diagonal parity \( Q \)](image)

2) Algebraic description of EVENODD: In the previous subsection, EVENODD codes were defined using their simple encoding functions. We now include the algebraic description of EVENODD codes from [4] that will be most useful for later proofs in this paper. Columns of the \( (p - 1) \times (p + 2) \) array are viewed as polynomials of degree \( \leq p - 2 \) over \( \mathbb{F}_2 \) modulo the polynomial \( M_p(x) \), where \( M_p(x) = (x^p + 1)/(x + 1) = x^{p-1} + x^{p-2} + \cdots + x + 1 \) (recall that in \( \mathbb{F}_2 \) summation and subtraction are the same and both done using the boolean XOR function). According to that view, the polynomial for a column vector \( c = [c_0, \ldots, c_{p-2}]^T \) is denoted \( c(\alpha) = c_{p-2} \alpha^{p-2} + \cdots + c_1 \alpha + c_0 \). Bit-wise addition modulo 2 of two columns is equivalent to summing the corresponding polynomials in the ring of polynomials modulo \( M_p(x) \), denoted \( R_p \). Multiplying \( c(\alpha) \) by \( \alpha \) results in a downward shift of \( c \) if \( c_{p-2} \) is zero. In the case \( c_{p-2} = 1 \), reduction modulo \( M_p(x) \) is needed and \( \alpha c(\alpha) \) is obtained by first downward shifting \( [c_0, \ldots, c_{p-3}, 0]^T \) and then inverting all the bits of the shifted vector. Then, it is not hard to see that the encoding rules depicted in Figure 2 induce the following parity check matrix over \( R_p \).

\[
H = \begin{bmatrix}
1 & 1 & \cdots & 1 & 1 & 0 \\
1 & \alpha & \cdots & \alpha^{p-1} & 0 & 1
\end{bmatrix}
\]

The top row of \( H \) represents the horizontal parity constraints of \( P \), where all columns have the same alignment. The bottom row represents the diagonal parity constraints of \( Q \), where a column is shifted one location upwards relative to its left neighbor. The structure of the ring \( R_p \) provides for the even/odd adjustment of the \( Q \) parities, as a function of the parity of the bits in the blank cells. The importance of this algebraic definition of the code is due to the fact that proving correctability of an erasure combination reduces to showing that the determinant of a sub-matrix of \( H \) has an inverse in the ring \( R_p \). Subsequent sections will discuss that in more detail.

3) EVENODD for correcting 3 and more Random disk erasures: In [8], EVENODD was generalized to codes that correct \( r > 2 \) random erasures. The main idea in the generalization is to add more parity columns that constrain the
code bits across diagonals with different slopes (recall that EVENODD uses slopes 0 and 1). Discussing EVENODD generalization in depth is beyond the scope of this paper. We only mention the following facts that are relevant to our presentation.

- The asymptotic small-write update-complexity of the general EVENODD code family is \(2r - 1 - o(1)\). \(o(1)\) refers to terms that tend to zero as the code length goes to infinity. Their full-column update-complexity is \(r\).
- For \(r > 3\), generalized \(r\) random erasure correcting EVENODD codes are only guaranteed to exist for \(p\) that belong to a subset of the primes: those that satisfy that 2 is a primitive element in the Galois field \(GF(p)\).
- The best known way to decode general EVENODD codes is using the algorithm of [9] over \(R_p\), for which the decoding complexity is dominated by the term \(rkp\).

C. The Ring \(R_p\)

Properties of the ring \(R_p\) are used in subsequent sections to prove the correctability of erasure patterns by RC codes. This same ring was studied for code analysis in [9] and later in [8]. Accordingly, the purpose of this section is to summarize useful properties of \(R_p\), following the necessary mathematical definitions. Recall that the elements of the ring \(R_p\) are polynomials with binary coefficients and degree \(\leq p - 2\). Addition is defined as the usual polynomial addition over \(\mathbb{F}_2\) and multiplication is taken modulo \(M_p(x) = \sum_{i=0}^{p-1} x^i\). Element \(f(x)\) is invertible in the ring if and only if \(\gcd(f(x), M_p(x)) = 1\), where \(\gcd(\cdot, \cdot)\) stands for the polynomial greatest common divisor. If \(f(x)\) is non-invertible \((\gcd(f(x), M_p(x)) \neq 1)\), then there exists an element \(g(\alpha) \in R_p\) such that \(f(\alpha)g(\alpha) = 0\). In that case we say that \(f(\alpha), g(\alpha)\) are both divisors of 0. For convenience of notation the element \(\sum_{i=0}^{p-2} \alpha^i\) is denoted \(\alpha^{p-1}\). Note also that \(\alpha^p = 1\). The following is a useful lemma from [9].

**Lemma 1:** For any prime \(p\), all elements of the forms \(\alpha^i\) and \(\alpha^i + 1\) are invertible, for any \(1 \leq i < p\).

Next a fundamental closure property of rings is worth mentioning.

**Lemma 2:** A product of invertible elements is invertible.

Lemmas 1 and 2 together provide a family of ring elements that are invertible for any prime \(p\), namely products of monomials and binomials. When the prime \(p\) has the property that 2 is a primitive element in \(GF(p)\), then \(M_p(x)\) is irreducible, \(R_p\) becomes a field and consequently all non-zero elements of \(R_p\) are invertible. (See [10] for more details on finite fields). Hence, for such primes, the following Lemma (Lemma 2.7 of [8]) provides additional invertible elements.

**Lemma 3:** If a polynomial \(g(x)\) has an odd number of terms and \(t < p\), then \(g(\alpha)\) is invertible in \(R_p\), provided \(M_p(x)\) is irreducible.

IV. DEFINITION OF THE RC CODE

A. Geometric Description

The RC code has \(2p\) information columns of \(p - 1\) bits each and 4 parity columns with the same number of bits. The information columns are numbered in ascending order from left to right using the integers \(\{0, 1, 2, \ldots, 2p - 1\}\). Parity columns are not numbered and we use letter labels for them. The code is defined using its encoding rules shown in Figure 3, for the case \(p = 5\). As before, an imaginary row is added to the array to show the structure of the encoding function. Each icon represents, similarly to the definition of EVENODD in III-B.1, a group of bits that are constrained by the code to have even/odd parities. Parity column \(P\), located in the left most column of the left parity section, is simply the bit-wise even parity of the \(2p\) information columns. Parity column \(R'_1\), located second from left, is the slope \(-1\) diagonal parity of the odd numbered information columns \(\{1, 3, \ldots, 2p - 1\}\). The bit groups of \(R'_1\) are set to have even parity if the bits marked EO have even parity, and odd parity otherwise. Parity column \(R_0\), located in the left most column of the right parity section, is the slope 2 diagonal parity of the even numbered information columns \(\{0, 2, \ldots, 2p - 2\}\). Parity column \(Q\), located in the right most column of the right parity section, is the XOR of the slope 1 diagonal parities of both the even numbered columns and the odd numbered columns. The parity groups of \(Q\) and \(R_0\), similarly to those of \(R'_1\), are set to be even/odd, based on the parity of the corresponding EO groups. Note that parity columns \(P\) and \(Q\) can be decomposed into \(P = P0 \oplus P1\) and \(Q = Q0 \oplus Q1\), where \(P0, Q0\) depend only on even information columns and \(P1, Q1\) solely on odd ones.
For a formal definition of the code we write the encoding functions explicitly. Denote by $c_{i,j}$ the bit in location $i$ in information column $j$. For an integer $l$, define $\langle l \rangle$ to be $l \pmod{p}$. Now we write the explicit expression of the parity bits.

$$P_i = \bigoplus_{j=0}^{2p-1} c_{i,j}$$
$$R'_{1i} = S_1 \oplus \bigoplus_{j=0}^{p-1} c_{(i+j),2j+1},$$

where

$$S_1 = \bigoplus_{j=0}^{p-1} c_{(p-1+j),2j+1}$$

$$R_{0i} = S_0 \oplus \bigoplus_{j=0}^{p-1} c_{(i-2j),2j},$$

where

$$S_0 = \bigoplus_{j=0}^{p-1} c_{(p-1-2j),2j}$$

$$Q_i = S_Q \oplus \bigoplus_{j=0}^{p-1} c_{(i-j),2j} \oplus \bigoplus_{j=0}^{p-1} c_{(i-j),2j+1},$$

where

$$S_Q = \bigoplus_{j=0}^{p-1} c_{(p-1-j),2j} \oplus \bigoplus_{j=0}^{p-1} c_{(p-1-j),2j+1}$$

**B. Algebraic Description**

Using the ring $\mathcal{R}_p$, the parity check matrix $H$ of the RC code for $p = 5$ is given by

$$\begin{bmatrix}
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\
0 & 1 & 0 & 1 & 0 & \alpha^4 & 0 & \alpha^3 & 0 & \alpha^2 & 0 & \alpha & 0 & 0 \\
0 & 0 & 1 & 0 & \alpha^2 & 0 & \alpha^4 & 0 & \alpha & 0 & \alpha^3 & 0 & 1 & 0 \\
0 & 0 & 1 & 1 & \alpha & \alpha & \alpha^2 & \alpha^2 & \alpha^3 & \alpha^3 & \alpha^4 & \alpha^4 & 0 & 1
\end{bmatrix}$$

The correspondence between the encoding function in Figure 3 and the parity check matrix above is straightforward. The columns of the parity check matrix correspond to columns of the code array. The two left most columns are for parity columns $P$ and $R'_{1i}$ and the two right most columns are for $R_0$ and $Q$. Columns in between correspond to information columns in the array. In the parity check matrix, row 1 represents the constraints enforced by parity column $P$, rows 2, 3, 4 similarly represent the parity constraints of $R'_{1i}, R_0, Q$, respectively. In any row $j$, the difference of exponents of $\alpha$ in two different columns is exactly the relative vertical shift of the two columns in the icon layout of the appropriate parity in Figure 3. For example, in the top row, all information columns have the same element, $1(=\alpha^0)$, to account for the identical vertical alignment of the icons in the encoding rule of parity $P$. For general $p$ the parity check matrix $H$ has the following form.

$$\begin{bmatrix}
1 & 0 & 1 \cdots & 1 & 1 & 1 & 1 & \cdots & 1 & 1 & \cdots & 1 & 0 & 0 \\
0 & 1 & 0 & \cdots & 0 & \alpha^{-i} & 0 & \alpha^{-(i+1)} & \cdots & \alpha & 0 & 0 \\
0 & 0 & 1 & \cdots & \alpha^{2i} & 0 & \alpha^{2(i+1)} & 0 & \cdots & 0 & \alpha & 0 & 0 \\
0 & 0 & 1 & \cdots & \alpha^i & \alpha^i & \alpha^{i+1} & \alpha^{i+1} & \cdots & \alpha^{i+p-1} & 0 & 1
\end{bmatrix}$$

After presenting the RC code family, we proceed in the next section to prove its random and clustered erasure correction capabilities.
V. ERASURE CORRECTABILITY OF RC CODES

A. Random erasure correctability

In this section we examine the effectiveness of using the RC codes for Random column erasures. RC codes are next shown to correct a 7/8 portion of all combinations of 4 erasures. This result is established in Theorem 1 below that follows a series of lemmas. The \(2p + 4\) columns of the RC codes are labeled \(\{P, R_1', 0, 1, \ldots, 2p - 2, 2p - 1, R_0, Q\}\).

Lemma 4: For any prime \(p\), for a combination of 4 erasures, if 3 columns are either even numbered information columns or parity columns in \(\{R_0, P, Q\}\), and 1 column is an odd numbered information column or the parity column \(R_1'\), then it is a correctable 4-erasure. The complement case, 3 odd (or \(R_1'\) or \(P\) or \(Q\)) and 1 even (or \(R_0\)), is correctable as well. (in particular, any 3-erasure is correctable).

Proof: The RC code can correct the erasure patterns under consideration using a two-step procedure. The first step is to recover the erased odd information column. Since only one odd column is erased, parity column \(R_1'\) can be used to easily recover all of its bits. Then, when all odd columns are available, \(P1\) and \(Q1\) are computed, and used to find \(P0\) and \(Q0\) from \(P\) and \(Q\) (if not erased) by

\[
P0 = P1 \oplus P, \quad Q0 = Q1 \oplus Q
\]

After that step, between even information columns, \(R_0, P0\) and \(Q0\), only 3 columns are missing. Since even columns, \(R_0, P0\) and \(Q0\) constitute an EVENODD code with \(r = 3\), the 3 missing columns can be recovered. The complement case of 3 odd and 1 even column erasures is settled by observing that odd columns, \(R_1', P1\) and \(Q1\) constitute an \(r = 3\) MDS code [11].
The next Lemma presents the key property that gives RC codes their favorable random and clustered erasure correctability.

**Lemma 5:** For any prime $p$ such that $p > 5$ and 2 is primitive in $GF(p)$, for a combination of 4 erasures, if 2 columns are even numbered information columns and 2 columns are odd numbered information columns, then it is a correctable 4-erasure.

**Proof:** For the case of 2 even and 2 odd information column erasures we consider two erasure patterns. All possible erasure combinations of that kind are either covered by these patterns directly or are equivalent to them in a way discussed below. The discussion of each erasure pattern will commence with its representing diagram. In these diagrams, a column marked 0 represents an even column and a column marked 1 represents an odd column. Between each pair of columns, an expression for the number of columns that separate them is given.

a) Erasures of the form

\[
\begin{align*}
0 & \leftrightarrow 2j \rightarrow 1 \\
& \leftrightarrow 2(k - 1) \rightarrow 0 \\
& \leftrightarrow 2l \rightarrow 1 \\
\end{align*}
\]

The variables $j, k, l$ satisfy the following conditions: $1 \leq k$, $1 \leq j + k + l \leq p - 1$.

The location of the first even erased column, together with $j, k, l$ determine the locations of the 4 erasures. Any even cyclic shift of the diagram above does not change the correctability of the erasure pattern since this shift gives the same sub-matrix of the parity-check matrix, up to a non-zero multiplicative constant. Hence, we can assume, without loss of generality, that the first even erased column is located in the leftmost information column. To prove the correctability of this erasure pattern we examine the determinant (over $\mathcal{R}_p$) of the square sub-matrix of $H$, that corresponds to the erased columns. This determinant is itself an element of $\mathcal{R}_p$ and if it is provably invertible in $\mathcal{R}_p$, for every $p$ that satisfies the conditions of the lemma, then the erasure combination is correctable by the RC code.

The sub-matrix that corresponds to the erasure pattern above is

\[
M_a^{(j,k,l)} = \begin{bmatrix}
1 & 1 & 1 & 1 \\
0 & \alpha^{-j} & \alpha^{-j-k-l} & \\
1 & 0 & \alpha^{2(j+k)} & 0 \\
1 & \alpha^j & \alpha^{j+k} & \alpha^{j+k+l}
\end{bmatrix}.
\]

Evaluating the determinant of this matrix gives

\[
\begin{align*}
| M_a^{(j,k,l)} | & = \alpha^{2j+3k+l} + \alpha^{2j+k-l} + \alpha^{j+2k} + \alpha^{j+k-l} + \\
& + \alpha^{k+l} + \alpha^k + \alpha^{-l} + \alpha^{-k-l} \\
& = \alpha^{-l}(\alpha^{j+k} + 1)(\alpha^{k+l} + 1) \\
& \cdot (\alpha^{j+k+l} + \alpha^j + \alpha^l + 1 + \alpha^{-k})
\end{align*}
\]

The first three factors in the product are invertible for any legal $j, k, l$ and any prime $p$ by Lemma 1. The last factor is invertible for all $j, k, l$ and any $p > 5$ such that 2 is primitive in $GF(p)$, by Lemma 3. Furthermore, for $p = 5$, checking all possible assignments of $j, k, l$ and verifying that the last factor does not equal $M_5(x)$, we conclude that this pattern is correctable for $p = 5$ as well.

b) Erasures of the form

\[
\begin{align*}
0 & \leftrightarrow 2j - 1 \rightarrow 0 \\
& \leftrightarrow 2k \rightarrow 1 \\
& \leftrightarrow 2l - 1 \rightarrow 1 \\
\end{align*}
\]

The variables $j, k, l$ satisfy the following conditions: $1 \leq j$, $1 \leq l$, $1 \leq j + k + l \leq p - 1$.

Here, like in the previous pattern, we assume, without loss of generality, that the first even erased column is located in the leftmost information column.
The sub-matrix that corresponds to the erasure pattern above is

\[
M_{b}^{(j,k,l)} = \begin{bmatrix}
1 & 1 & 1 & 1 \\
0 & 0 & \alpha^{-j} & \alpha^{-j-k-l} \\
1 & \alpha^{2j} & 0 & 0 \\
1 & \alpha^{j} & \alpha^{j+k} & \alpha^{j+k+l}
\end{bmatrix}.
\]

Evaluating the determinant of this matrix gives

\[
\left| M_{b}^{(j,k,l)} \right| = \alpha^{2j+l} + \alpha^{2j-l} + \alpha^{j-k} + \alpha^{j-k-l} + \\
+ \alpha^{l} + \alpha^{-l} + \alpha^{-k} + \alpha^{-k-l} \\
= (\alpha^{j} + 1)(\alpha^{l} + 1)(\alpha^{j} + \alpha^{j-l} + 1 + \alpha^{-l} + \alpha^{-k-l})
\]

The first two factors in the product are invertible for any legal \( j, k, l \) and any prime \( p \) by Lemma 1. The last factor is invertible for all \( j, k, l \) and any \( p > 5 \) such that \( 2 \) is primitive in \( GF(p) \), by Lemma 3.

The next Lemma treats additional erasure combinations that include parity columns and that are not covered by Lemma 4.

**Lemma 6:** For any prime \( p \) such that \( p > 3 \) and 2 is primitive in \( GF(p) \), the following 4-erasure combinations are correctable:

1) \( R'_{0}, 1 \) odd information column and 2 even information columns
2) \( R_{0}, 1 \) even information column and 2 odd information columns
3) \( R_{0}, R'_{1}, 1 \) even information column and 1 odd information column, except pairs of information columns numbered \( 2i, 2i+1 \), respectively, for \( 1 \leq i \leq p \).

**Proof:** The sub-matrix that corresponds to case 1 is, up to a multiplicative non-zero constant,

\[
M_{1}^{(j,k,l)} = \begin{bmatrix}
1 & 1 & 1 & 0 \\
0 & \alpha^{-j} & 0 & 1 \\
1 & 0 & \alpha^{2(j+k)} & 0 \\
1 & \alpha^{j} & \alpha^{j+k} & 0
\end{bmatrix}.
\]

The variables \( j, k \) satisfy the following conditions: \( 1 \leq k, 1 \leq j + k \leq p - 1 \). Evaluating the determinant of this matrix gives

\[
\left| M_{1}^{(j,k,l)} \right| = \alpha^{j}(\alpha^{j+k} + 1)(\alpha^{j+k} + \alpha^{k} + 1),
\]

an invertible element if \( p > 3 \).

The sub-matrix that corresponds to case 2 is, up to a multiplicative non-zero constant,

\[
M_{2}^{(j,k,l)} = \begin{bmatrix}
1 & 1 & 1 & 0 \\
0 & \alpha^{-j} & \alpha^{-j-k} & 0 \\
1 & 0 & 0 & 1 \\
1 & \alpha^{j} & \alpha^{j+k} & 0
\end{bmatrix}.
\]

The variables \( j, k \) satisfy the following conditions: \( 1 \leq k, 1 \leq j + k \leq p - 1 \). The determinant now equals

\[
\left| M_{2}^{(j,k,l)} \right| = \alpha^{-j-k}(\alpha^{k} + 1)(\alpha^{j+k} + \alpha^{k} + 1),
\]

an invertible element if \( p > 3 \).

The sub-matrix that corresponds to case 3 is, up to a multiplicative non-zero constant,

\[
M_{3}^{(j)} = \begin{bmatrix}
1 & 1 & 0 & 0 \\
0 & \alpha^{-j} & 1 & 0 \\
1 & 0 & 0 & 1 \\
1 & \alpha^{j} & 0 & 0
\end{bmatrix},
\]

an invertible element if \( p > 3 \).
whose determinant equals
\[ |\mathcal{M}_3^{(j)}| = \alpha^j + 1, \]
an invertible element if \( p > 3 \) and if \( j > 0 \). The latter condition is equivalent to requiring that the even and the odd information columns are not numbered \( 2i, 2i + 1 \), respectively, for \( 1 \leq i \leq p \).

Finally, we are ready to prove the main result of this subsection.

**Theorem 1:** For any prime \( p \) such that \( p > 5 \) and \( 2 \) is primitive in \( GF(p) \), the ratio between the number of RC-correctable 4-erasures and the total number of 4-erasures is greater than 0.865. As \( p \) goes to infinity, this ratio tends to \( 7/8 = 0.875 \).

**Proof:** Building on Lemmas 4, 5 and 6, the number of correctable 4-erasures equals
\[
\begin{align*}
(1) & \quad 2 \left( \binom{p+3}{3} (p+1) - (p+1)^2 \right) \\
(2) & \quad \binom{p}{2} \binom{p}{2} \\
(3) & \quad 2p \binom{p}{2} \\
(4) & \quad p(p-1)
\end{align*}
\]

(1), obtained by Lemma 4, is the number of ways to select 3 even information columns (or \( R_0 \) or \( P \) or \( Q \)) and 1 odd information column (or \( R_1 \)), multiplied by 2 to include the complement case, and subtracting the doubly counted combinations with both \( P \) and \( Q \).

(2), obtained by Lemma 5, is the number of ways to select 2 even and 2 odd information columns.

(3), obtained by 1) and 2) of Lemma 6, is the number of ways to select 2 even information columns and 1 odd information column, multiplied by 2 to include the complement case.

(4), obtained by 3) of Lemma 6, is the number of ways to select an even information column \( 2i \) and an odd information column which is not \( 2i + 1 \).

The total number of 4-erasure combinations is
\[ \binom{2p+4}{4} \]
Taking the ratio of the two we obtain
\[
\frac{7p^4 + 34p^3 + 59p^2 + 32p + 12}{8p^4 + 40p^3 + 70p^2 + 50p + 12}
\]
For \( p = 11 \), the ratio attains its minimal value of 0.865. Moreover, it is readily seen that this ratio equals \( 7/8 - o(1) \), while \( o(1) \) are terms that tend to zero as \( p \) goes to infinity.

**B. Clustered erasure correctability**

After showing that a significant portion of 4-erasure combinations are correctable by RC codes, we turn to show that for clustered erasures, RC codes offer essentially the same correction capabilities as MDS codes. RC codes are next shown to correct all 4-erasures in up to two clusters, and almost all 4-erasures in three clusters. Given the Lemmas of the previous section, establishing these results is rather straightforward.

**Theorem 2:** For any prime \( p \) such that \( p > 3 \) and \( 2 \) is primitive in \( GF(p) \), RC codes correct all 4-erasures that fall into at most two clusters.

**Proof:** If a 4-erasure falls into two or less clusters, then it either has 2 even and 2 odd columns or 3 even and 1 odd columns (or the complement). If \( P \) or \( Q \) is erased, then the remaining 3 columns cannot be all odd or all even. Lemmas 4, 5 and 6 address all such combinations, except the two special cases \( \{ R_1', 2, 3, R_0 \} \) and \( \{ R_1', 2p, 2p+1, R_0 \} \). These combinations can be addressed by internal reordering of even information columns in a way that does not affect any of the other proved properties of the code. Also note that here we only required \( p > 3 \), compared to \( p > 5 \) in Lemma 5, since the only 4-erasure that gives a non-invertible determinant of \( \mathcal{M}_b \) for \( p = 5 \) falls into three clusters (see the proof of Lemma 5).
Theorem 3: For any prime $p$ such that $p > 5$ and $2$ is primitive in $GF(p)$, the ratio between the number of RC-correctable 4-erasures that fall into three clusters and the total number of 4-erasures with three clusters is greater than $0.9696$. As $p$ goes to infinity, this ratio tends to $1$.

Proof: A 4-erasure with three clusters has two clusters of size 1 and one cluster of size 2. If a 4-erasure falls into three clusters, then it either has 2 even and 2 odd columns or 3 even and 1 odd columns (or the complement). Lemmas 4, 5 and 6 address all such combinations, except the following special cases. \{$R'_1, 2i, 2i + 1, R_0$\} cannot be corrected as it is not covered by case 3) of Lemma 6. Also, \{$P, R'_1, 2i + 1, 2j + 1$\} and \{$2i, 2j, R_0, Q$\} cannot be corrected as they are excluded from Lemma 4.

Hence the number of non-correctable 4-erasures with three clusters is

$$p + 2\binom{p}{2}$$

The total number of 4-erasures with three clusters is

$$3\binom{2p-1}{3}$$

(in general this equals $3\binom{n-3}{3}$ for length $n$ arrays since by taking any choice of 3 points on a length $n - 3$ line, we can uniquely obtain an erasure combination with three clusters, following the procedure below. We first choose 3 points from the $n - 3$ line to be the cluster locations. Then the point that represents the cluster with size 2 is selected from these 3 points (for that we have the factor 3). Given these choices, the 3 clusters are obtained by augmenting the size 2 cluster with an additional point to its right and in addition augmenting each of the two left points with a point to its right as a cluster spacer.)

Thus the ratio between the number of correctable such 4-erasures and the total number of such 4-erasures equals

$$\frac{3\binom{2p-1}{3} - p - 2\binom{p}{2}}{3\binom{2p-1}{3}} = 1 - \frac{p^2}{4p^3 - 12p^2 + 11p - 3} = 1 - \frac{9}{8p - 12} - \frac{1}{8p - 4} + \frac{1}{p - 1}.$$  

For $p = 11$, the ratio attains its minimal value of 0.9696. Moreover, it is readily seen that this ratio equals $1 - o(1)$, while $o(1)$ are terms that tend to zero as $p$ goes to infinity. 

VI. EFFICIENT DECODING OF ERASURES

In the previous two sections, the decodability of random and clustered erasures was proved by algebraic reasoning. In this section we take a more constructive path and study simple and efficient ways to decode random and clustered erasures. The purpose of this analysis is to prove that decoding the RC code can be done using $3kp + o(p^2)$ bit operations, while the best known algorithm for a 4-erasure correcting MDS code is $4kp + o(p^2)$ [9]. Since $k$ is taken to be in the order of $p$, saving about $kp$ bit operations gives a quadratic (in $p$) savings in computations that is very significant in practice for large $p$.

For the sake of the analysis, we only consider erasure of 4 information columns since these are the most common and most challenging cases. We moreover only consider erasures of two even columns and two odd columns, since for RC codes, the three even and one odd case (or three odd and one even), reduces to a correction of three even (or odd) erasures, preceded by a simple parity completion for the single odd (or even) column erasure. A very simple and efficient decoder for three odd-column erasures can be obtained by using the decoder of the STAR code, given in [11], and a same-complexity modification of that decoder can be used for the case of three even-column erasures. Throughout the section we will assume that one of the erased column is the leftmost even information column, as all other cases are cyclically equivalent.

A. Description of 4-erasure decoding algorithm

A general 4-erasure can be decoded using a straightforward procedure over $R_p$. Ways to perform the steps of that procedure in an efficient way are the topic of the next sub-section. The erased symbols, which represent the content
of the erased array columns, are denoted by \( \{e_1, o_1, e_2, o_2\} \). \( e_1, e_2 \) have even locations and \( o_1, o_2 \) have odd locations.

First the syndrome vector \( s \) of the array is calculated by taking the product

\[
s = Hr
\]

where \( r \) is the length 2\( p \) + 4 column vector over \( R_p \) that represents the array contents, with erased columns set to the zero element. Then the erased columns can be directly recovered by

\[
\begin{bmatrix}
  e_1 \\
o_1 \\
e_2 \\
o_2
\end{bmatrix}
= E^{-1}s
\]

(2)

where \( E \) denotes the \( 4 \times 4 \) sub-matrix of \( H \) that corresponds to the 4 erased columns’ locations:

\[
E = \begin{bmatrix}
  1 & 1 & 1 & 1 \\
  0 & \alpha_1^{-1} & 0 & \alpha_3^{-1} \\
  1 & 0 & \alpha_2 & 0 \\
  1 & \alpha_1 & \alpha_2 & \alpha_3
\end{bmatrix}
\]

Recall from (1) that each \( \alpha_i \) is an element in \( R_p \) of the form \( \alpha^{l_i} \), for some \( 0 < l_i < p \). Therefore, \( E \) can be written as

\[
E = \begin{bmatrix}
  1 & 1 & 1 & 1 \\
  0 & \alpha^{-u} & 0 & \alpha^{-w} \\
  1 & 0 & \alpha^{2v} & 0 \\
  1 & \alpha^u & \alpha^v & \alpha^w
\end{bmatrix}
\]

The inverse of \( E \), which is used in (2) to decode erasures, is now given in a closed form

\[
E^{-1} = (\alpha^u + \alpha^v + \alpha^w + \alpha^{u+v} + \alpha^{v+w})^{-1} \cdot \\
\left[
\begin{array}{cccc}
  1 + \alpha^u & 0 & 0 & 0 \\
  0 & \alpha^u + \alpha^w & 0 & 0 \\
  0 & 0 & 1 + \alpha^v & 0 \\
  0 & 0 & 0 & \alpha^u + \alpha^w
\end{array}
\right]
\]

\[
\cdot \\
\left[
\begin{array}{cccc}
  \alpha^{2u} & \alpha^u + \alpha^v & \alpha^u + \alpha^{2v} & \alpha^u + \alpha^w \\
  \alpha^u + \alpha^v & \alpha^u + \alpha^w & \alpha^u + \alpha^{v+w} & \alpha^u + \alpha^{v+w} \\
  \alpha^u & \alpha^u + \alpha^v & \alpha^u + \alpha^{2v} & \alpha^u + \alpha^{2v} \\
  \alpha^u + \alpha^v & \alpha^u + \alpha^w & \alpha^u + \alpha^{v+w} & \alpha^u + \alpha^{2v}
\end{array}
\right]
\]

From (2) and the closed-form expression above, the erased symbol \( e_1 \) can be recovered by the following product

\[
e_1 = \left( (\alpha^u + \alpha^v + \alpha^w + \alpha^{u+v} + \alpha^{v+w}) \cdot (1 + \alpha^v) \right)^{-1} \cdot \\
\left[
\begin{array}{cccc}
  \alpha^{2v} & \alpha^u + \alpha^v & \alpha^u + \alpha^{2v} & \alpha^u + \alpha^w \\
  \alpha^u + \alpha^v & \alpha^u + \alpha^w & \alpha^u + \alpha^{v+w} & \alpha^u + \alpha^{v+w} \\
  \alpha^u + \alpha^v & \alpha^u + \alpha^w & \alpha^u + \alpha^{v+w} & \alpha^u + \alpha^{2v} \\
  \alpha^u + \alpha^v & \alpha^u + \alpha^w & \alpha^u + \alpha^{v+w} & \alpha^u + \alpha^{2v}
\end{array}
\right] \cdot s
\]

Once \( e_1 \) is known, \( e_2 \) can be recovered using a simple parity completion with the aid of parity column \( R_0 \). The bits of the odd columns are then recovered by a chain of XOR operations with the aid of parity columns \( P, Q \), that can now be adjusted to \( P1, Q1 \) when all even columns are known.

Calculating \( e_1 \) then reduces to the following chain of calculations

1) Finding the inverse of \( (\alpha^u + \alpha^v + \alpha^w + \alpha^{u+v} + \alpha^{v+w}) \cdot (1 + \alpha^v) \) over \( R_p \).
2) Multiplication of sparse \( R_p \) elements by dense \( R_p \) elements. The sparse elements are the four elements from the \( E \) matrix (that have a small \( \leq 3 \) constant number of non-zero monomials, for any \( p \)) and the dense elements are the four syndrome elements.
3) Multiplication of two dense \( R_p \) elements resulting from the previous steps.
B. Analysis of 4-erasure decoding algorithm

We now analyze the number of bit operations required for each decoding task.

1) Finding inverses of $\mathcal{R}_p$ elements:
The inverse of an element $f(\alpha) \in \mathcal{R}_p$ is the element $\tilde{f}(\alpha)$ that satisfies $\tilde{f}(x)f(x) + a(x)M_p(x) = 1$, for some polynomial $a(x)$. When $f(\alpha)$ is invertible, the polynomial $\tilde{f}(x)$ can be found by the Extended Euclid Algorithm for finding the greatest common divisor of the polynomials $f(x)$ and $M_p(x)$. An efficient algorithm for polynomial greatest common divisors is given in [12, Ch.8] that requires $O(p \log^3 p)$ bit operations ($O(\log p)$ polynomial multiplications, each taking $O(p \log^2 p)$ bit operations, as shown in 3. below).

2) Multiplication of a sparse $\mathcal{R}_p$ element by a dense $\mathcal{R}_p$ element requires $O(p)$ bit operations. Since the number of non-zero monomials in the sparse polynomial is constant in $p$, the trivial polynomial multiplication algorithm requires $O(p)$ shifts and additions modulo 2.

3) Multiplication of two dense $\mathcal{R}_p$ elements can be done in $O(p \log^3 p)$ bit operations using Fourier domain polynomial multiplication. We describe this procedure for the special case of polynomial coefficients over $GF(2)$. Let $N \geq 2p - 2$ be the smallest such integer of the form $N = 2^k - 1$. Let $\omega$ be a principal $N$th root of unity in the finite field $GF(2^k)$. Then $\omega$ defines a Discrete Fourier Transform on length $N$ vectors over $GF(2^k)$. The product of two polynomials of degree $p - 2$ or less can be obtained by element-wise multiplication of their individual Discrete Fourier Transforms, and then applying the Inverse Discrete Fourier Transform to the resulting length $N$ vector. Using the FFT algorithm, each transformation requires $O(N \log N)$ operations over $GF(2^k)$, or $O(N \log^3 N)$ bit operations. The element-wise multiplication requires $N$ multiplications over $GF(2^k)$, or $O(N \log^2 N)$ bit operations. Since $N < 4p$, the total number of bit operations needed for multiplying two dense $\mathcal{R}_p$ elements is $O(p \log^3 p)$.

VII. RELIABILITY ANALYSIS OF RC-CODE PROTECTED DISK ARRAYS

The main motivation for the construction of array codes in general, and RC codes in particular, is to provide efficient protection for disk arrays against device failures. The benefit of deploying an array code in a practical storage system obviously lies in the trade-off it achieves between erasure correction capability and implementation complexity. To this end, the correction capability characterization of RC codes, and their benefits in implementation complexity were derived in concrete terms that can be clearly specified to the storage-system operator. However, to achieve an effective data protection using those codes, one needs to instill these specifications into a statistical framework for analyzing the reliability of the stored data. For a time instance $t$, the reliability of a disk array is defined as the probability that no data has been lost at time $t$ [13]. Finding the full reliability distribution for all times $t$ is hard except for very simple protection structures. Therefore, the expected time before data loss, denoted $MTTDL$ (Mean Time To Data Loss), is used in practice as a quantitative indicator for the system reliability. Ultimately, this section will detail a procedure to find the $MTTDL$ of RC-code protected disk arrays in the presence of Random and Clustered device failures. This will be done after first presenting the general method of $MTTDL$ calculation as applied in the literature to MDS codes under Random failures.

A. $MTTDL$ calculation for MDS codes under Random failures

Using the method presented in [13, Ch.5] for single-erasure-correcting arrays under Random failures (termed Independent disk lifetimes therein), we calculate the $MTTDL$ of all-4-erasure-correcting arrays as an example that will be later used for comparison with RC codes. The direct calculation of the $MTTDL$ becomes a simpler task if disk failures and repairs follow a Markov process and can thus be described by Markov state diagram. To allow that, the following assumptions are made.

- Disk lifetimes follow an exponential distribution with equal mean $MTTF_{disk} = 1/\lambda$.
- Repair times are also exponential with mean $MTTR_{disk} = 1/\mu$.
- The number of disks is large compared to the number of tolerable failures so the transition probabilities between states do not depend on the instantaneous number of failed disks.

When those assumptions are met, the reliability of a disk array can be described by the state diagram shown in Figure 4. The label of each state represents the number of failed disks. State $F$ (Fail) represents permanent data loss resulting

1$MTTF$ stands for Mean Time To Failure while $MTTR$ stands for Mean Time To Repair
Fig. 4. State diagram description of all-4-erasure-correcting arrays under Random failures

from a failure count that is above the array tolerance. The exponential distributions allow specifying the transitions between states in terms of rates. The transition rate from lower to higher states is the inverse $MTTF_{\text{disk}}$ of individual disks, times the number of disks in the array. The reverse transitions that represent repairs have rates that are the inverse $MTTR_{\text{disk}}$ assumed in the system. Using the state diagram, the $MTTDL$ is the expected time beginning in state 0 and ending on the transition into state $F$.

$$MTTDL \triangleq E[0 \rightarrow F]$$

The Markov property of the process permits the decomposition

$$E[0 \rightarrow F] = E[\text{time stays in } 0] + E[1 \rightarrow F] = \frac{1}{n\lambda} + E[1 \rightarrow F]$$

Linear relationships between $E[i \rightarrow F]$ and $E[j \rightarrow F]$ are induced whenever state $i$ and state $j$ are connected. The $MTTDL$ is then obtained as the solution (for $E[0 \rightarrow F]$) of the following linear system.

$$
\begin{bmatrix}
1 & -1 & 0 & 0 & 0 \\
\frac{\mu}{\mu + n\lambda} & 1 & -\frac{n\lambda}{\mu + n\lambda} & 0 & 0 \\
0 & \frac{\mu}{\mu + n\lambda} & 1 & -\frac{n\lambda}{\mu + n\lambda} & 0 \\
0 & 0 & \frac{\mu}{\mu + n\lambda} & 1 & -\frac{n\lambda}{\mu + n\lambda} \\
0 & 0 & 0 & \frac{\mu}{\mu + n\lambda} & 1
\end{bmatrix}
\begin{bmatrix}
E[0 \rightarrow F] \\
E[1 \rightarrow F] \\
E[2 \rightarrow F] \\
E[3 \rightarrow F] \\
E[4 \rightarrow F]
\end{bmatrix}
= 
\begin{bmatrix}
\frac{1}{n\lambda} \\
\frac{1}{\mu + n\lambda} \\
\frac{1}{\mu + n\lambda} \\
\frac{1}{\mu + n\lambda} \\
\frac{1}{\mu + n\lambda}
\end{bmatrix}
$$

that is found to be

$$MTTDL_{\text{MDS4}} = \frac{1}{\Lambda^5}(5\Lambda^4 + 4\mu\Lambda^3 + 3\mu^2\Lambda^2 + 2\mu^3\Lambda + \mu^4)$$

where $\Lambda \triangleq n\lambda$ was used for notational convenience.

**B. $MTTDL$ calculation for RC codes under Random and Clustered failures**

For the model of Random failures, the $MTTDL$ of RC codes can be calculated by a straightforward application of the method in the previous subsection - executed on the transition diagram of Figure 5. The corresponding linear

$$
\begin{bmatrix}
\Lambda \\
\Lambda \\
\Lambda \\
7\Lambda/8 \\
\Lambda
\end{bmatrix}
\begin{bmatrix}
0 \\
1 \\
2 \\
3 \\
4 \\
\Lambda
\end{bmatrix}
$$

Fig. 5. State diagram description of RC-coded arrays under Random failures
The solution of that system gives

\[MTTDL_{RC,random} = \frac{39\Lambda^4 + 35\mu\Lambda^3 + 26\mu^2\Lambda^2 + 17\mu^3\Lambda + 8\mu^4}{8\Lambda^5 + \mu\Lambda^4}\]

The exact \(MTTDL\) calculations are now used to compare the reliability of RC codes to the reliabilities of 4-Random and 3-Random failure-correcting codes. For the comparison, \(\Lambda\) is fixed to be \(100/8760\) [1/hr], which applies, e.g., to an array with 100 disks and \(MTTF_{\text{disk}} = 1\) [Year]. The \(MTTDL\) in hours ([hr]) is then calculated for repair rates \(\mu\) between 0.01 [1/hr] and 10 [1/hr]. The graph shows that RC codes outperform 3-Random failure codes by an order of magnitude, despite having the same encoding complexity, the same update complexity and asymptotically the same decoding complexity. However, when Random failures only is assumed, RC codes are still two orders of magnitude below 4-Random failure-correcting codes.

To compare RC codes and 4-Random failure codes in the presence of both Random and Clustered codes, the state diagram of RC codes in Figure 5 needs to be modified to comprise additional states that represent Clustered errors. The state diagram of 4-Random failure codes in Figure 4 remains the same since this code is oblivious to the distinction between Random and Clustered failures. To take Clustered failures into account in the Markov failure model, we add the following assumptions to those of the previous sub-section.

- Times to Clustered failures (failures that are adjacent to an unrepaired previous failure) are exponentially distributed with mean \(1/\chi\).
- The exponentially distributed repair process eliminates isolated failures before clustered ones.

With these assumptions, the state diagram of RC-code-protected arrays with Random and Clustered failures is given in Figure 7. States 2', 3' and 4' in the upper branch represent 2, 3 and 4 Clustered (not all-isolated) failures, respectively.
Fig. 7. State diagram description of RC-coded arrays under Random and Clustered failures

The transitions marked with $\chi$ represent moving from all-isolated failures to a Clustered failure combination. At the upper branch, both Random and additional Clustered failures result in a Clustered failure combinations - and that accounts for the transitions marked $\Lambda + \chi$. From state 0 a Clustered failure is not well defined, but the rate $\chi$ is added to the transition to maintain consistency with respect to the total failure rate ($\Lambda + \chi$) outgoing from all other states.

Solving the $8 \times 8$ linear system for the diagram in Figure 7, the $MTTDL$ can be calculated for all combinations of $\chi, \Lambda, \mu$. This ability to have a closed form expression for the $MTTDL$ of RC codes, under both Random and Clustered failures, is crucial for a system operator to predict the reliability of the storage array under more realistic failure assumptions. The resulting $MTTDL$ curves for RC codes under three different $\chi$ values are plotted in Figure 8, and compared to the $MTTDL$ of a 4-Random failure code under the same conditions (4-Random codes give the same $MTTDL$ independent of the ratio between $\chi$ and $\Lambda$, as long as their sum is fixed).

Fig. 8. Comparison of $MTTDL$ curves for 4-Random and RC codes, for three values of $\chi$
VIII. Code Evaluation and Comparison with Existing Schemes

We compare RC codes to EVENODD ($r = 4$) codes using various performance criteria. The failure-correction properties in Table I apply for any prime $p$ such that $2$ is primitive in $GF(p)$.

<table>
<thead>
<tr>
<th>Code Length (up to)</th>
<th>RC Codes</th>
<th>4-EVENODD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Redundancy</td>
<td>2$p$</td>
<td>$p$</td>
</tr>
<tr>
<td>Encoding Complexity</td>
<td>3$k$p</td>
<td>4$k$p</td>
</tr>
<tr>
<td>Decoding Complexity</td>
<td>3$k$p</td>
<td>4$k$p</td>
</tr>
<tr>
<td>Update Complexity</td>
<td>5</td>
<td>7</td>
</tr>
<tr>
<td>Clustered Failures</td>
<td>All</td>
<td>All</td>
</tr>
<tr>
<td>Random Failures</td>
<td>$7/8$</td>
<td>All</td>
</tr>
</tbody>
</table>

**TABLE I**
Comparison of RC Codes and EVENODD Codes

The redundancy $r$ is 4 for both codes. RC codes can support up to $2p$ information columns while EVENODD can only have up to $p$. Since parity columns $R_0$ and $R_1^*$ each depend on half of the information columns, the encoding complexity of RC codes is $3kp$, compared to $4kp$ in EVENODD. In both cases, when $k$ is of the same order of $p$, the decoding complexity is dominated by syndrome calculations (for RC codes this has been shown in Section VI). Therefore, similarly to the encoding case, RC codes need about $3kp$ bit operations to decode, compared to $4kp$ for EVENODD. As for the update-complexity, RC codes are significantly more efficient. Their small-write update complexity is $5$. Each of the $2p(p−1)$ updated information bits needs $3$ parity update, $P, Q, R_0$ for bits in even columns and $P, Q, R_1^*$ for bits in odd columns. The $4(p−1)$ bits that belong to EO diagonals ($2p(p−1)$ in $Q$ and $p−1$ in each of $R_0, R_1^*$) require additional $p−1$ parity bit updates each for adjusting even/odd parities. The small-write update-complexity of RC is then obtained by averaging

$$\frac{6p(p−1) + 4(p−1)^2}{2p(p−1)} = 5 − o(1)$$

Recall that EVENODD has small-write update-complexity of $2r−1−o(1) = 7−o(1)$. The full-column update-complexity of RC is 3 while EVENODD’s is 4. Thus RC offers a $28.57\%$ improvement in the average number of small-writes and $25\%$ improvement in the number of full-column updates. The fraction of Clustered erasures correctable by RC codes is $1 − o(1)$, essentially the same as EVENODD’s 1 fraction. Only in Random erasure-correction capability are RC codes slightly inferior to EVENODD codes, the fraction of correctable Random erasures is $7/8 − o(1)$ compared to 1 for EVENODD.

**REFERENCES**


