STAR : An Efficient Coding Scheme for Correcting Triple Storage Node Failures

Cheng Huang
Microsoft Research
One Microsoft Way,
Redmond, WA 98052
Email: cheng.huang@microsoft.com

Lihao Xu
Wayne State University
5143 Cass Avenue, 431 State Hall,
Detroit, MI 48202
Email: lihao@cs.wayne.edu

Abstract

Proper data placement schemes based on erasure correcting code are one of the most important components for a highly available data storage system. For such schemes, low decoding complexity for correcting (or recovering) storage node failures is essential for practical systems. In this paper, we describe a new coding scheme, which we call the STAR code, for correcting triple storage node failures (erasures). The STAR code is an extension of the double-erasure-correcting EVENODD code, and a modification of the generalized triple-erasure-correcting EVENODD code. The STAR code is an MDS code, and thus is optimal in terms of node failure recovery capability for a given data redundancy. We provide detailed STAR code's decoding algorithms for correcting various triple node failures. We show that the decoding complexity of the STAR code is much lower than those of the existing comparable codes, thus the STAR code is practically very meaningful for storage systems that need higher reliability.

1 Introduction

In virtually all information systems, it is essential to have a reliable data storage system that supports data availability, persistence and integrity. Here we refer to a storage system in general sense: it can be a disk array, a network of storage nodes in a clustered environment (SAN or NAS), or a wide area large scale P2P network. In fact, many research and development efforts have been made to address various issues of building reliable data storage systems to ensure data survivability, reliability, availability and integrity, including disk arrays, such as the RAID [14], clustered systems, such as the NOW [2] and the RAIN [12], distributed file systems, such as the NFS (Network File System) [39], HA-NFS [4], xFS [3], AFS [36], Zebra [23], CODA [37], Sprite [28], Scotch [20] and BFS [13], storage systems, such as NASD [19], Petal [25] and PASIS [42], and large scale data distribution and archival networks, such as Intermemory [21], OceanStore [24] and Logistical Network [33].

As already indicated by these efforts, proper data redundancy is the key to provide high reliability, availability and survivability. Evolving from simple data replication or data striping in early clustered data storage systems, such as the RAID system [14], people have realized it is more economical and efficient to use the so-called *threshold schemes* to distribute data over multiple nodes in distributed storage systems [42, 41, 21, 24] than naive (multi-copy) replications. The basic idea of threshold schemes is to map an original data item into n pieces, or *shares*, using certain mathematical transforms. Then all the n shares are distributed to n nodes in the system, with each node having one share. (Each node is a storage unit, which can be a disk, a disk array or even a clustered subsystem.) Upon accessing the data, a user needs to collect at least k shares to retrieve the original data, i.e., the original data can be *exactly* recovered from m different shares if $m \geq k$, but less than k shares will not recover the original data. Such threshold schemes are called (n, k)-threshold schemes. The threshold schemes can be realized by a few means. To maximize the usage of network and storage capacity, and to eliminate bottlenecks in a distributed storage system, each data share should be of the same size. Otherwise the failure of a node storing a share with bigger size will have bigger impact on the system performance, thus creating a bottleneck in the system.

From *error control code* point of view, an (n, k)-threshold scheme with equal-size shares is equivalent to an (n, k) block code, and especially most (n, k)-threshold schemes are equivalent to (n, k) MDS (*Maximum Distance Separable*) codes [27, 26]. An (n, k) error control code uses mathematical means to transform a k-symbol message data block to an n-symbol codeword block such that any m symbols of the codeword block can recover all the k symbols of the original message data block, where $k \leq m \leq n$. All the data symbols are of the same size in bits. Obviously by the simple *pigeon hole* principle, $k \leq m$. When $m = k$, such an (n, k) code is called MDS code, or meets the *Singleton Bound* [26]. Hereafter we simply use (n, k) to refer to any data distribution scheme using an (n, k) MDS code. Using coding terminology, each share of (n, k) is called a data *symbol*. The process of creating n data symbols from the original data whose size is of k symbols is called *encoding*, and the corresponding process of retrieving the original data from at least arbitrary k data symbols stored in the system is called *decoding*.

It is not hard to see an (n, k) scheme can tolerate up to $(n - k)$ node failures at the same time, and thus achieve data reliability, or data *survivability* in case the system is under attack where some nodes can not function normally. The (n, k) scheme can also ensure the *integrity* of data distributed in the system, since an (n, k) code can be used to detect data modifications on up to $(n - k)$ nodes. $r = n - k$ is a parameter that can describe the *reliability degree* of an (n, k) scheme.

While the concept of (n, k) has been well understood and suggested in various data storage projects, virtually all practical systems use the Reed-Solomon (RS) code [35] as an MDS code. (The so-called *information dispersal algorithm* [34] used in some schemes or systems [1] is indeed just a RS code.) The computation overhead of using the RS code, however, is large, as demonstrated in several projects, such as in OceanStore [24]. Thus practical storage systems seldom use a general (n, k) MDS code, except for full replication (which is an $(n, 1)$) or stripping without redundancy (corresponding to (n, n)) or single parity (which is $(n, n - 1)$). The advantages of using (n, k) schemes are hence very limited if not totally lost.

It is hence very important and useful to design general (n, k) codes with *both* MDS property and simple encoding and decoding operations. MDS *array codes* are such a class of codes with the both properties.

Array codes have been studied extensively [17, 22, 8, 5, 7, 43, 44, 6, 15]. A common property of these codes is that their encoding and decoding procedures use only simple binary *XOR* (*exclusive OR*) operations, which can be easily and most efficiently implemented in hardware and/or software, thus these codes are more efficient than the RS code in terms of computation complexity.

In an array code, each of the n (information or parity) symbols contain l "bits", where a bit could be binary or from a larger alphabet. The code can be arranged in an array of size $n \times l$, where each element of the array corresponds to a bit. (When there is no ambiguity, we refer to array elements also as *symbols* for representation convenience.) Mapping to a storage system, all the symbols in a same column are stored in the same storage node. If a storage node fails, then the corresponding column of the code is considered to be an *erasure*. (Here we adopt a commonly-used storage failure model, as discussed in [5, 15], where all the symbols are lost if the host storage node fails.)

A few class of MDS array codes have been successfully designed to recover double (simultaneous) storage node failures, i.e., in coding terminology, codes of distance 3 which can correct 2 erasures [26]. The recent ones include the EVENODD code [5] and its variations such as the RDP scheme [15], the X-Code [43], and the B-Code [44].

As storage systems expand, it becomes increasingly important to have MDS array codes of distance 4, which can correct 3 erasures, i.e., codes which can recover from *triple* (simultaneous) node failures. (There have been parallel efforts to design near-optimal codes, i.e., non-MDS codes, to tolerate triple failures, e.g. recent results from [32].) Such codes will be very desirable in large storage systems, such as the Google File System [18]. To the best of our knowledge, there exist only few classes of MDS array codes of distance 4: the generalized EVENODD code [7, 6] and later the Blaum-Roth code [9]. (There have been unsuccessful attempts resulting in codes that are not MDS [40, 29], which we will not discuss in detail in this paper.) The Blaum-Roth code is non-systematic, which requires decoding operations in any data retrieval even without node failures and thus probably is not desirable in storage systems. The generalized EVENODD code is already much more efficient than the RS code in both encoding and decoding operations. But a natural question we ask is: can its decoding complexity be further reduced? In this paper, we provide a positive answer with a new coding scheme, which we call the *STAR* code.

The STAR code is an alternative extension of the EVENODD code, a $(k + 3, k)$ MDS code which can recover triple node failures (erasures). The structure of the code is very similar to the generalized EVENODD code and their encoding complexities are also the same. Our key contribution, however, is to exploit the geometric property of the EVENODD code, and provide a new construction for an additional parity column. The difference in construction of the third parity column leads to a more efficient decoding algorithm than the generalized EVENODD code for triple erasure recovery. Our analysis shows the decoding complexity of the STAR code is very close to 3 XORs per bit (symbol), the theoretical *lower bound*, even when k is small, where the generalized EVENODD could need up to 10 XORs (Section 7) per bit (symbol). Thus the STAR code is perhaps the most efficient existing code in terms of decoding complexity when recovering from triple erasures.

It should be noted that the original generalized EVEN-

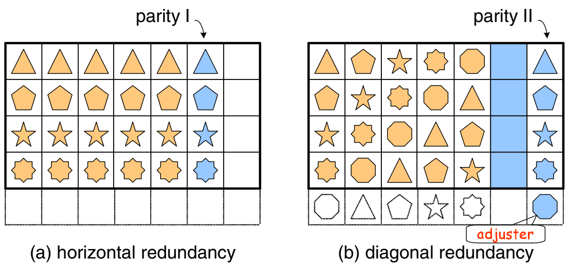

Figure 1: EVENODD Code Encoding

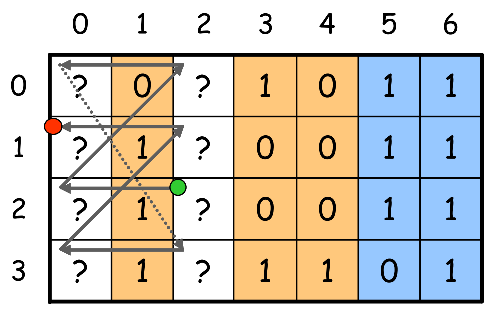

Figure 2: EVENODD Code Decoding

ODD papers [7, 6] only provide generic erasure decoding algorithms for multiple erasures. It might be possible to design a specific triple-erasure decoding algorithm to reduce decoding complexity of the generalized EVENODD. It is, however, not clear whether such a decoding algorithm for the generalized EVENODD code can achieve the same complexity as the STAR code. The interested readers thus are welcome to design an optimized triple-erasure decoding algorithm for the generalized EVENODD code and compare its performance with the STAR code.

This paper is organized as follows: we first briefly describe the EVENODD code, base on which the STAR code encoding is derived in the following section. In Section 4, we constructively prove that the STAR code can correct any triple erasures by providing detailed decoding algorithms. We also provide an algebraic description of the STAR code and show that the STAR code's distance is 4 in Section 5. We then analyze and discuss the STAR decoding complexity in Section 6 and make comparisons with two related codes in Section 7. We further share our implementation and performance tests of the STAR code in Section 8, and conclude in Section 9.

2 EVENODD Code: Double Erasure Recovery

2.1 EVENODD Code and Encoding

We first briefly describe the EVENODD code [5], which was initially proposed to address disk failures in disk array systems. Data from multiple disks form a two dimensional array, with one disk corresponding to one column of the array. A disk failure is equivalent to a column erasure. The EVENODD code uses two parity columns together with p information columns (where p is a prime number. As already observed [5, 15], p being prime in practice does not limit the k parameter in real system configuration with a simple technique called codeword *shortening* [26]. The code ensures that all information columns are fully recoverable when *any* two disks fail. In this sense, it is an optimal 2-erasure correcting code, i.e., it is an $(p+2,\ p,\ 3)$ MDS code. Besides this MDS property, the EVENODD code is computationally efficient in both encoding and decoding, which needs only XOR operations.

The encoding process considers a $(p-1)\times(p+2)$ array, where the first p columns are information columns and the last two parity columns. Symbol $a_{i,j}$ $(0 \le i \le p-2,\ 0 \le j \le p+1)$ represents symbol i in column j. A parity symbol in column p is computed as the XOR sum of all information symbols in the same row. The computation of column $(p+1)$ takes the following steps. First, the array is augmented with an imaginary row $p-1$, where all symbols are assigned *zero* values (note that all symbols are binary ones). The XOR sum of all information symbols along the same diagonal (indeed a diagonal of *slope* 1) is computed and assigned to their corresponding parity symbol, as marked by different shapes in Figure 1. Symbol $a_{p-1,p+1}$ now becomes non-zero and is called the EVENODD *adjuster*. To remove this symbol from the array, *adjuster complement* is performed, which adds (XOR addition) the adjuster to all symbols in column $p+1$.

The encoding can be algebraically described as follows $(0 \le i \le p-2)$:

$$a_{i,p} = \bigoplus_{j=0}^{p-1} a_{i,j}$$

$$a_{i,p+1} = S_1 \oplus \left(\bigoplus_{j=0}^{p-1} a_{\langle i-j\rangle_p, j} \right),$$

$$\text{where } S_1 = \bigoplus_{j=0}^{p-1} a_{\langle p-1-j\rangle_p, j}.$$

Here, S_1 is the EVENODD adjuster and $\langle x\rangle_p$ denotes $x \bmod p$. Refer to [5] for more details.

2.2 EVENODD Erasure Decoding

The EVENODD code is an optimal double erasure correcting code and any two column erasures in a coded block can be fully recovered. Regarding to the locations of the erasures, [5] divides decoding into four cases. Here, we only summarize the most common one, where neither of the erasures is a parity column. Note that the other three cases are special ones and can be dealt with easily. A decoder first computes horizontal and diagonal *syndromes* as the XOR sum of all available symbols along those directions. Then a *starting* point of decoding can be found, which is guaranteed to be the only erasure symbol in its diagonal. The decoder recovers this symbol and then moves horizontally to recover the symbol in the other erasure column. It then moves diagonally to the next erasure symbol and horizontally again. Upon completing this *Zig-Zag* process, all erasure symbols are fully recovered. In the example shown in Figure 2 ($p = 5$), the starting point is symbol $a_{2,2}$ and the decoder moves from $a_{2,2}$ to $a_{2,0}$, $a_{0,2}$, $a_{0,0} \cdots$ and finally completes at $a_{1,0}$.

3 STAR Code Encoding: Geometric Description

Extending from the EVENODD code, the STAR code consists of $p + 3$ columns, where the first p columns contain information data and the last 3 columns contain parity data. The STAR code uses the exact same encoding rules of the EVENODD code for the first two parity columns, i.e., without the third parity column, the STAR code is just the EVENODD code. The extension lies in the last parity column, column $p + 2$. This column is computed very similar to column $p + 1$, but along diagonals of slope -1 instead of slope 1 as in column $p + 1$. (The original generalized EVENODD code [7, 6] uses slope 2 for the last parity column. That is the only difference between the STAR code and the generalized EVENODD code. However, as will be seen from the following section, it is this difference that makes it much easier to design a much more efficient decoding algorithm for correcting triple erasures.) For simplicity, we call this *anti-diagonal* parity. The procedure is depicted by Figure 3, where symbol $a_{p-1,p+2}$ in parity column $p+2$ is also an *adjuster*, similar to the EVENODD code. The adjuster is then removed from the final code block by adjuster complement. Algebraically, the encoding of parity column $p + 2$ can be represented as ($0 \le i \le p - 2$):

$$a_{i,p+2} = S_2 \oplus \left(\bigoplus_{j=0}^{p-1} a_{\langle i+j \rangle_p, j} \right), \text{ where } S_2 = \bigoplus_{j=0}^{p-1} a_{\langle j-1 \rangle_p, j}.$$

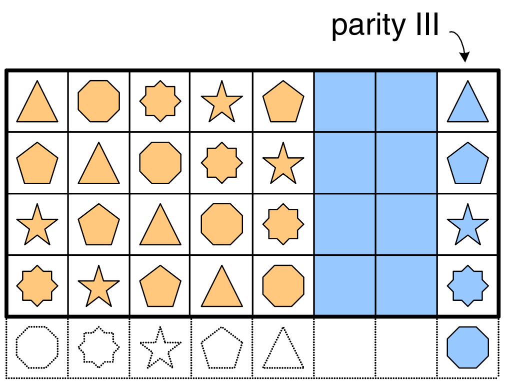

Figure 3: STAR Code Encoding

4 STAR Code Erasure Decoding

The essential part of the STAR code is the erasure decoding algorithm. As presented in this section, the decoding algorithm involves pure XOR operations, which allows efficient implementation and thus is suitable for computation/energy constrained applications. The MDS property of the STAR code, which guarantees the recovery from arbitrary triple erasures, is explained along with the description of the decoding algorithm. A mathematical proof of this property will be given in a later section.

The STAR code decoding can be divided into two cases based on different erasure patterns: 1) decoding without parity erasures, where all erasures are information columns; and 2) decoding with parity erasures, where at least one erasure is a parity column. The former case is harder to decode and is the focus of this section. This case in turn can be divided into two subcases: symmetric and asymmetric, based on whether the erasure columns are evenly spaced. The latter case, on the other hand, handles several special situations and is much simpler.

4.1 Decoding without Parity Erasures: Asymmetric Case

We consider the recovery of triple information column erasures at position r, s and t ($0 \le r,\ s,\ t \le p - 1$), among the total $p + 3$ columns. (Note: hereafter, sometimes we also use r to denote a column position. It should be easy to distinguish a column position r from a code's reliability degree $r = n - k$ from the contexts.) Without loss of generality, assume $r < s < t$. Let $u = s - r$ and $v = t - s$. The asymmetric case deals with erasure patterns satisfying $u \neq v$.

The decoding algorithm can be visualized with a concrete example, where $r = 0$, $s = 1$, $t = 3$ and $p = 5$, as shown in Figure 4(a), where empty columns are erasures.

The decoding procedure consists of the following four steps:

4.1.1 Recover Adjusters and Calculate Syndromes

Given the definitions of the adjusters S_1 and S_2, it is easy to see that they can be computed as the XOR sums of all symbols in parity columns 5, 6 and 5, 7, respectively.

Then the adjusters are assigned to symbols $a_{4,6}$, $a_{4,7}$ and also applied through XOR additions to all of the rest parity symbols in columns 6, 7, which is to reverse the adjuster complement. The redundancy property of the coded block states that the XOR sum of all symbols along any parity direction (horizontal, diagonal and anti-diagonal) should equal to *zero*. Due to erasure columns, however, the XOR sum of the rest symbols is non-zero and we denote it as the *syndrome* for this parity direction. To be specific, syndrome $\tilde{s}_{i,j}$ denotes the XOR sum of parity symbol $a_{i,j+p}$ and its corresponding non-erasure information symbols. For example, $\tilde{s}_{0,0} = a_{0,5} \oplus a_{0,2} \oplus a_{0,4}$ and $\tilde{s}_{0,1} = a_{0,6} \oplus a_{3,2} \oplus a_{1,4}$, etc. To satisfy the parity property, the XOR sum of all erasure information symbols along any redundancy direction needs to match the corresponding syndrome. For example, $\tilde{s}_{0,0} = a_{0,0} \oplus a_{0,1} \oplus a_{0,3}$ and $\tilde{s}_{0,1} = a_{0,0} \oplus a_{4,1} \oplus a_{2,3}$, etc.

In general, this step can be summarized as:
1) adjusters recovery ($j = 0,\ 1,\ 2$),

$$S_j = \bigoplus_{i=0}^{p-2} a_{i,p+j},$$

$S_1 = S_0 \oplus S_1$ and $S_2 = S_0 \oplus S_2$;
2) reversion of adjuster complement ($0 \leq i \leq p-2$),

$$a_{i,p+1} = a_{i,p+1} \oplus S_1,$$
$$a_{i,p+2} = a_{i,p+2} \oplus S_2;$$

3) syndrome calculation

$$\tilde{s}_{i,0} = a_{i,0} \oplus \left(\bigoplus_{j=0}^{p-1} a_{i,j} \right),$$
$$\tilde{s}_{i,1} = a_{i,1} \oplus \left(\bigoplus_{j=0}^{p-1} a_{\langle p+i-j \rangle_p,j} \right),$$
$$\tilde{s}_{i,2} = a_{i,2} \oplus \left(\bigoplus_{j=0}^{p-1} a_{\langle i+j \rangle_p,j} \right),$$

where $0 \leq i \leq p-1$ and $j \neq r$, s or t.

4.1.2 Finding a Starting Point

Recall that finding a starting point is the key step of the EVENODD decoding, which seeks one particular diagonal with only one *unknown* symbol. This symbol can then be recovered from its corresponding syndrome, and it enables the Zig-Zag decoding process until all unknown symbols are recovered. In the STAR decoding, however, it is *impossible* to find any parity direction (horizontal, diagonal or anti-diagonal) with only one unknown symbol. Therefore, the approach adopted in the EVENODD decoding does *not* directly apply here, and additional steps are needed to find a starting point.

For illustration purpose, we now assume all syndromes are represented by the shadowed symbols in the three parity columns, as shown in Figure 4(b). Based on the diagonal parity property, it is clear that $\tilde{s}_{3,1}$ equals to the XOR sum of three unknown symbols $a_{3,0}$, $a_{2,1}$ and $a_{0,3}$, as marked by "$\triangle$" signs in Figure 4(b). Similarly, $\tilde{s}_{0,2} = a_{0,0} \oplus a_{1,1} \oplus a_{3,3}$, which are all marked by "$\triangledown$" signs along an anti-diagonal. Imagine that all these marked symbols in the erasure information columns altogether form a *cross* pattern, whose XOR sum is computable ($\tilde{s}_{3,1} \oplus \tilde{s}_{0,2}$ in this case). The *key* of this step is to choose multiple crosses, such that the following two conditions are satisfied:

Condition 1
1) each cross is shifted vertically downward from a previous one by v symbols (offset)*;*
2) the bottom row of the final cross (after wrapping around) steps over (coincides with) *the top row of the first cross.*

In our particular example, two crosses are chosen. The second cross is $v = 2$ symbols offset from the first one and consists of erasure symbols $a_{0,0}$, $a_{4,1}$, $a_{2,3}$ (marked by "$\triangle$") and $a_{2,0}$, $a_{3,1}$, $a_{0,3}$ (marked by "$\triangledown$"), as shown in Figure 4(c). It is straightforward that the XOR sum of these two crosses equals to $\tilde{s}_{3,1} \oplus \tilde{s}_{0,2} \oplus \tilde{s}_{0,1} \oplus \tilde{s}_{2,2}$.

Notice, on the other hand, the calculation (XOR sum) of these two crosses includes symbols $a_{0,0}$ and $a_{0,3}$ twice, the result of the bottom row of the second cross stepping over the top row of the first one. Thus, their values are canceled out and do *not* affect the result. Also notice that the parities of unknown symbol sets ($a_{2,0}$, $a_{2,1}$ and $a_{2,3}$) and ($a_{3,0}$, $a_{3,1}$ and $a_{3,3}$) can be determined by horizontal syndromes $\tilde{s}_{2,0}$ and $\tilde{s}_{3,0}$ (marked by "$\bigcirc$"), respectively. Thus, we can get

$$a_{1,1} \oplus a_{4,1} = \tilde{s}_{3,1} \oplus \tilde{s}_{0,2} \oplus \tilde{s}_{0,1} \oplus \tilde{s}_{2,2} \oplus \tilde{s}_{2,0} \oplus \tilde{s}_{3,0},$$

as all marked in Figure 4(d).

Repeating this process and starting the first cross at different rows, we can obtain the XOR sum of any unknown symbol pair with a fixed distance 3 in column 1, i.e. $a_{0,1} \oplus a_{3,1}$, $a_{2,1} \oplus a_{0,1}$, etc.

From this example, we can see that the first condition of choosing crosses ensures the alignment of unknown symbols in the middle erasure column with those in the side erasure columns. Essentially, it groups unknown symbols together and replaces them with known syndromes. This is one way to cancel unknown symbols

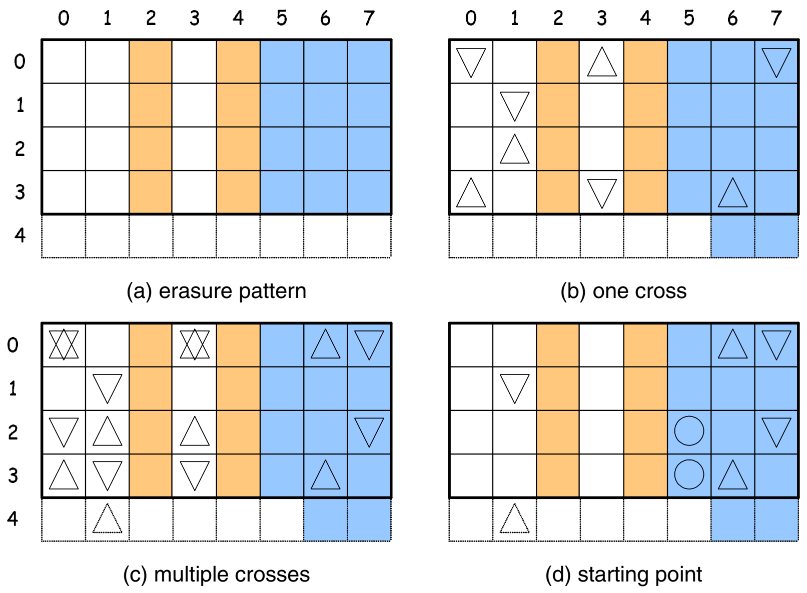

Figure 4: STAR Code Decoding

and results in a chain of crosses. The other way to cancel unknown symbols comes from the second condition, where unknown symbols in the *head row* (the first row of the first cross) of the cross chain are canceled with those in the *tail row* (the bottom row of the final cross). This is indeed "gluing" the head of the first cross with the tail of the last one and turns the chain into a *ring*. The number of crosses in the ring is completely determined by the erasure pattern (r, s and t) and the STAR code parameter p. The following Lemma 1 ensures the existence of such a ring for any given $u = s - r$, $v = t - s$ and p.

Lemma 1 *A ring satisfying* Condition *1 always exists and consists of l_d ($0 \le l_d < p$) crosses, where l_d is determined by the following equation:*

$$\langle u + l_d v \rangle_p = 0, \tag{1}$$

where $0 \le u,\ v < p$.

Proof. Since p is a prime number, integers modulo p define a finite field $GF(p)$. Let v^{-1} be the unique inverse of v in this field. Then, $l_d = (p - u)v^{-1}$ exits and is unique.

Given a ring, rows with 3 unknown symbols are substituted with horizontal syndromes (*substitution*), and symbols being included even times are simply removed (*simple cancellation*). For simplicity, we refer both cases as *cancellations*. Eventually, there are exactly two rows left with unknown symbols, which is confirmed by the following Lemma 2.

Lemma 2 *After cancellations, there are exact two rows with unknown symbols in a ring. The row numbers are u and $p - u$, as offsets from the top row of the first cross.*

Proof. To simplify the proof, we only examine the ring, whose first cross starts at row 0. Now the first cross contains two unknown symbols in column r and they are in rows 0 and $u + v$. We can represent them with a polynomial $(1 + x^{u+v})$, where power values (modulo p) of x correspond to row entices. Similarly, the unknown symbols in column s can be represented as $(x^u + x^v)$. Therefore, the first cross can be completely represented by $(1 + x^{u+v} + x^u + x^v)$ and the $l_1{}^{th}$ cross by

$$(1 + x^{u+v} + x^u + x^v)x^{l_1 v},$$

where $0 \le l_1 < l_d$ and the coefficients of x are binary. Note that we don't explicitly consider unknown symbols in column t, which are reflected by polynomials representing column r. Using this representation, the cancellation of a polynomial term includes both cases of substitution and simple cancellation. The XOR sum of all crosses is as

$$\begin{aligned}
&\sum_{l_1=0}^{l_d-1} (1 + x^{u+v} + x^u + x^v)x^{l_1 v} \\
=&(1 + x^u) \sum_{l_1=0}^{l_d-1} (1 + x^v)x^{l_1 v} \\
=&(1 + x^u)(1 + x^{p-u}) \\
=&x^u + x^{p-u},
\end{aligned} \tag{2}$$

where l_d is substituted using the result from Lemma 1. Thus, only two rows with unknown symbols are left after cancellations and the distance between them is $d = \langle p - 2u \rangle_p$.

It is important to point out that unknown symbols in the remaining two rows are *not* necessarily in column s. For example, if $r = 0$, $s = 2$ and $t = 3$, the remaining unknown symbols would be $a_{2,0}$, $a_{2,3}$, $a_{3,0}$ and $a_{3,3}$, which are indeed columns r and t. However, it is conceivable that we can easily get the XOR sum of corresponding unknown symbol pair in column s, since horizontal syndromes are available.

To summarize this step, we denote l_h to be the number of rows in a ring, which are canceled through substitution and define the set of corresponding row indices as $F_h = \{h_{l_2} \mid 0 \leq l_2 < l_h\}$. The set F_h is simply obtained by enumerating all crosses of the ring and then counting rows with 3 unknown symbols. Let $\tilde{a}_u$ denote the XOR sum of the unknown symbol pair $a_{0,s}$ and $a_{\langle p-2u \rangle_p, s}$, then the i^{th} pair has

$$\tilde{a}_{u+i} = \bigoplus_{l_1=0}^{l_d-1} \tilde{s}_{\langle -r+i \rangle_p, 2} \bigoplus_{l_2=0}^{l_h-1} \tilde{s}_{\langle h_{l_2}+i \rangle_p, 0}, \bigoplus_{l_1=0}^{l_d-1} \tilde{s}_{\langle t+i \rangle_p, 1} \tag{3}$$

where $0 \leq i \leq p-1$.

4.1.3 Recover Middle Erasure Column

In the previous step, we have computed the XOR sum of arbitrary unknown symbol pair in column s with the fixed distance 3. Since symbol $a_{4,1}$ is an imaginary symbol with zero value, it is straightforward to recover symbol $a_{1,1}$. Next, symbol $a_{3,1}$ can be recovered since the XOR sum of the pair $a_{1,1}$ and $a_{3,1}$ is available. Consequently, symbols $a_{0,1}$ and $a_{2,1}$ are recovered. This process is shown to succeed with arbitrary parameters by Lemma 3.

Lemma 3 *Given the XOR sum of arbitrary symbol pair with a fixed distance d, all symbols in the column are recoverable if there is at least one symbol available.*

Proof. Since p is prime, $F = \{\langle di \rangle_p | \ 0 \leq i \leq p-1\}$ covers all integers in $[0, p)$. Therefore, a "tour" starting from row $p-1$ with the stride size d will visit all other rows exactly once before returning to it. As the symbol in row $p-1$ is always available (zero indeed) and the XOR sum of any pair with distance d is also known, all symbols can then be recovered along the tour.

To summarize, this step computes

$$\tilde{a}_{\langle (p-1)-di \rangle_p} = \tilde{a}_{\langle (p-1)-di \rangle_p} \oplus a_{\langle (p-1)-d(i-1) \rangle_p}, \tag{4}$$

where $0 \leq i \leq p-1$. Then, $a_{i,s} = \tilde{a}_i$ (where there are 2 unknown symbols left in the ring after cancellations) or

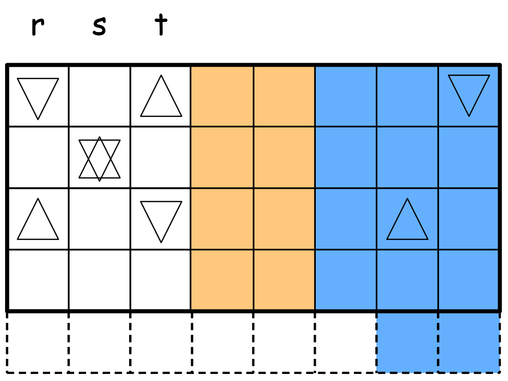

Figure 5: STAR Code Decoding (Symmetric Erasures)

$a_{i,s} = \tilde{a}_i \oplus \tilde{s}_{i,0}$ (where 4 unknown symbols are left) for all i's. Thus far, column s is completely recovered.

4.1.4 Recover Side Erasure Columns

Now that column s is known, the first $p+2$ columns compose an EVENODD coded block with 2 erasures. Thus this reduces to an EVENODD decoding of two erasures.

4.2 Decoding without Parity Erasures: Symmetric Case

When the erasure pattern is symmetric ($u = v$), the decoding becomes much easier, where step 2 is greatly simplified while all other steps remain the same.

To illustrate the step of finding a starting point, we still resort to the previous example, although the erasure pattern is different now. Let's assume $r = 0$, $s = 1$ and $t = 2$, as shown in Figure 5. It is easy to see that only one cross is needed to construct a "ring" (still denoted as a ring, although not closed anymore). As in this example, a cross consists of unknown symbols $a_{0,0}$, $a_{0,2}$, $a_{2,0}$ and $a_{2,2}$, and $a_{1,1}$ is canceled because it is included twice. The XOR sum of the cross thus equals to $\tilde{s}_{2,1} \oplus \tilde{s}_{0,2}$. This is very similar to the situation in the previous case, where there are 4 unknown symbols in a ring after cancellations. Therefore, the rest of the decoding can followed the already described procedure and we don't repeat in here.

In summary the symmetric case can be decoded using the procedure for the asymmetric case, by simply setting $l_d = 1$, $l_h = 0$, $u = 0$ and $d = t - r$.

4.3 Decoding with Parity Erasures

In this part, we consider the situation when there are erasures in parity columns. The decoding is divided into the following 3 subcases.

4.3.1 Column $p+2$ is an erasure

This then reduces to EVENODD decoding of two erasures. Note that this case also takes care of all patterns with fewer than 3 erasures.

4.3.2 Column $p+1$ is an erasure, while $p+2$ is not

This is almost the same as the previous case, except that now the "EVENODD" coded block consists of the first $p+1$ columns and column $p+2$. In fact, this coded block is no longer a normal EVENODD code, but rather a mirror reflection of one over the horizontal axis. Nevertheless, it can be decoded with slightly modification of the EVENODD decoding, which we simply leave to interested readers.

4.3.3 Column p is an erasure, while $p+1$ and $p+2$ are not

In this case, $0 \leq r < s \leq p-1$ and $t = p$.

First, it is not possible to recover adjusters S_1 and S_2, as symbols in column p are unknown. However, $S_1 \oplus S_2$ is still computable, which simply equals to the XOR sum of all symbols in column $p+1$ and $p+2$. This is easy to see from the definitions of S_1 and S_2, S_0 is added twice and canceled out. It is thus possible to reverse the adjuster complement. The results from syndrome calculation are XOR sums of syndromes and their corresponding adjusters, rather than syndromes themselves. We use $\hat{s}_{i,j}$ to denote the results, which thus satisfy

$$\hat{s}_{i,j} = \tilde{s}_{i,j} \oplus S_j, \tag{5}$$

where $j = 1$ or 2 and $0 \leq i \leq p-1$. Note that $\hat{s}_{i,0} = \tilde{s}_{i,0}$ for all i's.

The next step is similar to the decoding of the symmetric case without parity erasures, as it is also true that only one cross is needed to construct a ring. Taking the cross starting with row 0 as an example, it consists of unknown symbols $a_{0,r}$. $a_{0,s}$, $a_{u,r}$ and $a_{u,s}$. Since the XOR sum of this cross equals to $\tilde{s}_{s,1} \oplus \tilde{s}_{\langle -r\rangle_p,2}$, we can easily get the following equation by substituting Eq. 5:

$$a_{0,r} \oplus a_{0,s} \oplus a_{u,r} \oplus a_{u,s} = \hat{s}_{s,1} \oplus \hat{s}_{\langle -r\rangle_p,2} \oplus S_1 \oplus S_2.$$

Therefore, the XOR sum of the cross is computable. Following the approach as used to recover middle erasure column in an early section, the XOR sum of two unknown symbols on any row can be recovered, which is still denoted as $\tilde{a}_i$ $(0 \leq i \leq p-1)$. Then, parity column p can be recovered, as

$$a_{i,p} = \tilde{a}_i \oplus \tilde{s}_{i,0} = \tilde{a}_i \oplus \hat{s}_{i,0},$$

where $0 \leq i \leq p-1$.

After column p is recovered, the first $p+2$ columns can again be regarded as an EVENODD coded block with 2 erasures at column r and s. Therefore, the application of the EVENODD decoding can complete the recovery of all the remaining unknown symbols.

To summarize the procedure in this subcase, we have

$$S_1 \oplus S_2 = \left(\bigoplus_{i=0}^{p-2} a_{i,p+1}\right) \oplus \left(\bigoplus_{i=0}^{p-2} a_{i,p+2}\right),$$

and

$$\hat{s}_{i,0} = a_{i,0} \oplus \left(\bigoplus_{j=0}^{p-1} a_{i,j}\right),$$

$$\hat{s}_{i,1} = a_{i,1} \oplus \left(\bigoplus_{j=0}^{p-1} a_{\langle p+i-j\rangle_p,j}\right),$$

$$\hat{s}_{i,2} = a_{i,2} \oplus \left(\bigoplus_{j=0}^{p-1} a_{\langle i+j\rangle_p,j}\right),$$

where $0 \leq i \leq p-1$ and $j \neq r$ or s. Then,

$$\tilde{a}_i = \hat{s}_{\langle s+i\rangle_p,1} \oplus \hat{s}_{\langle -r+i\rangle_p,2} \oplus S_1 \oplus S_2,$$

where $0 \leq i \leq p-1$, and

$$\tilde{a}_{\langle (p-1)-ui\rangle_p} = \tilde{a}_{\langle (p-1)-ui\rangle_p} \oplus a_{\langle (p-1)-u(i-1)\rangle_p},$$

where $1 \leq i \leq p-1$. Finally, column p can be recovered as

$$a_{i,p} = \tilde{a}_i \oplus \hat{s}_{i,0},$$

for all i's. The rest is to use the EVENODD decoding to recover the remaining 2 columns, which is skipped in here.

Putting all the above cases together, we conclude this section with the following theorem:

Theorem 1 *The STAR code can correct any triple column erasures and thus it is a $(p+3, p)$ MDS code.*

5 Algebraic Representation of the STAR Code

As described in [5], each column in the EVENODD code can be regarded algebraically as an element of a polynomial ring, which is defined with multiplication taken modulo $M_p(x) = (x^p - 1)/(x-1) = 1 + x + \cdots + x^{p-2} + x^{p-1}$. For the ring element x, it is shown that its multiplicative order is p. Using β to denote this element, then column j $(0 \leq j \leq p+1)$ can be represented using the notation $a_j(\beta) = a_{p-2,j}\beta^{p-2} + \cdots + a_{1,j}\beta + a_{0,j}$, where $a_{i,j}$ $(0 \leq i \leq p-2)$ is the i^{th} symbol in the column. Note that the multiplicative inverse of β exists and can be denoted as β^{-1}. Applying same notations to the STAR code, we can then get its parity check matrix as:

$$H = \begin{bmatrix} 1 & 1 & \cdots & 1 & 1 & 0 & 0 \\ 1 & \beta & \cdots & \beta^{p-1} & 0 & 1 & 0 \\ 1 & \beta^{-1} & \cdots & \beta^{-(p-1)} & 0 & 0 & 1 \end{bmatrix} \quad (6)$$

It is not hard to verify that, as in [7], that any 3 columns in the parity check matrix are linearly independent. Therefore, the minimum distance of the STAR code is indeed 4 (each column is regarded as a single element in the ring) and thus arbitrary triple (column) erasures are recoverable. This is an alternative way to show its MDS property.

6 Complexity Analysis

In this section, we analyze the complexity of the STAR code erasure decoding. The complexity is dominated by XOR operations, thus we can count the total number of XORs and use that as an indication of the complexity. Since decoding without parity erasures is the most complicated case, including both asymmetric and symmetric erasure patterns, our analysis is focused on this case.

6.1 Erasure Decoding Complexity

It is not difficult to see that the complexity can be analyzed individually for each of the 4 decoding steps. Note that a complete STAR code consists of p information columns and $r = n - k = 3$ parity columns. When there are only k ($k \leq p$) information columns, we can still use the same code by resorting to the *shortening* technique, which simply assigns zero value to all symbols in the last $p - k$ information columns. Therefore, in the analysis here, we assume the code block is a $(p-1) \times (k+3)$ array.

In step 1, the calculation of S_0 takes $(p-2)$ XOR operations and those of S_1 and S_2 take $(p-1)$ XORs each. The reversion of adjuster complement takes $2(p-1)$ XORs in total. Directly counting XORs of the syndrome calculations is fairly complicated and we can resort to the following alternative approach. First, it is easy to see that the syndrome calculations of any parity direction for a code block without erasures (a $(p-1) \times (p+3)$ array) take $(p-1)p$ XORs. Then, notice that any information column contributes $(p-1)$ XORs to the calculations. Therefore, for a code block with $(k-3)$ information columns (with triple erasures), the number of XORs becomes $(p-1)p - (p-k+3)(p-1) = (k-3)(p-1)$. In total, the XORs in this step is:

$$\begin{aligned} &(p-2) + 2(p-1) + 2(p-1) + 3(k-3)(p-1) \\ =&(3k-4)(p-1) - 1. \end{aligned} \quad (7)$$

In step 2, the computation of each ring takes $(2l_d + l_h - 1)$ XORs and there are $(p-1)$ rings to compute. Thus, the number of XORs is

$$(2l_d + l_h - 1)(p-1). \quad (8)$$

In step 3, it is easy to see that the number of XORs is

$$(p-1) - 1 = p - 2. \quad (9)$$

In step 4, the horizontal and the diagonal syndromes need to be updated with the recovered symbols of column s, which takes $2(p-1)$ XORs. Note that there is no need to update the anti-diagonal syndromes, because the decoding hereafter deals with only double erasures. The Zig-Zag decoding then takes $2(p-1) - 1$ XORs. So the number of XORs in this step is

$$2(p-1) + 2(p-1) - 1 = 4(p-1) - 1. \quad (10)$$

Note that in step 2, the number of XORs is computed assuming the case where only 2 unknown symbols are left in a ring after cancellations. If the other case happens, where 4 unknown symbols are left, additional $(p-1)$ XOR operations are needed to recover column s. However, this case does *not* need to update the horizontal syndromes in step 4 and thus saves $(p-1)$ XORs there. Therefore, it is just a matter of moving XOR operations from step 2 to step 4 and the total number remains the same for both cases.

In summary, the total number of XORs required to decode triple information column erasures can be obtained by putting Eq. (7), (8), (9) and (10) together, as:

$$\begin{aligned} &(3k-4)(p-1) - 1 + (2l_d + l_h - 1)(p-1) \\ &+ (p-2) + 4(p-1) - 1 \\ =\ &(3k + 2l_d + l_h)(p-1) - 3 \quad (11) \\ \approx\ &(3k + 2l_d + l_h)(p-1). \quad (12) \end{aligned}$$

6.2 A Decoding Optimization

From Eq. (12), we can see that for fixed code parameters k and p, the decoding complexity depends on l_d and l_h, which are completely determined by actual erasure patterns (r, s and t). In Sec. 4, we present an algorithm to construct a ring of crosses, which will yield a starting point for successful decoding. Within the ring, all crosses are $v = t - s$ symbols offset from previous ones. From Eq. (2), there are exactly two rows with unknown symbols left after cancellations. From the symmetric property of the ring construction, it is not difficult to show that using offset $u = s - r$ will also achieve the same goal. If using u as offset results in smaller l_d and l_h values (to be specific, smaller $2l_d + l_h$), then there is advantage to do so.

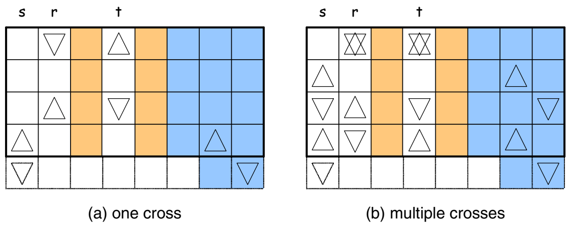

Figure 6: Optimization of STAR Decoding

Moreover, we make the assumption $r < s < t$ during the description of the decoding algorithm. Although it helps to visualize the key procedure of finding a starting point, this assumption is unnecessary. Indeed, it is easy to verify that all proofs in Sec. 4 still hold without this assumption. By swapping values among r, s and t, it might be possible to reduce the decoding complexity. For instance, in the previous example, $r = 0$, $s = 1$ and $t = 3$ results in $l_d = 2$ and $l_h = 2$. If letting $r = 1$, $s = 0$ and $t = 3$, then $u = -1$ and $v = 3$. The pattern of single cross is shown in Figure 6(a). From Figure 6(b), it is clear that two crosses close a ring, which contains exactly two rows (row 1 and 4) with unknown symbols after cancellations. Thus, this choice also yields $l_d = 2$ and $l_h = 2$. However, if letting $r = 0$, $s = 3$ and $t = 1$, we can get $u = s - r = 3$ and $v = t - s = -2$. It is easy to find out that unknown symbols in column s are canceled in every single cross. In fact, this is an equivalence of the symmetric case and in turn $l_d = 1$ and $l_h = 0$. Thus, the complexity is reduced by this choice. Note that for general u and v, the condition of symmetric now becomes $\langle u - v\rangle_p = 0$, instead of simply $u = v$.

Now let us revisit the ring construction algorithm described in Sec. 4. The key point there is to select multiple crosses such that the bottom row of the final cross "steps over" the top row of the first one, and there are exact two rows left with unknown symbols after cancellations. Further examination, however, reveals that it is possible to construct rings using alternative approaches. For instance, the crosses can be selected in such a way that *in the middle column* the bottom symbol of the final cross "steps over" the top symbol of the first one. Or perhaps there is even no need to construct closed rings and crosses might not have to be a fixed offset from previous ones. Indeed, if crosses can be selected arbitrarily while still ensuring exact two rows left with unknown symbols after cancellations, the successful decoding can be guaranteed. Recall that single cross is represented by $C(x) = 1 + x^u + x^v + x^{u+v}$ and a cross of f symbol offset by $C(x)x^f$. Therefore, the construction of a ring is to determine a polynomial term $R(x)$, such that $C(x)R(x)$ results in exact two entries. For instance, the example in Sec. 4 has $R(x) = 1 + x^2$ and $C(x)R(x) = x + x^4$. It is thus possible to further reduce the decoding complexity. Theorem 2 shows that the decoding complexity is minimized if a $R(x)$ with minimum entries is adopted.

Theorem 2 *The decoding complexity is nondecreasing with respect to the number of crosses (l_d) in a ring.*

Proof. Whenever a new cross is included into the ring, two new non-horizontal syndromes (one diagonal and one anti-diagonal) need to be added to the XOR sum. With this new cross, at most four rows can be canceled (simple cancellation due to even times addition), among which two can be mapped with this cross and the other two with an earlier cross. Thus, each cross adds two non-horizontal syndromes but subtracts at most two horizontal syndromes. The complexity is thus nondecreasing with respect to the number of crosses.

Note that l_d is in fact the number of entries in $R(x)$. An optimal ring needs to find a $R(x)$ with minimum entries, which then ensures that $C(x)R(x)$ has only two terms. An efficient approach to achieve this is to test all polynomials with two terms. If a polynomial is divisible by $C(x)$, then the quotient yields a valid $R(x)$. A $R(x)$ with minimum entries is then chosen to construct the ring. It is important to point out that there is no need to worry about common factors (always powers of x) between two terms in the polynomial, as it is not divisible by $C(x)$. Thus, the first entry of all polynomials can be fixed as 1, which means that only $p - 1$ polynomials ($1 + x^i$, $0 < i \le p - 1$) need to be examined. As stated in an earlier section, polynomials are essentially elements in the ring constructed with $M_p(x) = 1 + x + \cdots + x^{p-2} + x^{p-1}$. Based on the argument in [8], $(1 + x^u)$ and $(1 + x^v)$ are invertible in the ring. Thus, $C(x) = (1 + x^u)(1 + x^v)$ is also invertible, and it is straightforward to compute the inverse using Euclid's algorithm. For instance, $C(x) = 1 + x + x^2 + x^3$, as $u = 1$ and $v = 2$ in the previous example. The generator polynomial $M_p(x) = 1 + x + x^2 + x^3 + x^4$ as $p = 5$. Applying the Euclid's algorithm [26], it is clear that

$$1(\mathbf{1 + x + x^2 + x^3 + x^4}) + x(\mathbf{1 + x + x^2 + x^3}) = 1. \tag{13}$$

Thus, the inverse of $C(x)$ is $inv(C(x)) = x$. When examining the polynomial $1 + x^3$, we get $R(x) = inv(C(x))(1 + x^3) = x + x^4$ or equivalently,

$$(1 + x + x^2 + x^3)(x + x^4) = 1 + x^3 \bmod M_p(x). \tag{14}$$

It is desirable that $R(x)$ carries the entry of power 0, since the ring always contains the original cross. So we multiply x to both sides of Eq. (14), which now becomes

$$(1 + x + x^2 + x^3)(1 + x^2) = x + x^4 \bmod M_p(x).$$

Thus, we have $R(x) = 1 + x^2$ and the ring can be constructed using two crosses ($l_d = 2$) with an offset of two symbols. Once the ring is constructed, it is straightforward to get l_h.

Note this optimal ring construction only needs to be computed once in advance (offline). Thus we do not count the ring construction in the decoding procedure.

7 Comparison with Existing Schemes

In this section, we compare the erasure decoding complexity of the STAR code to two other XOR-based codes, one proposed by Blaum *et al.* [7] (Blaum code hereafter) and the other by Blomer *et al.* [10].

The Blaum code is a generalization of the EVENODD code, whose horizontal (the 1^{st}) and diagonal (the 2^{nd}) parities are now regarded as redundancies of slope 0 and 1, respectively. A redundancy of slope $q-1$ ($q \geq 3$) generates the q^{th} parity column. This construction is shown to maintain the MDS property for triple parity columns, when the code parameter p is a prime number. The MDS property continues to hold for selected p values when the number of parities exceeds 3. To make the comparison meaningful, we focus on the triple parity case of the Blaum code. We compare the complexity of triple erasure decoding in terms of XOR operations between the Blaum code and the STAR code. As in the previous sections, we confine all three erasures to information columns.

The erasure decoding of the Blaum code adopts an algorithm described in [8], which provides a general technique to solve a set of linear equations in a polynomial ring. Due to special properties of the code, however, ring operations are *not* required during the decoding procedure, which can be performed with pure XOR and shift operations. The algorithm consists of 4 steps, whose complexities are summarized as follows: 1) syndrome calculation: $3(k-3)(p-1)-1$; 2) computation of $\hat{Q}(x;z)$: $\frac{1}{2}r(3r-3)p$; 3) computation of the right-hand value: $r((r-1)p+(p-1))$; and 4) extracting the erasure values: $r(r-1)(2(p-1))$. Here $r = n-k = 3$. Therefore, the total number of XORs is

$$\begin{aligned} & 3(k-3)(p-1)-1+9p+(9p-3)+12(p-1) \\ = & (3k+21)(p-1)+14 \qquad (15) \\ \approx & (3k+21)(p-1). \qquad (16) \end{aligned}$$

Comparison results with the STAR code are shown in Figure 7, where we can see that the complexity of the STAR decoding remains fairly constant and is just slightly above 3. Note that this complexity depends on actual erasure locations, thus the results reported here are average values over all possible erasure patterns. The complexity of the Blaum code, however, is rather high for small k values, although it *does* approach 3 asymptotically. The STAR code is thus probably more desirable than the Blaum code. Figure 7 also includes the

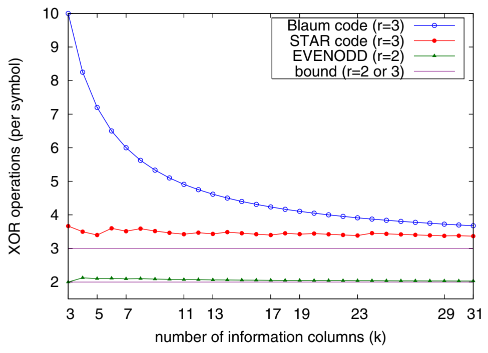

Figure 7: The Complexity Comparisons ($r = n - k$)

complexity of the EVENODD decoding as a reference, which is roughly constant and slightly above 2 XORs per symbol. Note in Figure 7, p is always taken for each k as the next largest prime.

Further reflection on the Blaum code and the STAR code would reveal that the construction difference between them lies solely on the choice of the 3^{rd} redundancy slope, where the Blaum code uses slope 2 and the STAR code -1. One might wonder whether the decoding approach adopted here could be applied to the Blaum code as well. Based on STAR decoding's *heavy* reliance on the geometric property of individual crosses in the step to find a starting point, it seems difficult to achieve the same ring construction in the Blaum code when symmetry is no longer obvious. Moreover, the intuitiveness of the decoding process would be completely lost even if it is possible at all. Instead, we would be more interested to investigate whether the STAR code construction, so as the decoding approach, could be extended to handle more than triple erasures, as the Blaum code already does.

The XOR-based code proposed in [10] uses Cauchy matrices to construct a Reed-Solomon (RS) code. It replaces generator matrix entries, information and parity symbols with binary representations. Then, the encoding and decoding can be performed with primarily XOR operations. To achieve maximum efficiency, it requires message length to be multiples of 32 bits. In that way, basic XOR unit is 32 bits, or single word, and can be performed by single operation. To compare with this scheme fairly, we require the symbol size of the STAR code to be multiples of 32 bits too. It is shown that the XOR-based decoding algorithm in [10] involves krL^2 XOR operations and r^2 operations in a finite field $GF(2^L)$, where k and r are the numbers of information symbols and erasures, respectively. We ignore those r^2 finite field operations (due to the inversion of a decoding coefficient matrix), which tend to be small as the number of erasures is limited. Then, the RS code's nor-

# of total columns (n)	# of XORs ($= rL$)	
	$r = 2$	$r = 3$
$n \leq 8$	6	9
$9 < n \leq 16$	8	12
$17 < n \leq 32$	10	15
$33 < n \leq 64$	12	18

Table 1: Complexity of the RS Code (per 32 bits)

malized decoding complexity (by the total information length of kL words) is rL. As the total number of symbols n ($= k + r$) is limited by L ($n \leq 2^L$), we have to increase L and thus in turn the decoding complexity when n increases (see Table 1). Compared to Figure 7, where the STAR code decoding complexity is slightly more than 3 XORs per symbol (multiples of 32 bits now), it is clear that the STAR code is much more efficient than the XOR-based RS code. Note that the complexity of *normal* (finite field-based) RS code implementation (e.g. [30]) turns out to be even higher than the XOR-based one, so we simply skip comparison here.

8 Implementation and Performance

The implementation of the STAR code encoding is straightforward, which simply follows the procedure described in Sec. 3. Thus, in this part, our main focus is on the erasure decoding procedure. As stated in Sec. 6, the decoding complexity is solely determined by l_d and l_h, given the number of information columns k and the code parameter p. As l_d and l_h vary according to actual erasure patterns, so does the decoding complexity. To achieve the maximum efficiency, we apply the optimization technique as described in the earlier section.

An erasure pattern is completely determined by the erasure columns r, s and t (again assume $r < s < t$), or further by the distances u and v between these columns, as the actual position of r does *not* affect l_d or l_h. Therefore, it is possible to set up a mapping from (u,v) to (l_d,l_h). To be specific, given u and v, the mapping returns the positions of horizontal, diagonal and anti-diagonal syndromes, which would otherwise be obtained via ring constructions. The mapping can be implemented as a lookup table and the syndrome positions using bit vectors. Since the lookup table can be built in advance of actual decoding procedure, it essentially shifts complexity from online decoding to offline preprocess. Note that the table lookup operation is only needed once for every erasure pattern, thus there is no need to keep the table in memory (or cache). This is different from finite field based coding procedures, where intensive table lookups are used to replace complicated finite field operations. For example, a RS code implementation might use an

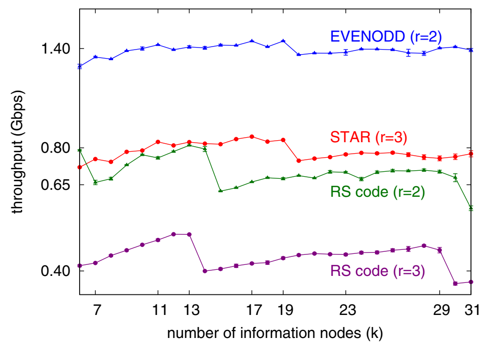

Figure 8: Throughput Performance. ($r = n - k$ erasures are randomly generated among information nodes.)

exponential table and a logarithm table for *each* multiplication/division. Furthermore, the number of entries in the lookup table is not large at all. For example, for code parameter $p = 31$, u and v are at most 30, which requires a table of at most $30 \times 30 = 900$ entries, where each entry contains 3 bit vectors (32-bit each) for the ring construction, one byte for the decoding pattern and another byte for l_h. The cost of maintaining a few tables of this size is then negligible.

During the decoding procedure, u and v are calculated from the actual erasure pattern. Based on these values, the lookup table returns all syndrome positions, which essentially indicates the ring construction. The calculation of the ring is thus performed as the XOR sums of all the indicated syndromes. Then, the next ring is calculated by offsetting all syndromes with one symbol and the procedure continues until all rings are computed. Steps afterward are to recover the middle column and then the side columns, as detailed in Sec. 4.

We implement the STAR code erasure decoding procedure and apply to reliable storage systems. The throughput performance is measured and compared to the publicly available implementation of the XOR-based RS code [11]. The results are shown in Figure 8, where the size of a single data block from each node is 2880 bytes and the number of information storage nodes (k) varies from 6 to 31. Note our focus is on decoding erasures that all occur at information columns, since otherwise the STAR code just reduces to the EVENODD code (when there is one parity column erasure) or a single parity code (when there are two parity column erasures), so we only simulate random information column erasures in Figure 8. Recall that a single data block from each node corresponds to a single column in the STAR code and is divided into $p - 1$ symbols, so the block size needs to be a multiple of $p - 1$. For comparison purpose, we use 2880 here since it is a common multiple of $p-1$ for most p values in the range. In real applications, we are free to choose the block size to be any multiple of $p-1$, once p, as a system parameter, is determined. These results are obtained from experiments on a P3 450MHz Linux machine with 128M memory running Redhat 7.1. It is clear that the STAR code achieves about twice throughput compared to the RS code. Note that there are jigsaw effects in the throughputs of both the EVENODD and the STAR code. This happens mainly due to the shortening technique. When the number of storage nodes is not prime, the codes are constructed using the closest larger prime number. A larger prime number means each column (data block here) is divided into more pieces, which in turn incurs additional control overhead. As the number of information nodes increases, the overhead is then amortized, reflected by the performance ramping up after each dip. (Similarly, the performance of the RS code shows jigsaw effects too, which happens at the change of L due to the increment of total storage nodes n.) Moreover, note that the throughputs are not directly comparable between $r\ (= n-k) = 2$ and $r\ (= n-k) = 3$ (e.g. the EVENODD and the STAR code), as they correspond to different reliability degrees. The results of codes with $r = 2$ are depicted only for reference purpose. Finally, note that necessary correction of the generator matrix (similar to the one documented in [31]) needs to be done in the aforementioned implementation of the XOR-based RS code to ensure the MDS property. This doesn't affect the throughput performance though.

9 Conclusions

In this paper, we describe the STAR code, a new coding scheme that can correct triple erasures. The STAR code extends from the EVENODD code, and requires only XOR operations in its encoding and decoding operations. We prove that the STAR code is an MDS code of distance 4, and thus is optimal in terms of erasure correction capability vs. data redundancy. Detailed analysis shows the STAR code has the lowest decoding complexity among the existing comparable codes. We hence believe the STAR code is very suitable for achieving high availability in practical data storage systems.

Acknowledgments

The authors wish to thank anonymous reviewers for their very insightful and valuable comments and suggestions, which certainly help improve the quality of this paper. This work was in part supported by NSF grants CNS-0322615 and IIS-0430224.

References

[1] G. A. Alvarez, W. A. Burkhard, and F. Christian, "Tolerating multiple failures in RAID architectures with optimal storage and uniform declustering," *Proc. of the 24th Annual Symposium on Computer Architecture* pgs. 62-72, 1997.

[2] T. E. Anderson, D.E. Culler and D.A. Patterson, "A Case for NOW (Networks of Workstations)," *IEEE Micro*, 15(1), 54–64, 1995.

[3] T. Anderson, M. Dahlin, J. Neefe, D. Patterson, D. Roselli and R. Wang, "Serverless Network File Systems", *ACM Trans. on Computer Systems*, 41-79, Feb. 1996.

[4] A. Bhide, E. Elnozahy and S. Morgan, "A Highly Available Network File Server", *Proc. of the Winter 1991 USENIX Technical Conf.*, 199-205, Jan. 1991.

[5] M. Blaum, J. Brady, J. Bruck and J. Menon, "EVENODD: An Efficient Scheme for Tolerating Double Disk Failures in RAID Architectures," *IEEE Trans. on Computers*, 44(2), 192-202, Feb. 1995.

[6] M. Blaum, J. Brady, J. Bruck, J. Menon, and A. Vardy, "The EVENODD code and its generalization," in *High Performance Mass Storage and Parallel I/O*, pp. 187–208. John Wiley & Sons, INC., 2002.

[7] M. Blaum, J. Bruck, and A. Vardy, "MDS array codes with independent parity symbols," *IEEE Trans. Information Theory*, vol. 42, no. 2, pp. 529–542, Mar. 1996.

[8] M. Blaum, R. M. Roth, "New Array Codes for Multiple Phased Burst Correction," *IEEE Trans. on Information Theory*, 39(1), 66-77, Jan. 1993.

[9] M. Blaum, R. M. Roth, "On Lowest-Density MDS Codes," *IEEE Trans. on Information Theory*, 45(1), 46-59, Jan. 1999.

[10] J. Blomer, M. Kalfane, R. Karp, M. Karpinski, M. Luby, and D. Zuckerman, "An XOR-based erasure-resilient coding scheme," Technical Report No. TR-95-048, ICSI, Berkeley, California, Aug. 1995.

[11] J. Blomer, M. Kalfane, R. Karp, M. Karpinski, M. Luby, and D. Zuckerman, *http://www.icsi.berkeley.edu/∼luby/-cauchy.tar.uu*

[12] V. Bohossian, C. Fan, P. LeMahieu, M. Riedel, L. Xu and J. Bruck, "Computing in the RAIN: A Reliable Array of Independent Node", *IEEE Trans. on Parallel and Distributed Systems*, Special Issue on Dependable Network Computing, 12(2), 99-114, Feb. 2001.

[13] M. Castro and B. Liskov, "Practical Byzantine Fault Tolerance", *Operating Systems Review*, ACM Press, NY, 173-186, 1999.

[14] P. M. Chen, E. K. Lee, G. A. Gibson, R. H. Katz, D. A. Patterson, "Raid – High-Performance, Reliable Secondary Storage," *ACM Computing Surveys*, 26(2), 145–185, 1994.

[15] P. Corbett, B. English, A. Goel, T. Grcanac, S. Kleiman, J. Leong and S. Sankar, "Row-Diagonal Parity for Double Disk Failure Correction", *Proc. of USENIX FAST 2004*, Mar. 31 to Apr. 2, San Francisco,CA, USA.

[16] C. Fan and J. Bruck, "The Raincore API for Clusters of Networking Elements", *IEEE Internet Computing*, 5(5), 70-76, Sep./Oct., 2001.

[17] P. G. Farrell, "A Survey of Array Error Control Codes," *ETT* , 3(5), 441-454, 1992.

[18] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung "The Google File System", Proc. of 19th ACM Symposium on Operating Systems Principles, Lake George, NY, October, 2003, pp. 29 - 43

[19] G. A. Gibson and R. van Meter, "Network Attached Storage Architecture", *Communications of the ACM*, 43(11), 37-45, Nov. 2000.

[20] G. A. Gibson, D. Stodolsky, F. W. Chang, W. V. Courtright II, C. G. Demetriou, E. Ginting, M. Holland, Q. Ma, L. Neal, R. H. Patterson, J. Su, R. Youssef and J. Zelenka, "The Scotch Parallel Storage Systems," *Proceedings of the IEEE CompCon Conference*, 1995.

[21] A. V. Goldberg and P. N. Yianilos, "Towards an Archival Intermemory", *Proc. of IEEE Advances in Digital Libraries*, Apr. 1998.

[22] R. M. Goodman, R. J. McEliece and M. Sayano, "Phased Burst Error Correcting Arrays Codes," *IEEE Trans. on Information Theory*, 39, 684-693,1993.

[23] J. H. Hartman and J. K. Ousterhout, "The Zebra Striped Network File System," *ACM Transactions on Computer Systems*, 13(3), 274–310, 1995.

[24] J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells and B. Zhao, "OceanStore: An Architecture for Global-Scale Persistent Storage", *Proc. of the Ninth international Conference on Architectural Support for Programming Languages and Operating Systems*, Nov. 2000.

[25] E. Lee and C. Thekkath, "Petal: Distributed Virtual Disks", *Proc. ACM ASPLOS*, 84-92, Oct. 1996.

[26] F. J. MacWilliams and N. J. A. Sloane, *The Theory of Error Correcting Codes*, Amsterdam: North-Holland, 1977.

[27] R. J. McEliece, D. Sarwate, "On sharing secrets and Reed-Solomon codes", *Comm. ACM*, 24(9), 583-584, 1981.

[28] J. Ousterhout, A. Cherenson, F. Douglis, M. Nelson and B. Welch, "The Sprite Network Operating System", *IEEE Computer*, 21(2): 23-26, Feb. 1988.

[29] Chong-Won Park and Jin-Won Park, "A multiple disk failure recovery scheme in RAID systems," *Journal of Systems Architecture*, vol. 50, pp. 169–175, 2004.

[30] J. S. Plank, "A tutorial on Reed-Solomon coding for fault-tolerance in RAID-like systems," *Software: Practice and Experience*, vol. 27, no. 9, pp. 995–1012, Jan. 1999.

[31] J. S. Plank and Y. Ding "Note: Correction to the 1997 Tutorial on Reed-Solomon Coding" *Software, Practice & Experience*, vol. 35, no. 2, pp. 189–194, Feb. 2005.

[32] J. S. Plank, R. L. Collins, A. L. Buchsbaum and M. G. Thomason, "Small Parity-Check Erasure Codes - Exploration and Observations," *International Conference on Dependable Systems and Networks (DSN)*, Yokohama, Japan, Jun. 2005.

[33] J. S. Plank, M. and T. Moore, "Logistical Networking Research and the Network Storage Stack," *USENIX FAST 2002, Conference on File and Storage Technologies*, work in progress report, January, 2002.

[34] M. Rabin, "Efficient Dispersal of Information for Security, Load Balancing and Fault Tolerance", *J. ACM*, 32(4), 335-348, Apr. 1989.

[35] I. S. Reed and G. Solomon, "Polynomial Codes over Certain Finite Fields", *J. SIAM*, 8(10), 300-304, 1960.

[36] M. Satyanarayanan, "Scalable, Secure and Highly Available Distributed File Access", *IEEE Computer*, 9-21, May 1990.

[37] M. Satyanarayanan, J.J. Kistler, P. Kumar, M. E. Okasaki, E. H. Siegel and D. C. Steere, "CODA - A Highly Available File System for a Distributed Workstation Environment," *IEEE Transactions on Computers*, 39(4), 447–459, 1990.

[38] A. Shamir, "How to Share a Secret", *Comm. ACM*, 612-613, Nov. 1979.

[39] SUN Microsystems, Inc. *NFS: Network File System version 3 Protocol Specification* , Feb. 1994.

[40] Chih-Shing Tau and Tzone-I Wang, "Efficient parity placement schemes for tolerating triple disk failures in RAID architectures," in *Proceedings of the17 th International Conference on Advanced Information Networking and Applications (AINA'03)*, Xi'an, China, mar 2003.

[41] M. Waldman, A. D. Rubin and L. F. Cranor, "Publius: A robust, tamper-evident, censorship-resistant, web publishing system", *Proc. 9th USENIX Security Symposium*, 59-72, Aug. 2000. Online at: *http://www.cs.nyu.edu/∼waldman/publius/publius.pdf*

[42] J. J. Wylie, M. W. Bigrigg, J. D. Strunk. G. R. Ganger, H. Kiliccote and P. K. Khosla, "Survivable Information Storage Systems", *IEEE Computer*, 33(8), 61-68, Aug. 2000.

[43] L. Xu and J. Bruck, "X-Code: MDS Array Codes with Optimal Encoding," *IEEE Trans. on Information Theory*, 45(1), 272-276, Jan., 1999.

[44] L. Xu, V. Bohossian, J. Bruck and D. Wagner, "Low Density MDS Codes and Factors of Complete Graphs," *IEEE Trans. on Information Theory*, 45(1), 1817-1826, Nov. 1999.