# GRID Codes: Strip-Based Erasure Codes with High Fault Tolerance for Storage Systems

MINGQIANG LI, JIWU SHU, and WEIMIN ZHENG
Tsinghua University

As storage systems grow in size and complexity, they are increasingly confronted with concurrent disk failures together with multiple unrecoverable sector errors. To ensure high data reliability and availability, erasure codes with high fault tolerance are required. In this article, we present a new family of erasure codes with high fault tolerance, named GRID codes. They are called such because they are a family of *strip-based codes* whose strips are arranged into multi-dimensional grids. In the construction of GRID codes, we first introduce a concept of *matched codes* and then discuss how to use matched codes to construct GRID codes. In addition, we propose an iterative reconstruction algorithm for GRID codes. We also discuss some important features of GRID codes. Finally, we compare GRID codes with several categories of existing codes. Our comparisons show that for large-scale storage systems, our GRID codes have attractive advantages over many existing erasure codes: (a) They are completely XOR-based and have very regular structures, ensuring easy implementation; (b) they can provide up to 15 and even higher fault tolerance; and (c) their storage efficiency can reach up to 80% and even higher. All the advantages make GRID codes more suitable for large-scale storage systems.

Categories and Subject Descriptors: B.8.1 [**Performance and Reliability**]: Reliability, Testing, and Fault-Tolerance; D.4.2 [**Operating Systems**]: Storage Management—*Secondary storage*; H.1.1 [**Models and Principles**]: Systems and Information Theory—*Information theory*; H.3.2 [**Information Storage and Retrieval**]: Information Storage

General Terms: Algorithms, Reliability, Theory

Additional Key Words and Phrases: Disk failure, erasure code, fault tolerance, storage system, unrecoverable sector error

**ACM Reference Format:**

Li, M., Shu, J., and Zheng, W. 2009. GRID codes: Strip-based erasure codes with high fault tolerance for storage systems. ACM Trans. Storage, 4, 4, Article 15 (January 2009), 22 pages. DOI = 10.1145/1480439.1480444 http://doi.acm.org/10.1145/1480439.1480444



This work was supported by the National Natural Science Foundation of China under Grant Nos. 60873066 and 60473101, the National Grand Fundamental Research 973 Program of China under Grant Nos. 2004CB318205 and 2007CB311100, and the Program for New Century Excellent Talents in University (NCET-05-0067).

Authors' addresses: M. Li, J. Shu, and W. Zheng, Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China; email: lmq06@mails.tsinghua.edu.cn; {shujw,zwmdcs}@tsinghua.edu.cn.

## 1. INTRODUCTION

One of the biggest challenges in designing storage systems is how to provide the reliability and availability that are expected by system users. However, as storage systems grow in size and complexity, they are increasingly confronted with concurrent disk failures [Eduardo Pinheiro and Barroso 2007; Schroeder and Gibson 2007] together with multiple unrecoverable sector errors [Bairavasundaram and Schindler 2007]. To ensure high data reliability and availability, highly fault-tolerant technologies are required. Furthermore, as mentioned in Hafner [2005], as the industry moves into very long-term archival storage systems or dRAID (distributed Redundant Arrangement of Independent Devices) node-based systems, the need for highly fault-tolerant technologies will become more urgent.

There are two kinds of highly fault-tolerant technologies: $n$-way mirroring technologies and erasure-coding technologies. $N$-way mirroring technologies can provide sufficient reliability, but have very low storage efficiency (only $1/n$). In contrast, erasure-coding technologies can provide both high fault tolerance and high storage efficiency. Thus, erasure-coding technologies have become very popular with the designers of storage systems and have been used in many practical systems [Chen and Patterson 1994; Frølund and Veitch 2004; Goodson and Reiter 2004; Wilcke and Loo 2006; Collins and Plank 2005; Aguilera and Xu 2005; Xia and Chien 2007].

Under this background, many erasure codes have been proposed in recent years. Referring to Plank [2005], we divide them into four categories.

—*Reed-Solomon codes*. Reed-Solomon codes [Reed and Solomon 1960; Roth and Lempel 1989] provide arbitrarily high fault tolerance and optimal storage efficiency (because they are Maximum Distance Separable, or MDS, codes), but require complex Galois field arithmetic. Even if formulated as parity array codes [Blomer and Zuckerman 1995; Plank and Xu 2006], they still have relatively low storage performance [Plank 2008].

—*Parity array codes*. Some of these (like EVENODD [Blaum and Menon 1995], Row-Diagonal Parity (RDP) codes [Corbett and Sankar 2004], STAR [Huang and Xu 2005], Liberation codes [Plank 2008], and X-code [Xu and Bruck 1999]) are completely XOR-based MDS codes, but have low fault tolerance (not higher than 3). Others can provide high fault tolerance, but still have disadvantages: (i) WEAVER codes [Hafner 2005] are non-MDS codes and have very low storage efficiency (not higher than 50%); and (ii) the generalizations of EVENODD [Blaum and Vardy 2001, 1996] and Feng's codes [Feng and Shen 2005a, 2005b] are based on ring theory and thus require not only XOR operations but also cyclic shift operations.

—*Parity check codes*. Parity check codes [Rubinoff 1961; Wong and Shea 2001; Anne and Latifi 2004] are completely based on XOR operations, but have relatively low storage efficiency.

—*LDPC codes*. Low-density Parity Check (LDPC) codes [Gallager 1962; Luby and Spielman 2001; Plank and Thomason 2004; Plank and Thomason 2005]

are completely XOR-based codes and can provide high fault tolerance and near-optimal storage efficiency, but their structures are too irregular to implement efficiently in storage systems.

Although there are so many erasure codes proposed for storage systems, none of them has become a de facto standard in the storage industry, and we believe that "no such 'perfect' code can exist" [Hafner 2005].

In this article, we present a new family of erasure codes with high fault tolerance, named GRID codes. They are called such because they are a family of *strip-based codes* whose strips are arranged into multi-dimensional grids. They evolve from parity check codes and are constructed based on single parity check (SPC) codes and completely XOR-based MDS parity array codes.

In the construction of GRID codes, we first introduce a concept of *matched codes*. Matched codes are a group of codes that are chosen for constructing a GRID code. We then discuss how to choose matched codes. Meanwhile, we discuss how to construct a GRID code from a group of chosen matched codes. In addition, we propose an iterative reconstruction algorithm for GRID codes.

After introducing GRID codes, we discuss some of their important features. We find that GRID codes possess the advantages of both parity check codes and parity array codes.

Finally, we compare GRID codes with several categories of existing codes. Our comparisons show that for large-scale storage systems, our GRID codes have attractive advantages over many existing erasure codes.

—They are completely XOR-based and have very regular structures, ensuring easy implementation.

—They can provide up to 15 and even higher fault tolerance.

—Their storage efficiencies can reach up to 80% and even higher.

All these advantages make GRID codes more suitable for large-scale storage systems.

This article is organized as follows. We first list some definitions and notations in the next section. Section 3 introduces some related work. We describe our GRID codes in Section 4. Section 5 discusses some important features of GRID codes. Comparisons between GRID codes and other existing codes are made in Section 6. We conclude this article in Section 7.

## 2. DEFINITIONS AND NOTATIONS

Since there are some inconsistencies concerning the use of common terms on storage and erasure code, we state our definitions here to avoid confusion. Referring to Hafner and Roa [2004] and Hafner [2006, 2005], we list some definitions and notations that will be used throughout the article.

—*Data*: a chunk of bytes or blocks that hold unmodified user data;

—*Parity*: a chunk of bytes or blocks that hold redundant information generated from user data by an erasure code;

—*Element*: a fundamental unit of data or parity that can be a bit, a byte, a sector, or a larger disk block;

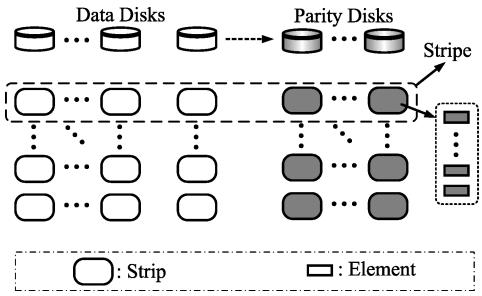



Fig. 1. Element, strip, and stripe in an horizontal code.

—*Stripe*: a maximal set of data and parity elements that are dependently related by an erasure code; the stripe size is defined as the number of disks that hold a stripe and is denoted by $n$.

—*Strip*: a maximal set of elements in a stripe that are stored on the same disk; the strip size is defined as the number of elements contained in a strip and is denoted by $s$.

—*Horizontal codes*: the family of erasure codes in which data and parity elements within a stripe are stored in separate strips (Figure 1 shows a representation of our notions of element, strip, and stripe in a typical horizontal code.);

—*Vertical codes*: the family of erasure codes in which each strip within a stripe contains both data and parity elements;

—*Distance*: the minimum number of erased strips that can result in user data loss; denoted by $d$.

—*Fault tolerance*: the maximum number of erased strips that an erasure code promises to be able to reconstruct; denoted by $t$. It is clear that $t = d - 1$.

—*Storage efficiency*: the ratio of user data to the total of user data plus redundancy data; denoted by $E$.

—*Maximum Distance Separable (MDS) codes*: the family of erasure codes that attain the Singleton bound [MacWilliams and Sloane 1977] and thus provide optimal storage efficiency;

—*Non-MDS codes*: the family of erasure codes that do not attain the Singleton bound;

—*Data out-degree*: the number of parity elements to which a data element contributes;

—*Parity in-degree*: the number of data elements that are involved in computing a parity element;

—*Write lock zone*: the set of data and parity elements within a stripe that should be locked during a write operation so as to ensure parity consistency in case of failures;

—*Reconstruction zone*: the set of strips within a stripe that are involved in a reconstruction operation.

## 3. RELATED WORK

Before presenting our GRID codes, we introduce some related work in this section. Many erasure codes have been proposed in recent years. Referring to Plank [2005], we divide them into four categories.

*Reed-Solomon codes*. There are two types of Reed-Solomon codes: standard Reed-Solomon codes (based on Vandermonde matrices) [Reed and Solomon 1960], and Cauchy Reed-Solomon codes (based on Cauchy matrices) [Roth and Lempel 1989], both of which are horizontal codes. Data on each device is broken up into $w$-bit strips, and the $i$th strip on each parity device is calculated from the $i$th strip on each data device using Galois field arithmetic ($GF(2^w)$). They provide arbitrarily high fault tolerance and optimal storage efficiency (because they are MDS codes), but require complex Galois field arithmetic. Recently, in order to reduce the overheads of encoding and decoding processes, the binary matrix coding technology has been employed in Cauchy Reed-Solomon codes to convert them to parity array codes that work on groups of $w$ bits [Blomer and Zuckerman 1995; Plank and Xu 2006]. Even so, they still have relatively low storage performance [Plank 2008].

*Parity array codes*. They come in three types: (i) horizontal codes, such as EVENODD [Blaum and Menon 1995] (and its generalizations Blaum and Vardy [2001, 1996]), row-diagonal parity (RDP) codes [Corbett and Sankar 2004], STAR [Huang and Xu 2005], Feng's codes [Feng and Shen 2005a, 2005b] and Liberation codes [Plank 2008]; (ii) vertical codes, such as X-code [Xu and Bruck 1999] and WEAVER codes [Hafner 2005]; and (iii) horizontal and vertical codes (the hybrids of horizontal codes and vertical codes), such as HoVer codes [Hafner 2006]. Some of them (like EVENODD, RDP codes, STAR, Liberation codes, and X-codes) are MDS codes and are completely based on simple XOR operations, but have low fault tolerance (not higher than 3). Others can provide high fault tolerance, but still have disadvantages: (i) WEAVER codes are non-MDS codes and have very low storage efficiency (not higher than 50%); and (ii) the generalizations of EVENODD and Feng's codes are based on ring theory and thus require not only XOR operations but also cyclic shift operations.

*Parity check codes*. Parity check codes are constructed based on single parity check (SPC) codes. One typical representative of them is the widely used two-dimensional one, namely horizontal and vertical parity check (HVPC) code. Recently, several higher-dimensional ones have also been proposed in Rubinoff [1961], Wong and Shea [2001], and Anne and Latifi [2004]. Parity check codes are completely based on XOR operations, but have relatively low storage efficiency.

*LDPC codes*. Low-density parity check (LDPC) codes [Gallager 1962; Luby and Spielman 2001] are one-dimensional XOR-based codes and are constructed based on the Tanner graph [Tanner 1981]. They were originally designed for communication channels, but have recently been studied in the context of storage applications over wide-area networks Plank and Thomason [2005, 2004]. In these applications, random packet loss (or delay) is the dominant erasure

| | | | | | |
|---|---|---|---|---|---|
| $D_{0,0}$ | $D_{0,1}$ | $\cdots$ | $D_{0,k_2-1}$ | $\rightarrow$ | $P_{0,*}$ |
| $D_{1,0}$ | $D_{1,1}$ | $\cdots$ | $D_{1,k_2-1}$ | $\rightarrow$ | $P_{1,*}$ |
| $\vdots$ | $\vdots$ | $\ddots$ | $\vdots$ | $\rightarrow$ | $\vdots$ |
| $D_{k_1-1,0}$ | $D_{k_1-1,1}$ | $\cdots$ | $D_{k_1-1,k_2-1}$ | $\rightarrow$ | $P_{k_1-1,*}$ |
| $\downarrow$ | $\downarrow$ | $\cdots$ | $\downarrow$ | $\searrow$ | |
| $P_{*,0}$ | $P_{*,1}$ | $\cdots$ | $P_{*,k_2-1}$ | | $P_{*,*}$ |

Fig. 2. An example of horizontal and vertical parity check (HVPC) codes.

model (rather than total device failure). Good LDPC codes with high fault tolerance and near-optimal storage efficiency often have highly irregular structures, which are not well suited to storage systems.

Although there are so many erasure codes proposed for storage systems, none of them has become a de facto standard in the storage industry because every code is designed by making some trade-offs among fault tolerance, performance, and storage efficiency. As mentioned in Hafner [2005], "No such 'perfect' code can exist."

## 4. GRID CODE DESCRIPTION

In this section, we will describe our GRID codes. Since the GRID codes evolve from parity check codes and are constructed based on single parity check (SPC) codes and completely XOR-based MDS parity array codes, before giving the definition of GRID codes, we first briefly introduce parity check codes and completely XOR-based MDS parity array codes.

### 4.1 A Brief Introduction of Parity Check Codes and Completely XOR-Based MDS Parity Array Codes

4.1.1 *Parity Check Codes.* Parity check codes [Rubinoff 1961; Wong and Shea 2001; Anne and Latifi 2004] are constructed based on single parity check (SPC) codes. Before introducing parity check codes, we first describe SPC codes.

In the scheme of SPC codes, we append a parity strip (denoted by $P$) at the end of data strips (denoted by $D_0$, $D_1, \ldots,$ and $D_{k-1}$, where $k \geq 2$), and the parity strip is calculated by the expression $P = D_0 \oplus D_1 \oplus \cdots \oplus D_{k-1}$. In SPC codes, there is no restriction on their stripe size and strip size. SPC codes can always tolerate one fault.

We now introduce parity check codes. Parity check codes are constructed based on single parity check (SPC) codes. One typical representative of them is the widely used two-dimensional one, namely horizontal and vertical parity check (HVPC) code. We will introduce HVPC codes as follows.

HVPC codes arrange data strips into a two-dimensional array. Figure 2 gives an example of HVPC codes. In this figure, there are $k_1 \times k_2$ data strips that form a array $(D_{i,j})_{k_1 \times k_2}$, where $k_1 \geq 2$ and $k_2 \geq 2$. The row parity strips and column parity strips are calculated by the expressions

$$\begin{cases} P_{i,*} = D_{i,0} \oplus D_{i,1} \oplus \cdots \oplus D_{i,k_2-1}, & i = 0, 1, \ldots, k_1 - 1; \\ P_{*,j} = D_{0,j} \oplus D_{1,j} \oplus \cdots \oplus D_{k_1-1,j}, & j = 0, 1, \ldots, k_2 - 1. \end{cases} \tag{1}$$

The diagonal parity strip can then be calculated by

$$P_{*,*} = P_{0,*} \oplus P_{1,*} \oplus \cdots \oplus P_{k_1-1,*}, \tag{2}$$

or

$$P_{*,*} = P_{*,0} \oplus P_{*,1} \oplus \cdots \oplus P_{*,k_2-1}. \tag{3}$$

From Figure 2, we can see that HVPC codes cannot reconstruct four erased strips that form a rectangle erasure pattern. Thus, the fault tolerance of HVPC codes is 3. In addition, in HVPC codes, there is no restriction on the strip size.

There also exist other higher-dimensional parity check codes. We will not enumerate all of them. Please see Rubinoff [1961], Wong and Shea [2001], and Anne and Latifi [2004] for details. In all of them, there is no restriction on the strip size.

4.1.2 *Completely XOR-Based MDS Parity Array Codes.* We now introduce completely XOR-based MDS parity array codes. They come in two types: horizontal codes and vertical codes.

Horizontal codes are those codes in which data and parity elements within a stripe are stored in separate strips. We do not enumerate all of them, but only take EVENODD [Blaum and Menon 1995] with 2 fault tolerance and STAR [Huang and Xu 2005] with 3 fault tolerance as examples. In an EVENODD code with the stripe size $n$, there are $n-2$ data strips and two parity strips in each stripe, and each strip should contain $n-3$ elements, where $n-2$ should be a prime number. In a STAR code with the stripe size $n$, there are $n-3$ data strips and three parity strips in each stripe, and each strip should contain $n-3$ elements, where $n-3$ should be a prime number.

Vertical codes are those codes in which each strip within a stripe contains both data and parity elements. Similarly, we do not enumerate all of them, but only take X-code [Xu and Bruck 1999] with 2 fault tolerance as an example. In an X-code with the stripe size $n$, there should be $n-2$ data elements and two parity elements contained in each strip, where $n$ should be a prime number.

In the earlier discussions, we only focus on those properties that will be involved in the construction of GRID codes. For further details about EVENODD, STAR, and X-code, please refer to Blaum and Menon [1995], Huang and Xu [2005], and Xu and Bruck [1999].

### 4.2 Definition of GRID Codes

Since our GRID codes are *strip-based codes*, we first introduce the concept of strip-based codes and then define our GRID codes.

As shown in the previous subsection, unlike other existing erasure codes that build their redundant relations based on the elements, parity check codes build their redundant relations based on the strips. This leads us to introduce two new concepts.

—*Element-based codes*: the family of erasure codes that build their redundant relations based on the elements;

—*Strip-based codes*: the family of erasure codes that build their redundant relations based on the strips.

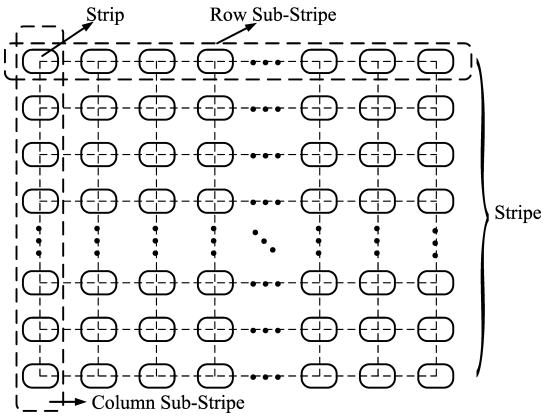



Fig. 3. Stripe layout of a two-dimensional GRID code.

Because strip-based codes build their redundant relations based on higher-level units, we are very interested in how to exploit their potential.

Following this thread, we develop a new family of strip-based codes, named GRID codes. The definition of GRID codes is given as follows.

*Definition* 4.1. GRID codes are a family of strip-based codes whose strips are arranged into multi-dimensional grids.

In this article, we will focus our discussions on two-dimensional GRID codes that evolve from HVPC codes. It is to be noted that the higher-dimensional GRID codes can also be constructed in a similar manner.

Figure 3 illustrates the stripe layout of a two-dimensional GRID code. In a stripe, all the strips are arranged into a two-dimensional grid. All the strips in each row form a row substripe, and those in each column form a column substripe. Suppose that the row substripe size (defined as the number of strips contained in each row substripe) is $n_r$, and the column substripe size (defined as the number of strips contained in each column substripe) is $n_c$. Then, the stripe size of this two-dimensional GRID code is $n = n_r \times n_c$.

In GRID codes, all the row substripes should be encoded by the same erasure code, and all the column substripes should also be encoded by the same erasure code. Moreover, the code chosen for all the row substripes should match with that for all the column substripes. We call the two codes *matched codes*.

*Definition* 4.2. Matched codes are a group of codes that are chosen for constructing a GRID code.

We will discuss how to choose matched codes in the construction of GRID codes in the next subsection.

### 4.3 Construction of GRID Codes

In the construction of GRID codes, the first step is to choose matched codes. In order to construct completely XOR-based GRID codes with high storage efficiency, we use completely XOR-based MDS codes in matched codes, including

Table I. Some Examples of Completely XOR-Based MDS Codes

| Code | | | Fault Tolerance | Stripe Size | Strip Size |
|---|---|---|---|---|---|
| SPC Code | | | $t = 1$ | Arbitrary | — |
| Parity Array Code | Horizontal Code | EVENODD | $t = 2$ | $n = p + 2$ | $s = p - 1$ |
| | | STAR | $t = 3$ | $n = p + 3$ | $s = p$ |
| | Vertical Code | X-Code | $t = 2$ | $n = p$ | $s = p$ |

In this table, $p$ is a prime number.

SPC codes and completely XOR-based MDS parity array codes (such as EVENODD, STAR, and X-code). Table I gives some examples of completely XOR-based MDS codes and shows their important properties, such as fault tolerance, stripe size, and strip size.

The matched codes chosen for a two-dimensional GRID code should consist of two completely XOR-based MDS codes. Suppose that the code chosen for all the row substripes is $code_r$, and that chosen for all the column substripes is $code_c$. We then use $\mathbb{M}(code_r, code_c)$ to represent this group of matched codes. Correspondingly, the two-dimensional GRID code constructed from $\mathbb{M}(code_r, code_c)$ is denoted by $GRID(code_r, code_c)$.

We now discuss how to choose $\mathbb{M}(code_r, code_c)$ for a two-dimensional GRID code. Meanwhile, we discuss how to use $\mathbb{M}(code_r, code_c)$ to construct $GRID(code_r, code_c)$.

To simplify our discussions, we divided all the completely XOR-based MDS codes into two categories: horizontal codes and vertical codes. Obviously, the SPC codes belong to the category of horizontal codes.

According to the preceding classification, there are three cases for a pair of completely XOR-based MDS codes that may be chosen to form $\mathbb{M}(code_r, code_c)$:

—two horizontal codes;

—one horizontal code and one vertical code; and

—two vertical codes.

We will discuss them as follows.

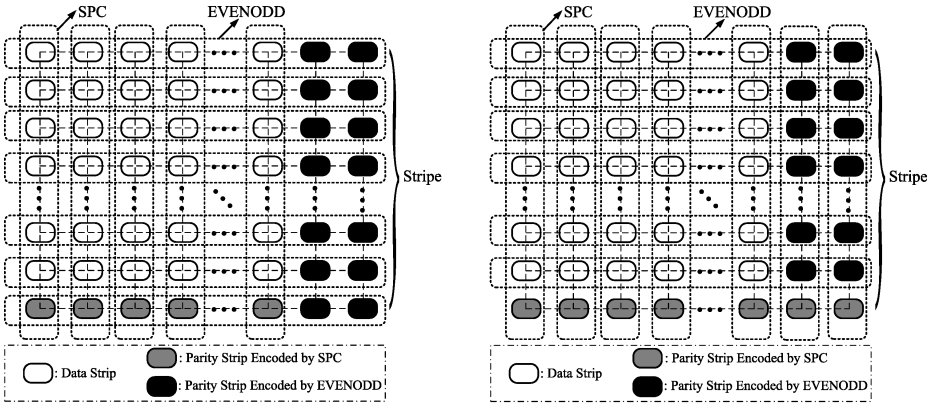
4.3.1 *Case* 1: *Two Horizontal Codes.* In this case, the strip sizes of the two codes should meet the following condition.

*Condition* 4.3. When the two codes in $\mathbb{M}(code_r, code_c)$ consist of two horizontal codes, the strip sizes of $code_r$ and $code_c$ should be equal.

Especially if one code is an SPC code, the other one can be any completely XOR-based MDS code. This is because the fundamental unit of an SPC code is the strip. Moreover, if $code_r$ and $code_c$ are both SPC codes, $GRID(code_r, code_c)$ becomes an HVPC code.

In this case, the corresponding $GRID(code_r, code_c)$ has two kinds of stripe layouts:

—*Row-first layout*. This is the stripe layout in which all the row substripes are first encoded by $code_r$, and the column substripes that do not contain parity strips are then encoded by $code_c$.




(a) row-first layout (b) column-first layout

Fig. 4. Two kinds of stripe layouts of *GRID* (*EVENODD*, *SPC*).

—*Column-first layout*. This is the stripe layout in which all the column substripes are first encoded by $code_c$, and the row substripes that do not contain parity strips are then encoded by $code_r$.

We now take an EVENODD code and an SPC code as an example. They can form $\mathbb{M}$(*EVENODD, SPC*) and thus can be used to construct the GRID code *GRID*(*EVENODD, SPC*). Figure 4 shows the two kinds of stripe layouts of *GRID*(*EVENODD, SPC*). As shown in Figure 4(a), in the row-first layout, all the row substripes are first encoded by the EVENODD code, and the column substripes that do not contain parity strips are then encoded by the SPC code. Similarly, as shown in Figure 4(b), in the column-first layout, all the column substripes are first encoded by the SPC code, and the row substripes that do not contain parity strips are then encoded by the EVENODD code.

4.3.2 *Case* 2: *One Horizontal Code and One Vertical Code.* In this case, the strip sizes of the two codes should meet the following condition.

*Condition* 4.4. When the two codes in $\mathbb{M}(code_r, code_c)$ consist of one horizontal code and one vertical code, the strip sizes of $code_r$ and $code_c$ should be equal.

Unlike in the previous case, the corresponding *GRID* $(code_r, code_c)$ here has only one kind of stripe layout, namely the horizontal-code-first layout. The reason is that in vertical codes, each strip contains both data and parity elements, and this prevents us from constructing the vertical-code-first layout.

We now take a STAR code and an X-code as an example. From Table I, we can see that their strip sizes are both prime numbers and thus can be equal. They can form $\mathbb{M}(STAR, X - Code)$ and thus can be used to construct $GRID(STAR, X - Code)$. Figure 5 shows the one and only kind of stripe layout of $GRID(STAR, X - Code)$. As shown in this figure, all the row substripes are first encoded by the horizontal code STAR, and the column substripes that do not contain parity strips are then encoded by the vertical code X-code.

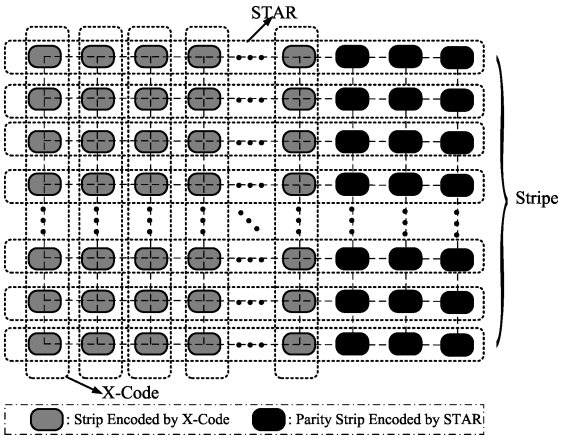



Fig. 5. One and only kind of stripe layout of $GRID(STAR, X-Code)$.

Table II. Some Examples to Illustrate which Two Codes can Form Matched Codes

| | | | SPC | Parity Array Code | | |
|---|---|---|---|---|---|---|
| | | | | Horizontal Code | | Vertical Code |
| | | | Code | EVENODD | STAR | X-Code |
| SPC Code | | | √ | √ | √ | √ |
| Parity Array Code | Horizontal Code | EVENODD | √ | √ | × | × |
| | | STAR | √ | × | √ | √ |
| | Vertical Code | X-Code | √ | × | √ | × |

In this table, the symbol √ is used to denote that the two codes can form *matched codes*, while the symbol × is used to denote that the two codes cannot form *matched codes*.

4.3.3 *Case* 3: *Two Vertical Codes.* From what has been discussed in the previous case, we can see that in the construction of GRID codes, if all the row (column) substripes choose a vertical code, all the column (row) substripes then cannot choose a vertical code again. Therefore, two vertical codes cannot form matched codes and thus cannot be used to construct a two-dimensional GRID code.

Having discussed all the three cases, we now give some examples to illustrate which two codes can form matched codes in Table II. The results in this table can be easily deduced from Table I. From this table, we can make the following observations.

—In $\mathbb{M}(code_r, code_c)$, if one of the two codes is an SPC code, the other one can be any completely XOR-based MDS code.

—A horizontal code always matches with itself.

—In $\mathbb{M}(code_r, code_c)$, the two codes cannot both be vertical codes.

For all of the examples of matched codes in Table II, their corresponding GRID codes can be easily constructed by using the approaches that have been developed in our previous discussions.

**An Iterative Reconstruction Algorithm for** $GRID(code_r, code_c)$

*(1).* For the row substripes containing the erased strips that have not been reconstructed, reconstruct those containing no more than $t_r$ erased strips by using the reconstruction algorithm of $code_r$. If all the row substripes contain more than $t_r$ erased strips, set the flag $F_r \leftarrow 0$; else, set the flag $F_r \leftarrow 1$.
*(2).* For the column substripes containing the erased strips that have not been reconstructed, reconstruct those containing no more than $t_c$ erased strips by using the reconstruction algorithm of $code_c$. If all the column substripes contain more than $t_c$ erased strips, set the flag $F_c \leftarrow 0$; else, set the flag $F_c \leftarrow 1$.
*(3).* If all the erased strips have been reconstructed, return TRUE; else, continue to the next step.
*(4).* If $F_r == 0$ and $F_c == 0$, return FALSE; else, go back to Step (1).

Fig. 6. An iterative reconstruction algorithm for *GRID*($code_r$, $code_c$).

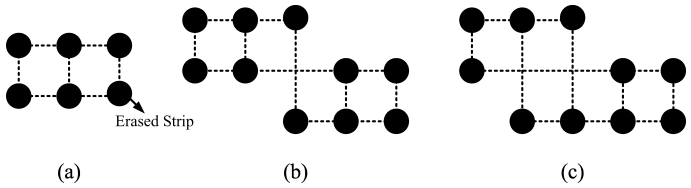



Fig. 7. Three examples of unrecoverable erasure patterns for *GRID*(*EVENODD, SPC*).

### 4.4 Lost Data Reconstruction in GRID Codes

In this subsection, we first propose an iterative reconstruction algorithm for GRID codes. For a two-dimensional GRID code *GRID*($code_r$, $code_c$), suppose that the fault tolerance of $code_r$ is $t_r$, and that of $code_c$ is $t_c$. Then, the iterative reconstruction algorithm for *GRID*($code_r$, $code_c$) is given in Figure 6.

In this algorithm, if TRUE is returned, all the erased strips have been reconstructed; if FALSE is returned, some erased strips cannot be reconstructed, and these remaining erased strips form an *unrecoverable erasure pattern*, which has the following two properties:

(1) the number of erased strips in each row is larger than $t_r$;
(2) the number of erased strips in each column is larger than $t_c$.

Take *GRID* (*EVENODD, SPC*) for example. Figure 7 gives three examples of unrecoverable erasure patterns for *GRID* (*EVENODD, SPC*).

Having given the reconstruction algorithm, we now discuss the fault tolerance of *GRID* ($code_r$, $code_c$). We notice that for a given erasure scenario, if the number of erased strips is smaller than $(t_c+1)\times(t_r+1)$, all the erased strips will be reconstructed by the iterative reconstruction algorithm shown in Figure 6. However, if the number of erased strips is equal to $(t_c + 1) \times (t_r + 1)$, and if all the erased strips form a $(t_c+1)\times(t_r+1)$ array that is an unrecoverable erasure pattern for *GRID* ($code_r$, $code_c$), all the erased strips cannot be reconstructed. Thus, the distance of *GRID* ($code_r$, $code_c$) is $d = (t_c+1)\times(t_r+1)$. Then, we have the following conclusion.

Table III. Fault Tolerance of Some Two-Dimensional GRID Codes

| $GRID(code_r, code_c)$ | Fault Tolerance |
|---|---|
| $GRID(SPC,SPC)$ (i.e. HVPC code) | $t = 3$ |
| $GRID(SPC,EVENODD)$ or $GRID(EVENODD,SPC)$ | $t = 5$ |
| $GRID(SPC, STAR)$ or $GRID(STAR, SPC)$ | $t = 7$ |
| $GRID(SPC, X-Code)$ or $GRID(X-Code, SPC)$ | $t = 5$ |
| $GRID(EVENODD,EVENODD)$ | $t = 8$ |
| $GRID(STAR, STAR)$ | $t = 15$ |
| $GRID(STAR, X-Code)$ or $GRID(X-Code, STAR)$ | $t = 11$ |

*The fault tolerance of* $GRID(code_r, code_c)$ *is*

$$t = (t_c + 1) \times (t_r + 1) - 1, \tag{4}$$

*where* $t_r$ *is the fault tolerance of* $code_r$*, and* $t_c$ *is that of* $code_c$*.*

## 5. FEATURES

In this section, we discuss some important features of GRID codes, including both advantages and disadvantages.

### 5.1 High Fault Tolerance

From Theorem 4.5 in Section 4.4, we can see that for a two-dimensional GRID code $GRID(code_r, code_c)$, its fault tolerance is $t = (t_c + 1) \times (t_r + 1) - 1$, where $t_r$ is the fault tolerance of $code_r$, and $t_c$ is that of $code_c$. Table III gives the fault tolerance of some two-dimensional GRID codes. From this table, we can see that two-dimensional GRID codes can provide up to 15 fault tolerance.

Furthermore, for an any-dimensional GRID code $GRID(code_1, code_2, \ldots, code_m)$ $(m \geq 2)$, we can easily deduce the following conclusion.

THEOREM 5.1. *The fault tolerance of* $GRID(code_1, code_2, \ldots, code_m)$ *is*

$$t = (t_1 + 1) \times (t_2 + 1) \times \cdots \times (t_m + 1) - 1, \tag{5}$$

*where* $m \geq 2$*, and* $t_i$ *is the fault tolerance of* $code_i$ $(i = 1, 2, \ldots, m)$*.*

This theorem further shows that GRID codes can provide very high fault tolerance.

From the preceding discussions, we can see that GRID codes can provide up to 15 and even higher fault tolerance and thus are very suitable for large-scale storage systems in which the possibility of concurrent disk failures, together with multiple unrecoverable sector errors, is very remarkable.

In addition, since GRID codes are completely XOR-based non-MDS codes, they can be exploited to tolerate more faults beyond their fault tolerance by using the method proposed in Wylie and Swaminathan [2007]. Especially when applied to those storage systems that are composed of heterogeneous devices with different failure and recovery rates, GRID codes can be further exploited to work better by using the method proposed in Greenan and Wylie [2008].

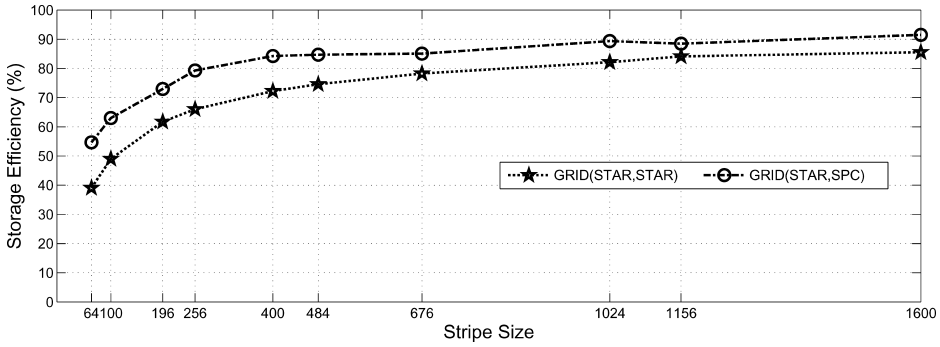



Fig. 8. Optimal storage efficiency of $GRID(STAR, STAR)$ and $GRID(STAR, SPC)$ in some stripe sizes. In this figure, for each stripe size $n$, the storage efficiency of each code is calculated by using the $(n_r, n_c)$ pair with the highest storage efficiency.

## 5.2 High Storage Efficiency

For a two-dimensional GRID code $GRID\,(code_r, code_c)$, according to the types of $code_r$ and $code_c$, we will discuss its storage efficiency in two cases.

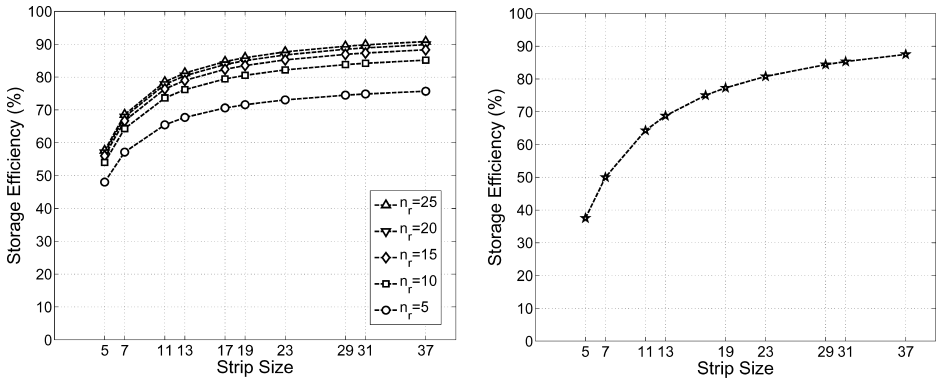
5.2.1 *Case* 1: *Two Horizontal Codes.* In this case, suppose that the row substripe size is $n_r$, and the column substripe size is $n_c$. Then, the stripe size of $GRID(code_r, code_c)$ is $n = n_r \times n_c$. In addition, suppose that the fault tolerance of $code_r$ is $t_r$, and that of $code_c$ is $t_c$. Then, there are $t_r \times n_c + t_c \times n_r - t_r \times t_c$ parity strips in each stripe. Thus, the corresponding storage efficiency is

$$E = 1 - \frac{t_r \times n_c + t_c \times n_r - t_r \times t_c}{n}. \tag{6}$$

From this formula, we can see that $E$ depends on $n$, $n_r$, $n_c$, $t_r$, and $t_c$, and increases with $n$.

To further understand how the storage efficiency $E$ increases with the stripe size $n$, we take $GRID\,(STAR, STAR)$ with $t = 15$ and $GRID\,(STAR, SPC)$ with $t = 7$ for an example and investigate their storage efficiency in different stripe sizes. Since the stripe size is $n = n_c \times n_r$, we notice that for a given stripe size $n$, there sometimes exists more than one $(n_r, n_c)$ pair. For example, for $GRID\,(STAR, SPC)$, when $n = 256$, there exist four such $(n_r, n_c)$ pairs: $(8, 32)$, $(16, 16)$, $(32, 8)$, and $(64, 4)$. For each $(n_r, n_c)$ pair, we can calculate the corresponding storage efficiency by using Eq. (6). In this process, we find that among these $(n_r, n_c)$ pairs, the one in which the value of $n_r/n_c$ is closer to that of $t_r/t_c$ has higher storage efficiency. In practice, we will always choose the $(n_r, n_c)$ pair with the highest storage efficiency. Figure 8 gives the optimal storage efficiency of $GRID\,(STAR, STAR)$ and $GRID\,(STAR, SPC)$ in some stripe sizes. From this figure, we can make the following observations.

—The storage efficiency of a GRID code increases with the stripe size and can increase to a very high level, even higher than 80%.




(a) $GRID(SPC, X-Code)$ (b) $GRID(STAR, X-Code)$

Fig. 9. Storage efficiency of *GRID* (*SPC*, $X-Code$) and *GRID* (*STAR*, $X-Code$).

—The GRID code with higher fault tolerance always has lower storage efficiency than that with lower fault tolerance. This shows a trade-off between fault tolerance and storage efficiency.

5.2.2 *Case* 2: *One Horizontal Code and One Vertical Code.* In this case, suppose that the stripe size and fault tolerance of the horizontal code are $n_h$ and $t_h$, respectively, and those of the vertical code are $n_v$ and $t_v$, respectively. It is clear that there are $t_h \times n_v$ parity strips encoded by the horizontal code. Thus, the number of parity elements contained in these $t_h \times n_v$ parity strips is $s \times t_h \times n_v$. In addition, in the remaining $(n_h - t_h) \times n_v$ strips that are encoded by the vertical code, there are $t_v \times (n_h - t_h) \times n_v$ parity elements. Then, there are $s \times t_h \times n_v + t_v \times (n_h - t_h) \times n_v$ parity elements in total. Thus, the corresponding storage efficiency is $E = 1 - \frac{s \times t_h \times n_v + t_v \times (n_h - t_h) \times n_v}{s \times n_h \times n_v}$, which is equivalent to

$$E = 1 - \frac{s \times t_h + t_v \times n_h - t_h \times t_v}{s \times n_h}. \tag{7}$$

From this formula, we can see that $E$ depends on $s$, $n_h$, $t_h$, and $t_v$, and increases with $s$ and $n_h$.

To further understand how the storage efficiency $E$ increases with the strip size $s$ and the stripe size of the horizontal code $n_h$, we take *GRID* (*SPC*, $X-Code$) and *GRID* (*STAR*, $X-Code$) as an example and discuss their storage efficiency, respectively.

In *GRID* (*SPC,X-Code*), $s$ should be a prime number that is larger than $t_v$, $n_h$ is $n_r$ that should meet $n_h \geq 3$, $t_h = 1$, and $t_v = 2$. Thus, the storage efficiency of *GRID* (*SPC*, $X - Code$) is $E = 1 - \frac{s \times 1 + 2 \times n_r - 1 \times 2}{s \times n_r} = 1 - \frac{s + 2n_r - 2}{s \times n_r}$, where $s$ is a prime number that is larger than 2, and $n_r \geq 3$. Figure 9(a) gives the storage efficiency of *GRID* (*SPC*, $X - Code$) in some scenarios with different strip sizes and different row substripe sizes. From this figure, we can see that the storage efficiency of *GRID* (*STAR*, $X - Code$) increases with the strip size and the row substripe size and can increase to a very high level, even higher than 80%.

In *GRID* (*STAR*, $X-Code$), $s$ should be a prime number that is larger than $t_v$, $n_h = s+3$, $t_h = 3$, and $t_v = 2$. Thus, the storage efficiency of *GRID* (*STAR*, $X-Code$) is $E = 1 - \frac{s\times 3+2\times(s+3)-3\times 2}{s\times(s+3)} = 1 - \frac{5}{s+3}$, where $s$ is a prime number that is larger than 2. Figure 9(b) gives the storage efficiency of *GRID* (*STAR*, $X-Code$) in some strip sizes. From this figure, we can see that the storage efficiency of *GRID* (*STAR*, $X-Code$) increases with the strip size and can increase to a very high level, even higher than 80%.

From what we have discussed in this subsection, we can see that two-dimensional GRID codes often provide high storage efficiency, and in some scenarios their storage efficiency can reach up to 80% and even higher. This advantage can become very remarkable when they are applied in large-scale storage systems.

Furthermore, for higher-dimensional GRID codes, we can easily deduce similar conclusions.

### 5.3 Optimal Small-Write Performance

Poor small-write performance is one of the primary challenges in the design of erasure-coded storage systems. We thus discuss the small-write performance of GRID codes in this subsection.

As we know, in erasure codes, for a given fault tolerance $t$, the theoretical lower bound of the data out-degree is $t$, and the corresponding lower bound of the number of I/Os involved in each small-write is $2(t+1)$ (here, the small-write operation is performed by a read-modify-write process). Thus, to provide good small-write performance, in the construction of GRID codes, we always choose those codes with the property that the data out-degree is equal to the fault tolerance. Examples of such codes are SPC codes, EVENODD, STAR, and X-code.

In a two-dimensional GRID code *GRID* ($code_r$, $code_c$), we can deduce that the data out-degree is $(t_c+1)\times(t_r+1)-1$, which is equal to the fault tolerance $t$. Thus, there are only $2(t+1)$ I/Os involved in each small write. Therefore, *GRID* ($code_r$, $code_c$) provides optimal small-write performance.

Similarly, for higher-dimensional GRID codes, we can easily deduce the same conclusion.

### 5.4 Low Reconstruction Cost

In GRID codes, when $t$ strips are erased in a stripe, a host can reconstruct them by using the iterative reconstruction algorithm developed in Section 4.4. In this process, the host only needs to read those substripes that contain the $t$ erased strips. Thus, the number of I/Os involved in this process is often much smaller than the stripe size $n$. As we know, in most existing erasure codes, such as Reed-Solomon codes [Reed and Solomon 1960], when $t$ strips are erased in a stripe, the host needs to read all the remaining strips to reconstruct them and then write the reconstructed data back to the disks, and this process can involve $n$ I/Os. Thus, when $t$ strips are erased in a stripe, compared with these existing codes, GRID codes have a significant advantage with respect to reconstruction cost. Similarly, when fewer than $t$ strips are erased in a stripe, we can easily

deduce the same conclusion. Therefore, when some strips are erased in a stripe, GRID codes often have low reconstruction cost.

### 5.5 Localization Effects

In GRID codes, each stripe consists of substripes, and each substripe is encoded by its own code. Thus, we can deduce that the data out-degree and parity in-degree of a GRID code are often much smaller than its stripe size, providing proof of both a localized write-lock zone and a localized reconstruction zone. Therefore, in GRID codes, many operations, such as small-write operations and reconstruction operations, have localization effects.

### 5.6 Flexibilities and Restrictions

There are some flexibilities and restrictions in GRID codes. We discuss them as follows.

5.6.1 *Flexibilities in the Construction of GRID Codes.* There are some flexibilities in the construction of GRID codes. From Table III, we can see that for a given fault tolerance, there sometimes exists more than one kind of GRID code. For example, for $t = 5$, there exist four kinds of GRID codes, namely $GRID(SPC, EVENODD)$, $GRID(EVENODD, SPC)$, $GRID(SPC, X - Code)$, and $GRID(X - Code, SPC)$. Moreover, as mentioned in Section 4.3, for a GRID code $GRID(code_r, code_c)$, if the two codes $code_r$ and $code_c$ are both horizontal codes, there always exist two kinds of stripe layouts. One such example is $GRID(EVENODD, SPC)$, of which the two kinds of stripe layouts are shown in Figure 4. In practice, we can make our decisions on choosing which GRID code and which stripe layout according to the requirements of the practical systems.

5.6.2 *Restrictions on the Stripe Layouts of GRID Codes.* There are some restrictions on the stripe layouts of GRID codes. We take a two-dimensional GRID code $GRID(code_r, code_c)$ as an example. The stripe size $n$ should be $n = n_r \times n_c$, where $n_r$ and $n_c$ are the row substripe size and column substripe size, respectively. In addition, the strip size of $code_r$ should be equal to that of $code_c$. Moreover, as shown in Table I, there are sometimes some restrictions on the stripe sizes of $code_r$ and $code_c$ (though we can simply assume that there are more disks that have all zeros without affecting the encoding and decoding procedures [Blaum and Menon 1995]). From this example, we can see that in GRID codes, there are some restrictions on the stripe size and strip size, which can together determine the stripe layouts.

5.6.3 *Restrictions on the Fault Tolerance of GRID Codes.* In addition, there are also some restrictions on the fault tolerance of GRID codes. From Theorem 5.1, we can see that for a GRID code $GRID(code_1, code_2, \ldots, code_m)$ $(m \geq 2)$, its fault tolerance is $t = (t_1 + 1) \times (t_2 + 1) \times \cdots \times (t_m + 1) - 1$, where $t_i$ is the fault tolerance of $code_i$ $(i = 1, 2, \ldots, m)$. Thus, we can deduce the following conclusion.

Table IV. GRID Codes vs. Reed-Solomon Codes

| | Fault Tolerance | Storage Efficiency | Operation |
|---|---|---|---|
| GRID Codes | $t = (t_1 + 1) \times (t_2 + 1) \times \cdots \times (t_m + 1) - 1$ | Non-optimal | XOR |
| Reed-Solomon Codes | Arbitrary | Optimal | Finite field arithmetic |

THEOREM 5.2. *There is no GRID code with the fault tolerance $t^*$ when $t^* + 1$ is a prime number.*

For example, there is no GRID code with the fault tolerance 10 because $10 + 1 = 11$ is a prime number.

In practice, for a given fault tolerance, if there is no GRID code with this fault tolerance, we always choose a GRID code with a higher fault tolerance.

## 6. COMPARISONS

We compare GRID codes with other existing erasure codes in this section.

### 6.1 GRID Codes vs. Reed-Solomon Codes

Table IV compares GRID codes with Reed-Solomon codes [Reed and Solomon 1960; Roth and Lempel 1989]. From this table, we can see that the primary advantage of GRID codes over Reed-Solomon codes is that the former require simpler operations. As mentioned in Plank [2005], the bandwidth of XOR (often 3 GB/s) is much higher than that of multiplication over the Galois field $GF(2^w)$ (e.g., 150 MB/s over $GF(2^{16})$). Therefore, GRID codes can have much better performance than Reed-Solomon codes.

### 6.2 GRID Codes vs. Parity Array Codes

Table V compares GRID codes with parity array codes. From this table, we can make the following observations.

—Compared with EVENODD [Blaum and Menon 1995], RDP codes [Corbett and Sankar 2004], STAR [Huang and Xu 2005], Liberation codes [Plank 2008], and X-code [Xu and Bruck 1999], which are MDS codes and are completely based on simple XOR operations, GRID codes can provide much higher fault tolerance.

—Compared with WEAVER codes [Hafner 2005], which are completely based on simple XOR operations and can provide high fault tolerance, GRID codes can provide higher fault tolerance and higher storage efficiency.

—Compared with the generalizations of EVENODD [Blaum and Vardy 2001, 1996] and Feng's codes [Feng and Shen 2005a, 2005b], which are MDS codes and can provide high fault tolerance, GRID codes require simpler operations and thus can have better performance.

Table V. GRID Codes vs. Parity Array Codes

| | Fault Tolerance | Storage Efficiency | Operation |
|---|---|---|---|
| GRID Codes | Up to 15 and even higher | Non-optimal | XOR |
| EVENODD | 2 | Optimal | |
| RDP Codes | 2 | | |
| STAR | 3 | | |
| Liberation Codes | 2 | | |
| X-Code | 2 | | |
| GRID Codes | Up to 15 and even higher | Up to 80% and even higher | XOR |
| WEAVER Codes | Up to 12 | Not higher than 50% | |
| GRID Codes | $t = (t_1 + 1) \times (t_2 + 1) \times \cdots \times (t_m + 1) - 1$ | Non-optimal | XOR |
| Generalizations of EVENODD | From 3 to 8 | Optimal | Ring arithmetic |
| Feng's Codes | Arbitrary | | |

### 6.3 GRID Codes vs. Parity Check Codes

GRID codes and parity check codes [Rubinoff 1961; Wong and Shea 2001; Anne and Latifi 2004] are both completely XOR-based codes. We compare their fault tolerance and storage efficiency in the following content.

We first compare their fault tolerance. We take two-dimensional strip-based codes as an example. The fault tolerance of horizontal and vertical parity check (HVPC) code is 3, while that of a two-dimensional GRID code can be up to 15 and perhaps higher. Thus, we can easily deduce that GRID codes always have higher fault tolerance than parity check codes.

We now compare their storage efficiency. We take a three-dimensional parity check (3DPC) code [Wong and Shea 2001] and a GRID code *GRID* (*STAR, SPC*), both with 7 fault tolerance, as an example. Figure 10 gives their optimal storage efficiency in some stripe sizes. In this figure, all the values of storage efficiency are calculated in the same manner as in Figure 8. From this figure, we can see that *GRID* (*STAR*, *SPC*) has higher storage efficiency than 3DPC. Thus, we can deduce that GRID codes always have higher storage efficiency than parity check codes.

### 6.4 GRID Codes vs. LDPC Codes

Low-density parity check (LDPC) codes [Gallager 1962; Luby and Spielman 2001] were originally designed for communication channels, but have recently been studied in the context of storage applications over wide-area networks [Plank and Thomason 2004, 2005]. Good LDPC codes with high fault tolerance and near-optimal storage efficiency often have highly irregular structures. Compared with these codes, GRID codes have very regular structures and thus can be more easily implemented in storage systems.

From what we have discussed in this section, we can see that GRID codes have attractive advantages over many existing erasure codes.

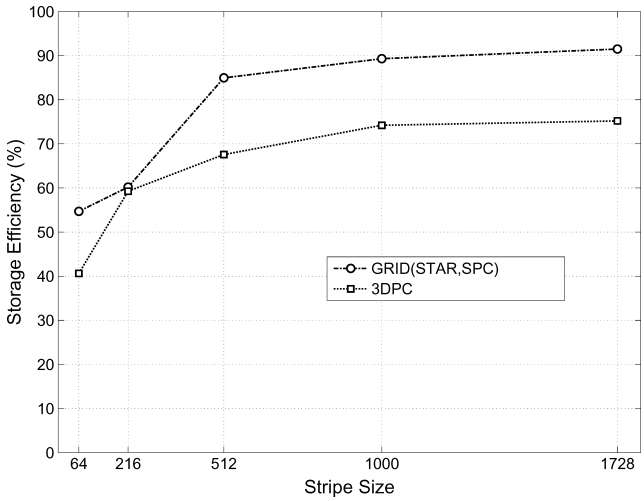



Fig. 10. Optimal storage efficiency of *GRID*(*STAR*, *SPC*) and 3DPC in some stripe sizes.

—They are completely XOR-based and have very regular structures, ensuring easy implementation;
—they can provide up to 15 and even higher fault tolerance; and
—their storage efficiencies can reach up to 80% and even higher.

All these advantages make GRID codes more suitable for large-scale storage systems.

## 7. CONCLUSIONS

This article has presented a new family of erasure codes with high fault tolerance, named GRID codes. They are called such because they are a family of strip-based codes whose strips are arranged into multi-dimensional grids. These codes have several significant features: (a) They are completely XOR-based and have very regular structures, ensuring easy implementation; (b) they can provide up to 15 and even higher fault tolerance; (c) although they are not Maximum Distance Separable (MDS) codes, their storage efficiency can reach up to 80% and even higher; (d) they provide optimal small-write performance; (e) they often have low reconstruction cost; (f) their operations, such as small-write operations and reconstruction operations, have localization effects; (g) there are some flexibilities in their construction and also some restrictions on their stripe layouts and fault tolerance. GRID codes provide the designers of storage systems with good trade-offs among fault tolerance, performance, and storage efficiency. All the features make GRID codes more suitable for any large-scale storage system with the requirements of high fault tolerance, high performance, and high storage efficiency.

It is to be noted that besides the GRID codes whose strips are arranged into multi-dimensional grids, there could be other kinds of strip-based codes whose strips could be arranged into other geometrical shapes. Hence, one of our future

research directions is to exploit other geometrical shapes for constructing new kinds of strip-based codes.

## REFERENCES


AGUILERA, M. K., JANAKIRAMAN, R., AND XU, L. 2005. Using erasure codes efficiently for storage in a distributed system. In *Proceedings of the Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'05)*, 336–345.

ANNE, N. B., THIRUNAVUKKARASU, U., AND LATIFI, S. 2004. Three and four-dimensional parity-check codes for correction and detection of multiple errors. In *Proceedings of the International Conference on Information Technology: Coding and Computing (ITCC),* vol. 2. IEEE Computer Society, 840–845.

BAIRAVASUNDARAM, L. N., GOODSON, G. R., PASUPATHY, S., AND SCHINDLER, J. 2007. An analysis of latent sector errors in disk drives. In *Proceedings of the ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*. ACM Press, New York, 289–300.

BLAUM, M., BRADY, J., BRUCK, J., MENON, J., AND VARDY, A. 2001. The EVENODD code and its generalization: An efficient scheme for tolerating multiple disk failures in RAID architectures. In *High Performance Mass Storage and Parallel I/O: Technologies and Applications,* 187–208.

BLAUM, M., BRUCK, J. AND VARDY, A. 1996. MDS array codes with independent parity symnbols. *IEEE Trans. Inf. Theory 42*, 2, 529–542.

BLAUM, M., BRADY, J., BRUCK, J., AND MENON, J. 1995. EVENODD: An efficient scheme for tolerating double disk failures in RAID architectures. *IEEE Trans. Comput. 44*, 2, 192–202.

BLOEMER, J., KALFANE, M., KARP, R., KARPINSKI, M., LUBY, M., AND ZUCKERMAN, D. 1995. An XOR-based erasure resilient coding scheme. Tech. rep. TR-95-048, International Computer Science Institute, Berkeley, California.

CHEN, P. M., LEE, E. K., GIBSON, G. A., KATZ, R. H., AND PATTERSON, D. A. 1994. RAID: High-Performance, reliable secondary storage. *ACM Comput. Surv. 26*, 2, 145–185.

COLLINS, R. L. AND PLANK, J. S. 2005. Assessing the performance of erasure codes in the wide-area. In *Proceedings of the Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'05)*, 182–187.

CORBETT, P., ENGLISH, B., GOEL, A., GRCANAC, T., KLEIMAN, S., LEONG, J., AND SANKAR, S. 2004. Row-Diagonal parity for double disk failure correction. In *Proceedings of the 3rd USENIX Conference on File and Storage Technologies (FAST'04).* USENIX Association, 1–14.

FENG, G.-L., DENG, R. H., BAO, F., AND SHEN, J. C. 2005a. New efficient MDS array codes for RAID, Part I. Reed-Solomon-Like codes for tolerating three disk failures. *IEEE Trans. Comput. 54*, 9, 1071–1080.

FENG, G.-L., DENG, R. H., BAO, F., AND SHEN, J. C. 2005b. New efficient MDS array codes for RAID, Part II. Rabin-Like codes for tolerating multiple (greater than or equal to 4) disk failures. *IEEE Trans. Comput. 54*, 12, 1473–1483.

FRØLUND, S. MERCHANT, A., SAITO, Y., SPENCE, S., AND VEITCH, A. 2004. A decentralized algorithm for erasure-coded virtual disks. In *Proceedings of the Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'04),* 125–134.

GALLAGER, R. G. 1962. Low density parity check codes. *IRE Trans. Inf. Theory 8*, 1, 21–28.

GOODSON, G. R., WYLIE, J. J., GRANGER, G. R., AND REITER, M. K. 2004. Efficient Byzantine-tolerant erasure-coded storage. In *Proceedings of the Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'04),* 135–144.

GREENAN, K. M., MILLER, E. L., AND WYLIE, J. J. 2008. Reliability of flat XOR-based erasure codes on heterogeneous devices. In *Proceedings of the 38th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'08).* IEEE Computer Society, 147–156.

HAFNER, J. L. 2006. Hover erasure codes for disk arrays. In *Proceedings of the Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'06).* IEEE Computer Society, 217–226.

HAFNER, J. L. 2005. Weaver codes: Highly fault tolerant erasure codes for storage systems. In *Proceedings of the 4th USENIX Conference on File and Storage Technologies (FAST'05).* USENIX Association, 211–224.

HAFNER, J. L., DEENADHAYALAN, V., KANUNGO, T., AND RAO, K. K. 2004. Performance metrics for erasure codes in storage systems. Tech. rep. RJ 10321 (A0408-003). IBM Research Division, Almaden Research Center. August.

HUANG, C. AND XU, L. 2005. Star: An efficient coding scheme for correcting triple storage node failures. In *Proceedings of the 4th USENIX Conference on File and Storage Technologies (FAST'05)*. USENIX Association.

LUBY, M. C., MITZENMACHER, M., SHOKROLLAHI, M. A., AND SPIELMAN, D. A. 2001. Efficient erasure correcting codes. *IEEE Trans. Inf. Theory 47*, 2, 569–584.

MACWILLIAMS, F. J. AND SLOANE, N. J. A. 1977. *The Theory of Error-Correcting Codes*. North-Holland, New York.

PINHEIRO, E., WEBER, W. D., AND BARROSO, L. A. 2007. Failure trends in a large disk drive population. In *Proceedings of the 5th USENIX Conference on File and Storage Technologies (FAST'07)*. USENIX Association, 17–29.

PLANK, J. S. 2008. The RAID-6 liberation codes. In *Proceedings of the 6th USENIX Conference on File and Storage Technologies (FAST'08)*. USENIX Association, 1–14.

PLANK, J. S. 2005. Erasure codes for storage applications. Tutorial slides presented at *the 4th USENIX Conference on File and Storage Technologies (FAST'05)*. http://www.cs.utk.edu/~plank/plank/papers/FAST-2005.html.

PLANK, J. S. AND THOMASON, M. G. 2004. A practical analysis of low-density parity-check erasure codes for wide-area storage applications. In *Proceedings of the Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'04)*. IEEE Computer Society.

PLANK, J. S. AND XU, L. 2006. Optimizing Cauchy Reed-Solomon codes for fault-tolerant network storage applications. In *Proceedings of the 5th IEEE International Symposium on Network Computing and Applications (NCA'06)*. IEEE Computer Society, 173–180.

PLANK, J. S., BUCHSBAUM, A. L., COLLINS, R. L., AND THOMASON, M. G. 2005. Small parity-check erasure codes- Exploration and observations. In *Proceedings of the Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'05)*. IEEE Computer Society, 326–335.

REED, I. S. AND SOLOMON, G. 1960. Polynomial codes over certain finite fields. *J. Soc. Industrial Appl. Math. 8*, 2, 300–304.

ROTH, R. M. AND LEMPEL, A. 1989. On MDS codes via Cauchy matrices. *IEEE Trans. Inf. Theory 35*, 6, 1314–1319.

RUBINOFF, M. 1961. N-Dimensional codes for detecting and correcting multiple errors. *Commun. ACM 4*, 12, 545–551.

SCHROEDER, B. AND GIBSON, G. A. 2007. Disk failures in the real world: What does an MTTF of 1,000,000 hours mean to you? In *Proceedings of the 5th USENIX Conference on File and Storage Technologies (FAST'07)*. USENIX Association, 1–16.

TANNER, R. M. 1981. A recursive approach to low complexity codes. *IEEE Trans. Inf. Theory 27*, 5, 533–547.

WILCKE, W. W., GARNER, R. B., FLEINER, C., FREITAS, R. F., GOLDING, R. A., GLIDER, J. S., KENCHAMMANA-HOSEKOTE, D. R., HAFNER, J. L., MOHIUDDIN, K. M., RAO, K. K., BECKER-SZENDY, R. A., WONG, T. M., ZAKI, O. A., HERNANDEZ, M., FERNANDEZ, K. R., HUELS, H., LENK, H., SMOLIN, K., RIES, M., GOETTERT, C., PICUNKO, T., KAHN, H., AND LOO, T. 2006. IBM intelligent bricks project: Petabytes and beyond. *IBM J. Res. Devel. 50*, 2-3, 181–197.

WONG, T. E., AND SHEA, J. M. 2001. Multi-Dimensional parity check codes for bursty channels. In *Proceedings of the IEEE International Symposium on Information Theory (ISIT'01)*, 123.

WYLIE, J. J. AND SWAMINATHAN, R. 2007. Determining fault tolerance of XOR-based erasure codes efficiently. In *Proceedings of the 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'07)*. IEEE Computer Society, 206–215.

XIA, H. AND CHIEN, A. A. 2007. Robustore: A distributed storage architecture with robust and high performance. In *Proceedings of the ACM/IEEE Conference on SuperComputing (SC'07)*, 1–11.

XU, L. AND BRUCK, J. 1999. X-Code: MDS array codes with optimal encoding. *IEEE Trans. Inf. Theory 45*, 1, 272–276.

Received June 2007; revised December 2008; accepted December 2008