Pyramid Codes: Flexible Schemes to Trade Space for Access Efficiency in Reliable Data Storage Systems

CHENG HUANG, Microsoft Research
MINGHUA CHEN, Chinese University of Hong Kong
JIN LI, Microsoft Research

We design flexible schemes to explore the tradeoffs between storage space and access efficiency in reliable data storage systems. Aiming at this goal, two new classes of erasure-resilient codes are introduced – Basic Pyramid Codes (BPC) and Generalized Pyramid Codes (GPC). Both schemes require slightly more storage space than conventional schemes, but significantly improve the critical performance of read during failures and unavailability.

As a by-product, we establish a necessary matching condition to characterize the limit of failure recovery, that is, unless the matching condition is satisfied, a failure case is impossible to recover. In addition, we define a maximally recoverable (MR) property. For all ERC schemes holding the MR property, the matching condition becomes sufficient, that is, all failure cases satisfying the matching condition are indeed recoverable. We show that GPC is the *first* class of non-MDS schemes holding the MR property.

Categories and Subject Descriptors: C.4 [**Computer Systems Organization**]: Performance of Systems—*Reliability, availability, and serviceability*; E.4 [**Data**]: Coding and Information Theory; E.5 [**Data**]: Files—*Backup/recovery*

General Terms: Design, Algorithms, Performance, Reliability

Additional Key Words and Phrases: Storage, erasure codes, reconstruction, fault tolerance

ACM Reference Format:
Huang, C., Chen, M., and Li, J. 2013. Pyramid codes: Flexible schemes to trade space for access efficiency in reliable data storage systems. *ACM Trans. Storage* 9, 1, Article 3 (March 2013), 28 pages.
DOI:http://dx.doi.org/10.1145/2435204.2435207

1. INTRODUCTION

A promising direction in building large-scale storage systems is to harness the collective storage capacity of massive commodity computers. While many systems demand high reliability (such as eleven 9s), individual components can rarely live up to that standard. For example, a recent study [Schroeder and Gibson 2007] shows that the real-world reliability of disk drives is far lower than expected.

On the other hand, large-scale production systems (e.g., Google File System [Ghemawat et al. 2003], Hadoop [Shvachko et al. 2010], and Windows Azure Storage [Calder et al. 2011]) have successfully demonstrated the feasibility of building reliable

M. Chen's research is partially supported by the China 973 Program (Project 2012CB315904), the General Research Fund (Projects 411209, 411010, and 411011) and an Area of Excellence Grant (Project AoE/E-02/08), all established under the University Grant Committee of the Hong Kong SAR, China, as well as an Open Project of Shenzhen Key Lab of Cloud Computing Technology and Application, and two gift grants from Microsoft and Cisco.

Authors' addresses: C. Huang (corresponding author) and J. Li, Microsoft Research; email: cheng.huang@microsoft.com; M. Chen, Chinese University of Hong Kong.

DOI:http://dx.doi.org/10.1145/2435204.2435207

data storage systems using much less reliable commodity components. These systems often use replication schemes to ensure reliability, where each data block is replicated a few times. Trivial as it seems, there are sound reasons for such a choice. The simplicity of design, implementation, and verification is perhaps the most important one. In addition, replication schemes demonstrate good I/O performance. For instance, in a 3-replication scheme (each data block is stored with 2 additional replicas), writing a data block takes 3 write operations (1 write to itself and 2 to its replicas) and reading simply takes 1 read operation (from the original data block or either of the replicas).

On the downside, replication schemes consume several times more storage spaces than the data collection itself. In data centers, storage overhead directly translates into costs in hardware (disk drives and associated machines), as well as costs to operate systems, which include building space, power, cooling, and maintenance. As a matter of fact, it is reported that over 55% of the cost of Microsoft Windows Live data center is due to building, power distribution and equipment [Hamilton 2007]. In wide-area peer-to-peer storage systems, as many as 24 replicas are required to achieve only modest reliability [Grolimund 2007], which makes replication schemes prohibitively expensive.

To that end, *erasure resilient coding* (ERC)-based schemes are proposed to reduce storage overhead without compromising reliability. The most common ERC scheme applies *systematic* MDS codes [MacWilliams and Sloane 1977], where a mathematical transform maps k data blocks into n total blocks (k original data blocks and $n - k$ redundant blocks). When blocks fail, as long as there are no more than $n - k$ failures, failed blocks can be recovered using remaining data and redundant blocks (imaginably, via the inverse of the mathematical transform). Such an ERC scheme is often called a (n, k)-MDS scheme. The storage saving of an MDS scheme is superior. For instance, compared to 3-replication, a (9, 6)-MDS scheme can reduce storage overhead from 3x to 1.5x, a 50% saving!

The adoption of ERC schemes in large-scale production systems, however, faces two fundamental performance obstacles. First, the complexity and cost of update is high. In the $(9, 6)$-MDS scheme, in-place updating a data block requires 8 read/write operations – 1 read of the data block itself to compute the bit-wise difference between old and new data (called *delta*), 1 write to update the block, 3 reads of the 3 redundant blocks to compute deltas, and then 3 writes to update the redundant blocks [Aguilera et al. 2005; Saito et al. 2004]). In contrast, update a data block in 3-replication systems requires three writes. Second, the performance of data access may suffer as well. In systematic ERC schemes, the original data is preserved in the k data blocks. Hence, a read operation, which accesses a certain byte range of data, can be directed to the data block containing the target range. There is no penalty in such data access. However, if the target data block is unavailable, the read operation has to access the remaining blocks and perform the inverse of the mathematical transform to *reconstruct* the missing data. For instance, in the (9, 6)-MDS scheme, reconstructing 1 data block requires reading from 6 data and redundant blocks. Clearly, reconstruction incurs penalty due to more disk I/O and network traffic.

The first obstacle is well-addressed in today's production storage systems (such as Hadoop, GFS, and WAS). These systems have been successfully built on top of append-only storage, where write operations only append to the end of existing data and data is never modified once written. Updates are appended as new data in the systems and lazily consolidated as background jobs. Such systems provide a perfect opportunity to implement ERC schemes because in-place update is completely avoided.

The second obstacle, unfortunately, has *not* been adequately examined. Due to the scale of the production storage systems, component failures and transient unavailability are no longer rare events, but happen rather regularly [Ford et al. 2010]. Data access during failures and unavailability now impacts SLAs directly. The focus of this work is to address this critical problem and improve the performance of data access during failures and unavailability.

Our key idea is inspired by the drastic gap between replication and MDS-based ERC. The storage overhead of 3-replication is as high as 3x, but reconstruction requires merely 1 block. On the other hand, the storage overhead of the (9, 6)-MDS scheme is low at 1.5x, but reconstruction requires as many as 6 blocks. If replication and MDS-based ERC are regarded as two extremes of the tradeoffs between storage space and access efficiency, is it possible to design schemes to explore the middle ground? We provide an affirmative answer through this work. Our contributions are as follows.

- We design two new ERC schemes: basic pyramid codes (BPC) and generalized pyramid codes (GPC). Both schemes require slightly more storage space than MDS-based ERC, but significantly improve the performance of reconstruction. For instance, compared to MDS-based ERC, pyramid codes reduce reconstruction read cost by 50% with merely 11% additional storage overhead.
- As a by-product, we establish a necessary matching condition to characterize the limit of failure recovery, that is, unless the matching condition is satisfied, a failure case is impossible to recover. In addition, we define a maximally recoverable (MR) property. For all ERC schemes holding the MR property, the matching condition becomes sufficient, that is, all failure cases satisfying the matching condition are indeed recoverable. We show that GPC is the first class of non-MDS schemes holding the MR property.

Since first published as conference papers in 2007 [Chen et al. 2007; Huang et al. 2007], BPC has been implemented in archival storage systems [Wildani et al. 2009]. GPC and the MR property have inspired the design of new classes of codes, such as (r, d)-codes [Gopalan et al. 2011] and local reconstruction codes (LRC) [Huang et al. 2012]. In particular, LRC has been implemented in Windows Azure Storage, saving the Microsoft cloud millions of dollars [Gantenbein 2012].

2. RELATED WORK

2.1. Erasure Coding in Storage Systems

Erasure coding has long been applied in storage systems, from earlier disk arrays [Chen et al. 1994] to later wide-area storage system, such as OceanStore [Kubiatowicz et al. 2000; Rhea et al. 2003]; Free Haven [Dingledine et al. 2000]; PAST [Rowstron and Druschel 2001]; and local distributed cluster-based storage, such as Intermemory [Chen et al. 1999]; PASIS [Wylie et al. 2000]; Myraid [Chang et al. 2002]; Glacier [Haeberlen et al. 2005]; Ursa Minor [Abd-El-Malek et al. 2005]; Panasas Parallel FS [Welch et al. 2008]; and HydraStor [Dubnicki et al. 2009; Ungureanu et al. 2010]. The advantages of erasure coding over simple replication are two-fold. It can achieve much higher reliability with the same storage. Also, it requires much lower storage for the same reliability [Weatherspoon and Kubiatowics 2001]. More recently, erasure coding is also making its way into planet-scale cloud storage systems, such as HDFS-RAID in Facebook [Borthakur et al. 2010] and Google Colossus [Fikes 2010].

Nevertheless, the existing systems do not explore alternative erasure coding design other than MDS codes [MacWilliams and Sloane 1977; Reed and Solomon 1960] or simple combination of different RAID modes [Chen et al. 1994]. Pyramid Codes are non-MDS, and allow much more flexible tradeoff between storage space and access efficiency than MDS codes.

2.2. Erasure Code Design

Besides pyramid codes, there are a few other non-MDS coding schemes, such as LDPC codes [Gallager 1963; Luby et al. 2001; Maymounkov and Mazieres 2003]; Weaver codes [Hafner 2005]; flat XOR codes [Greenan et al. 2008, 2010] and simple regenerating codes [Papailiopoulos et al. 2012], which improve repair and reconstruction read performance over MDS codes. LDPC codes [Gallager 1963; Luby et al. 2001; Maymounkov and Mazieres 2003] achieve comparable reliability as MDS codes when either the number of data fragments or parity fragments is relatively large (in the order of hundreds). Pyramid codes target much smaller coding groups with just a few redundant blocks. Weaver codes [Hafner 2005] are very efficient, but unfortunately require high storage overhead (2x and above). Pyramid codes target much lower storage overhead. The storage cost of both stepped combination codes [Greenan et al. 2010] and simple regenerating codes [Papailiopoulos et al. 2012] is below 2x, but still higher than than pyramid codes. In addition, none of the above codes achieves the recoverability limit (Section 4).

To improve reconstruction performance, pyramid codes in general read fewer blocks than a comparable MDS code. A promising alternative is instead to read from more blocks, but less data from each. Regenerating codes [Dimakis et al. 2010] explore this direction. Finding coding coefficients for regenerating codes is still an active and open research topic. The best general results known today require splitting blocks stored on individual nodes into an exponential number of pieces [Cadambe et al. 2011; Wang et al. 2011], and are *not* yet practical. On the other hand, the best practical codes known today are only available at limited options, such as at a storage overhead of 2x (and more) [Suh and Ramchandran 2011] or exactly 1.5x [Cadambe et al. 2011], whereas pyramid codes are flexible and can be constructed for arbitrary storage overhead.

Instead of searching for regenerating codes, other studies explore the same direction, but focusing on optimizing existing codes [Khan et al. 2011, 2012; Xiang et al. 2010]. These works modify only decoding algorithms and not the erasure codes themselves. The savings of these schemes are typically around 20%-30% [Khan et al. 2011, 2012; Xiang et al. 2010], much less than pyramid codes.

2.3. Computation Complexity

There is a plethora of studies focusing on minimizing the computational complexity of encoding and decoding operations. XOR-based array codes replace finite field computation with less computationally expensive XOR operations [Blaum and Roth 1999; Blaum et al. 1995; Blomer et al. 1995; Corbett et al. 2004; Huang and Xu 2008; Plank 2008; Xu and Bruck 1999; Xu et al. 1999]. Further optimizations investigate how to schedule the XOR operations so that the common ones are calculated only once [Huang and Xu 2003; Huang et al. 2007; Luo et al. 2009; Plank and Xu 2006]. The results from these studies can directly benefit pyramid codes.

3. BASIC PYRAMID CODES

In this section, we briefly review the erasure-resilient coding (ERC) scheme, focusing on systematic MDS codes. We define four key metrics concerning any ERC scheme. We illustrate basic pyramid codes (BPC) through examples and provide formal definitions. We show BPC offers a more flexible tradeoff among the key metrics than MDS codes, for example, reconstruction read cost can be reduced by 50% with merely 11% additional storage overhead.

3.1. Brief Primer on Erasure-Resilient Coding

Before presenting pyramid codes, let us briefly review the erasure-resilient coding (ERC) scheme. In particular, we focus on an ERC scheme that is *systematic* and *maximum distance separable* (MDS) [MacWilliams and Sloane 1977], attracting particular attention in distributed storage system design.

Let a distributed storage system host k equal-size *data blocks*. An (n, k) ERC scheme expands the k data blocks into n $(n \geq k)$ same-size *coded blocks* and distributes the coded blocks to n storage nodes.

An ERC scheme is systematic when the original k data blocks are preserved in the n coded blocks. Hence, in a systematic ERC scheme, the n coded blocks contain the original k data blocks and additional $m = n - k$ *redundant blocks*. Use $\mathbf{d}_i$ $(i = 1, \cdots, k)$ to denote the data blocks, and $\mathbf{c}_j$ $(j = 1, \cdots, m)$ to denote the redundant blocks.

An (n, k) ERC scheme is MDS if and only if all the original k data blocks can be obtained from any subset of coded blocks of size k. In the context of distributed storage, an (n, k) MDS ERC scheme implies no data loss even when there are up to $n-k$ storage node failures, or the system is resilient to arbitrary $n-k$ failures. For example, Figure 1 illustrates a (9, 6) systematic MDS code. The code consists of 6 data blocks and 3 redundant blocks. It tolerates up to 3 arbitrary failures.

The process of computing redundant blocks from data blocks is called *encoding*, and the process of computing failed data blocks from other data and redundant blocks is called *decoding* (or *recovery*). Most ERC schemes apply *linear* block codes, where the redundant blocks are *linear* combinations of the data blocks, which are presented algebraically as follows:

$$\mathbf{c}_j = \sum_{i=1}^{k} \alpha_{i,j} \mathbf{d}_i, \tag{1}$$

where $\alpha_{i,j}$s are coding coefficients in a finite field (or ring) [MacWilliams and Sloane 1977].

Many commonly used ERC schemes in storage systems are specific examples of the MDS codes. For example, the *simple parity* scheme, which is widely used in RAID-5 systems, computes the only redundant block as the binary sum (XOR) of all the data blocks. It is essentially a $(k + 1, k)$ MDS code. The replication scheme, which creates r replicas for each data block, is indeed $(r, 1)$ MDS code. Finally, Reed–Solomon codes [Reed and Solomon 1960] are one of the most widely used classes of linear MDS codes.

3.2. Key Metrics

There are four key metrics concerning any ERC scheme.

Storage Overhead. The storage overhead of an ERC scheme is computed as the ratio between all the blocks (data + redundant) and the data blocks, that is, n/k. For the (9, 6) MDS code, the storage overhead is $9/6 = 1.5$.

Fault Tolerance. We use a simple probability model to characterize the fault tolerance of an ERC scheme. Assume each (data or redundant) block can fail independently with probability p_b. When there are x failures, denote the recoverable ratio as $r_p(x)$. For the (9, 6) MDS code, up to three failures are fully recoverable, so $r_p(0) = \ldots = r_p(3) = 1.0$. On the other hand, four failures or more are not recoverable, so $r_p(x) = 0$ for $x \geq 4$. Enumerating through all possible failures, we characterize the fault tolerance of the code by *unrecoverable probability* (denoted as p_f), computed as following:

$$p_f = 1.0 - \sum_{x=0}^{n} r_p(x) \binom{n}{x} p_b^x (1 - p_b)^{(n-x)}. \tag{2}$$

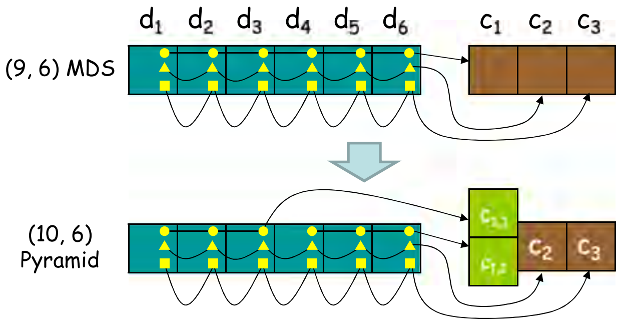


Fig. 1. Constructing a $(10, 6)$ basic pyramid code from a $(9, 6)$ MDS code.

Assuming $p_b = 0.01$, the unrecoverable probability of the (9, 6) MDS code is 1.2×10^{-6}.

More advanced Markov models can be applied to study mean time to data loss (MTTDL), which takes into account the failure rate, repair time, and even failure correlation. We refer interested readers to Xin et al. [2003] and Ford et al. [2010]. Alternatively, an elastic fault tolerance vector [Hafner and Rao 2006] can be computed without requiring failure rates or making the Markovian assumption.

Access Efficiency. Systematic ERC schemes preserve the original data blocks as part of the coded blocks. Hence, reading a data block can be directly served by the storage node hosting the block. However, if the storage node is unavailable, the read operation has to access the remaining blocks and recover the missing data block.

To characterize access efficiency, we define *reconstruction read cost*, denoted by $R(x)$, as the expected number of blocks required to serve an unavailable *data block*, when there are x failures. In the (9, 6) MDS code, despite the number of failures, it always requires six blocks to serve an unavailable data block. Therefore, $R(1) = R(2) = R(3) = 6$.

Update Complexity. When data blocks are updated, redundant blocks have to be updated correspondingly. For every data block update, the number of redundant block updates is typically referred to as *update complexity*. Update complexity is an important metric for storage systems with in-place updates. For the (9, 6) MDS code, whenever any data block is updated, all the three redundant blocks ($\mathbf{c}_1$, $\mathbf{c}_2$, and $\mathbf{c}_3$) have to be updated as well, so the update complexity is 3.

3.3. Basic Pyramid Codes: An Example

Now we use an example to describe BPC and demonstrate how they significantly improve reconstruction read performance. Our example constructs a (10, 6) BPC from the (9, 6) MDS code in Figure 1, which could be a Reed–Solomon code, or any other MDS codes (such as a STAR code [Huang and Xu 2008]).

We divide the six data blocks into two equal size groups $S_1 = \{\mathbf{d}_1, \mathbf{d}_2, \mathbf{d}_3\}$ and $S_2 = \{\mathbf{d}_4, \mathbf{d}_5, \mathbf{d}_6\}$. We then compute one redundant block for each group, denoted as $\mathbf{c}_{1,1}$ for S_1 and $\mathbf{c}_{1,2}$ for S_2, as follows.

$$\mathbf{c}_{1,1} = \sum_{i=1}^{3} \alpha_{i,1}\mathbf{d}_i; \quad \mathbf{c}_{1,2} = \sum_{i=4}^{6} \alpha_{i,1}\mathbf{d}_i. \qquad (3)$$

Since $\mathbf{c}_{1,1}$ and $\mathbf{c}_{1,2}$ are computed within their groups, we call them *group redundant blocks*. Note that we use the same $\alpha_{i,1}$s as in Eq. (1). Hence, it is easy to see that

$$\mathbf{c}_{1,1} + \mathbf{c}_{1,2} = \mathbf{c}_1,$$

that is, the sum of the two group redundant blocks equals to $\mathbf{c}_1$ (the first redundant block) in the MDS code. Alternatively, the group redundant blocks ($\mathbf{c}_{1,1}$ and $\mathbf{c}_{1,2}$, respectively) can be interpreted as the projection of the MDS redundant block ($\mathbf{c}_1$) onto each group (by nulling the data blocks in all other groups).

Next, we compute two additional redundant blocks ($\mathbf{c}_2$ and $\mathbf{c}_3$), exactly the same way as the MDS code. In contrast to the group redundant blocks, these two redundant blocks are computed from all the six data blocks, so we call them *global redundant blocks*. The code is illustrated in Figure 1.

3.4. Key Metrics of Basic Pyramid Codes

Now, we examine the key metrics of the (10, 6) BPC and compare them to the (9, 6) MDS code.

Access Efficiency. The BPC is superior in its reconstruction read cost. When a single data block (say $\mathbf{d}_1$) fails, reconstruction read of the failed block requires three other blocks – $\mathbf{d}_2$, $\mathbf{d}_3$, and $\mathbf{c}_{1,1}$. Hence, $R(1) = 3$, half that of the MDS code! When two blocks fail, if both failed blocks belong to different groups, reconstructing any failed block requires only three blocks. Only when both failed blocks belong to the same group, reconstruction requires six blocks. Enumerating through all the cases, $R(2) = 4$, much lower than six in the MDS code. Even when three blocks fail, $R(3)$ is only 4.75, still lower than six in the MDS code.

Storage Overhead. Compared to the MDS code, the BPC requires additional storage space. The storage overhead of the BPC is $10/6 = 1.67$, compared to $9/6 = 1.5$ of the MDS code. Hence, the improvement of access efficiency comes at the cost of extra storage overhead. That is, compared to the MDS code, the BPC reduces reconstruction read cost by 50% with merely 11% additional storage overhead. This example perfectly illustrates the *core* concept of pyramid codes – *trading storage space for access efficiency*.

Fault Tolerance. We now examine the fault tolerance of the (10, 6) BPC. We first show that the code can recover arbitrary three failures. Assume there are arbitrary three failures out of the total ten blocks, which can fall into one of the following two cases: (1) both $\mathbf{c}_{1,1}$ and $\mathbf{c}_{1,2}$ are available; or (2) at least one of them fails. In the first case, $\mathbf{c}_1$ can be computed from $\mathbf{c}_{1,1}$ and $\mathbf{c}_{1,2}$. Then, it becomes recovering three failures from the (9, 6) MDS code, which is straightforward to decode. In the second case, it is impossible to compute $\mathbf{c}_1$. However, other than $\mathbf{c}_{1,1}$ or $\mathbf{c}_{1,2}$, there are at most two block failures. Hence, from the perspective of the (9, 6) MDS code, there are at most three failures ($\mathbf{c}_1$ and those two failed blocks), and thus the case is also decodable.

The BPC also tolerates some four failures. With similar arguments, we can show that 73% of the four-failure cases are recoverable. The BPC employs four redudant blocks, but does *not* tolerate arbitrary 4 failures. Hence, it is *not* MDS.[1]

In summary, for the (10, 6) BPC, recoverable ratio $r_p(0) = ... = r_p(3) = 1.0$, $r_p(4) = 0.73$ and $r_p(x \geq 5) = 0$. Again, assuming failure probability $p_b = 0.01$, we obtain unrecoverable probability $p_f = 5.6 \times 10^{-7}$. Compared to $p_f = 1.2 \times 10^{-6}$ for the MDS code, the BPC provides higher fault tolerance. The comparison is shown in Figure 2.

Update Complexity. Whenever any data block is updated, the BPC has to update three redundant blocks (both $\mathbf{c}_2$, $\mathbf{c}_3$, plus either $\mathbf{c}_{1,1}$ or $\mathbf{c}_{1,2}$). Its update complexity is the same as that of the MDS code.

[1] A (10, 6) MDS code requires the same storage overhead, but can tolerate arbitrary four failures. However, its reconstruction requires six blocks and is more costly.

	reconstruction read cost			fault tolerance	storage	update
	single-failure	double-failure	triple-failure	(unrecoverable prob.)	overhead	complexity
(9, 6) MDS	6	6	6	1.2×10^{-6}	1.50	3
(10, 6) Pyramid	3	4	4.75	5.6×10^{-7}	1.67	3
savings	50%	33%	21%			

Fig. 2. (9, 6) MDS vs. (10, 6) BPC.

In conclusion, the BPC uses more storage space than the MDS code, but it gains in reconstruction read cost and fault tolerance, while maintaining the same update complexity.

3.5. Formal Definition of Basic Pyramid Codes

In general, a BPC can be constructed as follows. We start with a (n, k) MDS code, and separate the k data blocks into L disjoint groups (denoted as S_l, $l = 1, \cdots, L$), where group S_l contains k_l blocks (i.e., $|S_l| = k_l$).[2] Next, we keep m_1 out of the m redundant blocks unchanged. These are the global redundant blocks. We then computes $m_0 = m - m_1$ group redundant blocks for each group S_l. The jth redundant block for group S_l (denoted as $\mathbf{c}_{j,l}$) is simply a projection of the jth redundant block in the MDS code (i.e., $\mathbf{c}_j$) onto the group S_l. In other words, $\mathbf{c}_{j,l}$ is computed the same as $\mathbf{c}_j$ in the MDS code, by simply setting all groups other than S_l to 0. The combination of all $\mathbf{c}_{j,l}$s for the same l yields the redundant block $\mathbf{c}_j$ in the MDS code.

The resulting BPC is systematic. It contains k data blocks and $m_0 L + m_1$ redundant blocks, where there are m_0 group redundant blocks for each of the L groups and m_1 global redundant blocks. The code is presented as $(k + m_0 L + m_1, k)$, where $m_0 + m_1 = n - k$.

The BPC satisfies the following theorem.

THEOREM 1. *A $(k + m_0 L + m_1, k)$ BPC constructed from a (n, k) MDS code $(m_0 + m_1 = n - k)$ can recover arbitrary $m = m_0 + m_1$ erasures.*

PROOF. We consider a failure case with m erasures. Assuming r out of the m erasures are among the $m_0 L$ group redundant blocks, then the remaining $m - r$ erasures are among the k data and the m_1 global redundant blocks. We prove the theorem in the following two cases: (1) $r \geq m_0$; and (2) $r < m_0$.

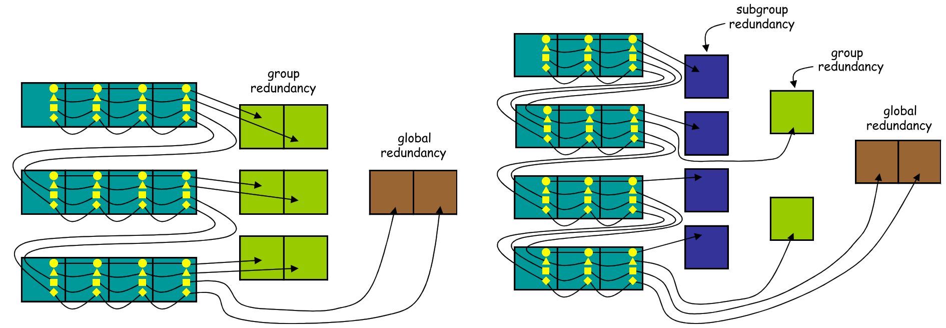
When $r \geq m_0$, simply drop all the group redundant blocks. The remaining k data and m_1 global redundant blocks form a $(k + m_1, k)$ MDS code. The MDS code tolerates up to m_1 erasures. The number of actual erasures is $m - r \leq m - m_0 = m_1$. Therefore, the case is recoverable.

Now, we consider the case when $r < m_0$. For each j, when all the group redundant blocks $\mathbf{c}_{j,l}$s (over all ls) are available, their XOR sum yields $\mathbf{c}_j$ (a redundant block in the MDS code). Denote m_0' as the number of XOR sums can be calculated. Then, we have $m_0' \geq m_0 - r > 0$. The m_0' XOR sums, together with the k data and the m_1 global redundant blocks, form a $(k + m_0' + m_1, k)$ MDS code. The number of erasures tolerated by the MDS code is up to $m_0' + m_1 \geq m_0 - r + m_1 = m - r$. The number of actual erasures is $m - r$. Therefore, the case is also recoverable. □

In addition, the BPC holds the following property.

COROLLARY 3.1. *In the $(k + m_0 L + m_1, k)$ BPC, each group S_l is a $(k_l + m_0, k_l)$ MDS code.*

[2] Groups can be of different size, so there is no need to enforce k divides L.



(a) two hierarchy - (12+2×3+2, 12)

(b) three hierarchy - (12+1×4+1×2+2, 12)

Fig. 3. Multi-hierarchical extension of basic pyramid codes. (Same storage overhead, different reconstruction read cost and fault tolerance).

PROOF. We prove the statement by contradiction. Assume group S_l, which is a $(k_l + m_0, k_l)$ code, is not MDS. Then, the group fails to recover a certain failure case with m_0 erasures. Now, further assume all the m_1 global redundant blocks are also failures. Lacking global redundant blocks, the m_0 erasures remain unrecoverable at the global level. Hence, the failure case with $m_0 + m_1$ erasures is unrecoverable in the BPC. This, however, contradicts with the above theorem. □

3.6. Decoding Basic Pyramid Codes

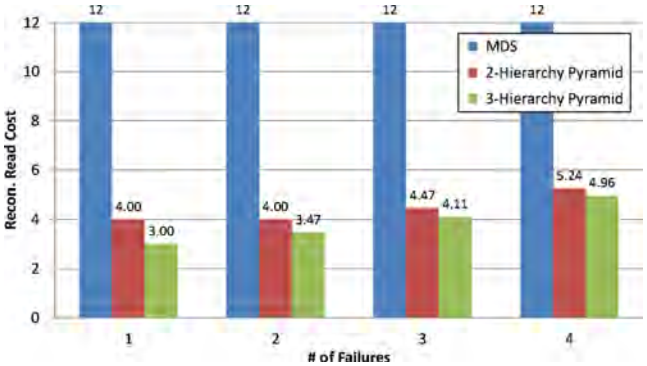
Decoding BPC is straightforward, and we briefly summarize as follows.

- *Step* 1. Start from the group level. For each group, if the available redundant blocks are no less than the failed data blocks, recover all the failed data blocks and mark all the blocks (both data and redundant) available. For failed redundant blocks, compute them only if they are used in the following step.
- *Step* 2. Move to the global level. For each j $(1 \leq j \leq m_0)$, if all the group redundant block $\mathbf{c}_{j,l}$s (over all ls) are marked as available, compute their XOR sum as $\mathbf{c}_j$ (a redundant block in the original MDS code), and add $\mathbf{c}_j$ as an additional global redundant block. On the global level, if the number of available redundant blocks equals or is more than the number of failed data blocks, recover all the failed blocks. Otherwise, remaining failed blocks are declared unrecoverable.

3.7. A Multi-Hierarchical Extension of Basic Pyramid Codes

Next, we illustrate how to extend BPCs to more than two hierarchies. First, we construct a $(12 + 2 \times 3 + 2, 12)$ BPC from a $(16, 12)$ MDS code, as shown in Figure 3(a). Again, the code contains two group redundant blocks for each of the three groups, in addition to two global redundant blocks. Next, we keep the same storage overhead and extend the BPC to multiple hierarchies. Figure 3(b) shows an example of a 3-hierarchy, where the data blocks are first divided into two groups and then further divided into four subgroups. For redundant blocks, there are global ones and group ones. In addition, there are subgroup redundant blocks. The particular example in Figure 3(b) contains one subgroup redundant block for each of the four subgroups, one group redundant block for each of the two groups, and two global redundant blocks. It can be denoted as a $(12 + 1 \times 4 + 1 \times 2 + 2, 12)$ BPC.

As a simple exercise, Theorem 1 and Corollary 3.1 can be readily extended to multiple hierarchies. Similarly, the decoding of multi-hierarchy BPCs will start with the



(a) Reconstruction read cost

	reconstruction read cost				fault tolerance	storage
	1-failure	2-failure	3-failure	4-failure	(unrecoverable prob.)	overhead
(16, 12) MDS	12	12	12	12	4.0×10^{-7}	1.33
2-Hierarchy Pyramid	4	4	4.47	5.24	1.7×10^{-8}	1.67
3-Hierarchy Pyramid	3	3.47	4.11	4.96	3.0×10^{-8}	1.67

(b) Read cost, fault tolerance vs. storage overhead

Fig. 4. MDS code vs. multi-hierarchy BPCs.

lowest level and gradually move to the global level. This is analogous to climbing up a pyramid, just as the name of the codes suggests.

Now, we compare the two BPCs to the (16, 12) MDS code. The comparison of reconstruction read cost is presented in Figure 4(a), while that of fault tolerance and storage overhead in Figure 4(b). We conclude that both BPCs reduce reconstruction read cost significantly over the MDS code, across all failure cases. Between the two BPCs, while the storage overhead is the same, one achieves slightly lower reconstruction read cost at the cost of slightly worse fault tolerance (or higher unrecoverable probability). Hence, it is important to note that, besides offering more flexibility than MDS codes in trading storage space for access efficiency, BPC also allows additional level of flexibility – with fixed storage overhead, it is possible to vary BPC and further balance between access efficiency and fault tolerance.[3]

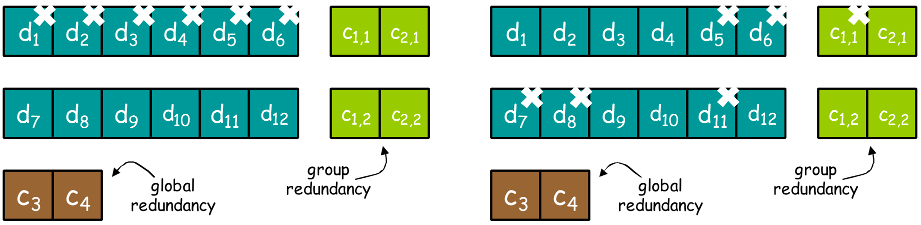
4. RECOVERABILITY LIMIT

A BPC achieves flexible tradeoff among the four key metrics: storage overhead, fault tolerance, access efficiency, and update complexity. In this section we show that it can be further improved. In particular, it is possible to improve one key metric – fault tolerance – without affecting the remaining three metrics.

4.1. Motivating Example

We start with a motivating example. Consider a (12 + 2 × 2 + 2, 12) BPC, as shown in Figure 5. The code is constructed from a (16, 12) MDS code. It contains two groups, each with two group redundant blocks. In addition, it contains two global redundant blocks. Here, $m_0 = m_1 = 2$ (and $L = 2$), so we know that the code tolerates *arbitrary* $m = m_0 + m_1 = 4$ failures. Since there are six redundant blocks in total, the code also

[3] Note that the probabilities will be very different between 1-failure and 4-failure. Hence, when using pyramid codes in practical systems, the expectation of reconstruction read cost should be calculated by taking into account the probability difference across the failure cases.



(a) unrecoverable at all

(b) unrecoverable as BPC

Fig. 5. Motivating example for recoverability condition. (Six failures each, marked by “x”).

tolerates some cases with five or six failures. The focus of this section is to establish the limit of recoverability for cases beyond four failures.

The BPC code is defined by the following set of linear equations:

$$\mathbf{c}_{j,1} = \sum_{i=1}^{6} \alpha_{i,j} \mathbf{d}_i, \quad j = 1, 2, \tag{4}$$

$$\mathbf{c}_{j,2} = \sum_{i=7}^{12} \alpha_{i,j} \mathbf{d}_i, \quad j = 1, 2, \tag{5}$$

$$\mathbf{c}_{j} = \sum_{i=1}^{12} \alpha_{i,j} \mathbf{d}_i \quad j = 3, 4, \tag{6}$$

where the coding coefficients $\alpha_{i,j}$s are from the MDS code. To check whether a failure case is recoverable, we simply treat failure blocks as *unknowns* in the linear equations above. A failure case is recoverable *if and only if* the linear equations are solvable.[4]

Now, we apply the standard method on the two failure cases, each with six failures, shown in Figure 5. The coding equations in neither failure case in Figure 5 are solvable. Hence, we conclude that neither failure case is recoverable.

For the first failure case in Figure 5(a), all the six data blocks in the second group are available, so the two redundant blocks in the group are *useless* because they can be computed anyway. Hence, those two redundant blocks should be removed from decoding. Effectively, we are left with six unknowns and only four coding equations, which is impossible to solve.

For the second failure case in Figure 5(b), the linear equations are unsolvable, so the failures definitely cannot be recovered. However, is this the limit? None of the redundant blocks appears useless. Effectively, there are six unknowns and six coding equations. Next, we establish the limit of recoverability and show that this failure case can in fact become recoverable when the coding equations are modified.

4.2. Coding Dependency and Coding Equations

Before describing the limit of recoverability, we first clarify two concepts that uniquely define an ERC scheme: *coding dependency* and *coding equations*.

[4] Gaussian elimination is a standard method to check whether a set of linear equations is solvable. Alternatively, more efficient matrix methods [Hosekote et al. 2005, 2007] can also be applied.

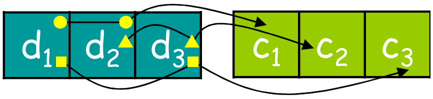


Fig. 6. A simple (6, 3) code.

Coding dependency defines which set of data blocks is used to compute each individual redundant block. Let $\mathbf{P}(\mathbf{c}_j)$ denote a bit vector of length k, where $\mathbf{P}(\mathbf{c}_j)[i] = 1$ means data block $\mathbf{d}_i$ is used to compute redundant block $\mathbf{c}_j$ and $\mathbf{P}(\mathbf{c}_j)[i] = 0$ means otherwise. Then, the union of all the sets, that is, $\cup_j \mathbf{P}(\mathbf{c}_j)$, completely defines the coding dependency of the ERC scheme.

In addition to coding dependency, coding equations are required to uniquely define the ERC scheme. Besides defining *which* set of data blocks, coding equations also define *how* the set of data blocks compute each redundant block, or the coding coefficients for computing the redundant block.

Among the four key metrics of the ERC scheme, coding dependency determines three of them: storage overhead, reconstruction read cost, and update complexity. Coding equations only affect the forth metric: fault tolerance.

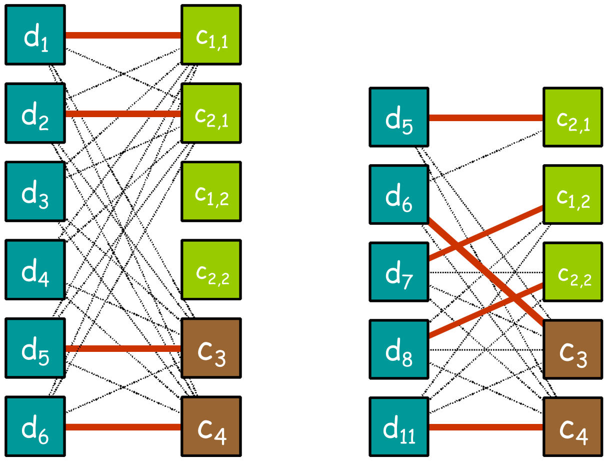
We use the (6, 3) code in Figure 6 as an example to illustrate these two concepts. The code contains three data blocks ($\mathbf{d}_1$, $\mathbf{d}_2$, and $\mathbf{d}_3$) and three redundant blocks ($\mathbf{c}_1$, $\mathbf{c}_2$, and $\mathbf{c}_3$). Its coding dependency is defined by $\cup_{j=1}^{3} \mathbf{P}(\mathbf{c}_j)$, where $\mathbf{P}(\mathbf{c}_1) = [1, 1, 0]$, $\mathbf{P}(\mathbf{c}_2) = [0, 1, 1]$, and $\mathbf{P}(\mathbf{c}_3) = [1, 0, 1]$. Without knowing the coding equations, we can already determine three key metrics: (*i*) storage overhead $6/3 = 2$; (*ii*) reconstruction read cost $R(1) = R(2) = R(3) = 2$ (reconstructing any unavailable data block requires two other blocks); and (*iii*) update complexity is two (updating any data block affects two redundant blocks).

The code is completely defined with coding equations. Assume $\mathbf{c}_1 = \mathbf{d}_1 + \mathbf{d}_2$, $\mathbf{c}_2 = \mathbf{d}_2 + \mathbf{d}_3$, and $\mathbf{c}_3 = \mathbf{d}_1 + \mathbf{d}_3$. To calculate fault tolerance, we compute recoverable ratio $r_p(x)$ (recall x is the number of failures). It is easy to verify that the code tolerates up to two arbitrary failures, so $r_p(0) = r_p(1) = r_p(2) = 1.0$. For three failures, the interesting case is when all the three data blocks fail, but the three redundant blocks are available. It turns out the three linear coding equations are dependent and thus unsolvable, so the case is unrecoverable. Considering all failure cases, we obtain $r_p(3) = 0.8$. Therefore, the unrecoverable probability is $p_f = 4.0 \times 10^{-6}$.

However, the fault tolerance can be improved by modifying the coding equations. If we make $\mathbf{c}_3 = \mathbf{d}_1 + 2\mathbf{d}_3$,[5] then the case with three data block failures becomes recoverable. Now, the recoverable ratio increases to $r_p(3) = 0.85$ and the unrecoverable probability reduces to $p_f = 3.1 \times 10^{-6}$. Note that this modification does *not* affect coding dependency, so the storage overhead, access efficiency, and update complexity remain the same.

This example illustrates that it is possible to improve fault tolerance without affecting storage overhead, access efficiency, and update complexity. It is achieved through modifying only coding equations while keeping coding dependency untouched. Now, a fundamental question arises—given predetermined coding dependency, what is the limit of recoverability?

[5] Unless noted otherwise, all additions and multiplications are defined in finite field GF(2^8). The field is generated using $x^8 + x^4 + x^3 + x^2 + 1$ as the prime polynomial [MacWilliams and Sloane 1977]. All field elements are represented using integer notation [Blahut 2003, Ch. 4]. For example, element 2 in integer notation is equivalent to x in polynomial notation.



(a) unrecoverable case (full-size matching does *not* exist)

(b) recoverable case (full-size matching exists)

Fig. 7. Tanner graphs (bold edges show maximum matchings).

4.3. Recoverability Limit

To answer this fundamental question, we establish a necessary condition, called the *matching condition*, to characterize the limit of recoverability for any failure case. The condition is *necessary*, and therefore must be satisfied if a case is recoverable. However, the condition is *not sufficient*. That is, even when the condition is satisfied, the case could still be unrecoverable.

4.3.1. Maximally Recoverable Property. For any failure case, whether the matching condition is satisfied can be determined based solely on coding dependency. That is, it is not affected by coding equations. For instance, the failure case in Figure 6 always satisfies the matching condition, regardless of coding equations. On the other hand, when the matching condition is satisfied, it is possible to modify coding equations such that an unrecoverable case becomes recoverable. Conceivably, it is desirable to design coding equations such that all failures cases satisfying the matching condition are recoverable. Formally, this is defined as a *maximally recoverable* property -

Definition 4.1. An erasure-resilient coding scheme is said to hold the maximally recoverable (MR) property under predetermined coding dependency, if all failure cases satisfying the matching condition are recoverable.

For the unmodified (6, 3) code, the failure case in Figure 6 satisfies the matching condition but is unrecoverable. Hence, the code does *not* hold the MR property. The modified coding equations result in a new code. Once we formally define the matching condition, it can be readily verified that all failure cases satisfying the condition are indeed recoverable. Therefore, the new code *does* hold the MR property.

4.3.2. Matching Condition. Coding dependency can be represented using the Tanner graph [Tanner 1981]. A Tanner graph is a bipartite graph, where nodes on the left part of the graph represent data blocks (*data nodes* hereafter), and nodes on the right represent redundant blocks (*redundant nodes*). An edge is drawn between a data node and a redundant node, if the computation of the redundant block uses the data block.

The matching condition can be then represented using *reduced decoding Tanner graph*. Given a failure case, a reduced decoding Tanner graph (denoted as T) is derived by removing all available data blocks and failed redundant blocks from the Tanner graph. For instance, the reduced decoding Tanner graphs corresponding to the failure cases in Figure 5 are shown in Figure 7.

Furthermore, we define *matching* (denoted by M) as a set of edges in the reduced decoding Tanner graph, where no two edges connect at the same node. The size of the matching $|M|$ equals to the number of edges in the set. Define *maximum matching* (denoted by M_m) as a matching with the maximum number of edges. Also, if $|M_m|$ equals the number of data nodes, such a matching is called a *full-size matching* (denoted by M_f). For example, the reduced decoding Tanner graph in Figure 7(b) contains a full-size matching, while the one in Figure 7(a) does *not*.

With these definitions, the matching condition for recoverability is stated in the following theorem. (Note that when there is no ambiguity, blocks and nodes are used interchangeably, as the recovery of a failure case and the recover of a reduced decoding Tanner graph.)

THEOREM 2. *For any ERC scheme, a failure case is said to satisfy the matching condition whenever the corresponding reduced decoding Tanner graph contains a full-size matching. A failure case is recoverable* only when *it satisfies the matching condition.*

PROOF. We prove this theorem by contradiction. Examine an arbitrary *recoverable* failure case, where the reduced decoding Tanner graph T consists of r_d data nodes and r_c redundant nodes. (Again, this means the failure case has r_d failed data blocks and r_c available redundant blocks.) Obviously, $r_d \le r_c$. Now assume T does *not* contain a full-size matching. Then, the size of its maximum matching M_m is less than r_d, that is, $|M_m| < r_d$. Based on the König–Egerváry theorem [Schrijver 2003] in graph theory, in a bipartite graph, the maximum size of a matching is equal to the minimum size of a node cover. Hence, a *minimum node cover* (denoted by N_c), which contains a minimum set of nodes covering all edges in T, has $|M_m|$ nodes, that is, $|N_c| = |M_m|$. Let n_d be the number of data nodes in N_c, then $|M_m| - n_d$ is the number of redundant nodes in N_c. It is clear that $n_d \le |M_m| < r_d$.

Now let us assume all the data blocks in N_c are somehow known (not failures any more), then we can deduce a new failure case with fewer failed blocks, which corresponds to a new Tanner graph T'. Any redundant node that is not in N_c can be removed from T' because those redundant nodes can only connect to the data nodes in N_c (otherwise there will be edges in T not covered by N_c), and thus isolated in T'. Hence, there are at most $|M_m| - n_d$ redundant nodes left in T'. On the other hand, there are still $r_d - n_d$ (positive value) data nodes left. As $|M_m| - n_d < r_d - n_d$, there are fewer redundant nodes than the data nodes, and thus T' is not recoverable. Therefore, T should not be recoverable either, which contradicts with the assumption. □

We emphasize that evaluating recoverability using the matching condition is more general than decoding pure XOR-based codes, such as LDPC codes [Gallager 1963; Luby et al. 2001; Maymounkov and Mazieres 2003] or flat XOR codes [Greenan et al. 2008], using Tanner graphs. Again, recall the (6, 3) code in Figure 6, where the case with three data block failures satisfies the matching condition, but is unrecoverable as a pure XOR-based code.

Now, we revisit the two failure cases of the BPC in Figure 5. For the first case, the maximum matching contains four edges, as shown in Figure 7(a). It is not a full-size matching. Therefore, the case does *not* satisfy the matching condition and is definitely unrecoverable. For the second case, the maximum matching contains six edges, as shown in Figure 7(b). It is indeed a full-size matching. Therefore, the case satisfies the matching condition.

Because the case is unrecoverable (the corresponding coding equations unsolvable), we conclude that BPC does *not* hold the maximally recoverable property. Next, we will show how the property can be satisfied by modifying coding equations, while keeping coding dependency untouched. The resulting codes are no longer BPC. They belong to a new class of codes, called *generalized pyramid codes* (GPC). We show that GPC holds the maximally recoverable property, that is, all failure cases satisfying the matching condition now become recoverable.

5. GENERALIZED PYRAMID CODES

In this section we describe *generalized pyramid codes* (GPC), which are *not* trivial extensions of BPC, but rather a new class of ERC schemes. GPC holds the maximally recoverable property. To the best of our knowledge, GPC is the first class of *non-MDS* codes holding such a property.[6] Pure XOR-based non-MDS codes, such as LDPC codes [Gallager 1963; Luby et al. 2001; Maymounkov and Mazieres 2003]; Weaver codes [Hafner 2005] and flat XOR codes [Greenan et al. 2008, 2010], subject to the same issue as the (6, 3) code in Figure 6, and do *not* hold the MR property.

Moreover, compared to BPC, GPC also accommodates more flexible coding dependency. While BPC enforces nested and non-overlapping groups, GPC accommodates arbitrary coding dependency. For the (6, 3) code in Figure 6, the overlapping coding dependency is not allowed in BPC, but perfectly acceptable in GPC.

Despite of the distinctions, we decide to categorize both classes of codes with a common name, *pyramid codes*, as they both aim at the same goal of trading storage space for access efficiency, and also follow the same concept of *climbing up a pyramid* during failure recovery.

An ERC scheme is completely defined by coding dependency and coding equations. Since GPC accommodates arbitrary coding dependency, the key to GPC lies in construct appropriate coding equations to hold the maximally recoverable property. In this section we provide a systematic algorithm for such construction.

5.1. Matrix Representation of ERC

We first describe a matrix representation of systematic ERC. Let $\mathbf{D}$ represent the data blocks $[\mathbf{d}_1, \mathbf{d}_2, \cdots, \mathbf{d}_k]^T$ and $\mathbf{C}$ represent all the blocks (data + redundant) $[\mathbf{c}_1, \mathbf{c}_2, \cdots, \mathbf{c}_{k+m}]^T$. Then, the matrix representation of ERC is $\mathbf{C} = \mathbf{G} \times \mathbf{D}$, where $\mathbf{G}$ is a $n \times k$ *generator matrix*. Let $\mathbf{g}_j$ denote the jth row vector in $\mathbf{G}$. For systematic ERC, $\mathbf{g}_j$s $(1 \leq j \leq k)$ are unit vectors from a $k \times k$ identify matrix, and thus $\mathbf{c}_j = \mathbf{d}_j$ $(1 \leq j \leq k)$. Moreover, $\mathbf{g}_j$s $(k < j \leq n)$ consists of coding coefficients and the ith entry $g_{j,i} = \alpha_{i,j-k}$, where αs are defined in Eq. (1).

When blocks fail (the failed blocks are denoted as $\mathbf{c}_f$s, where $1 \leq f \leq k+m$), we set row vector $\mathbf{c}_f$s to zero in $\mathbf{C}$ (make it $\mathbf{C}_s$) and similarly row vectors $\mathbf{g}_f$s to zero in $\mathbf{G}$ (make it $\mathbf{G}_s$ and call it *generator submatrix*). Now, we have

$$\mathbf{C}_s = \mathbf{G}_s \times \mathbf{D}. \tag{7}$$

To this end, both $\mathbf{C}_s$ and $\mathbf{G}_s$ are known, while $\mathbf{D}$ contains unknown vectors ($\mathbf{d}_f$s, $1 \leq f \leq k$). The failed data blocks can be recovered if and only if Eq. (7) is solvable (see Plank [1997] for a more detailed tutorial).

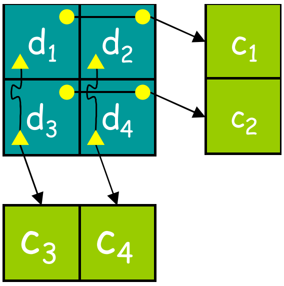


Fig. 8. Generalized pyramid code example.

5.2. Constructing Generalized Pyramid Codes: An Example

We now walk through an example to illustrate the construction of GPC. To complete the construction, we need to define both coding dependency and coding equations.

We consider the coding dependency defined in Figure 8, where there are four data blocks and four redundant blocks. The four data blocks are divided into four groups and each group computes one redundant block from only two data blocks. Clearly, coding dependency determines which entries in the generator matrix are *non-zero*. Hence, for this particular code, we have

$$\mathbf{G} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ g_{5,1} & g_{5,2} & 0 & 0 \\ 0 & 0 & g_{6,3} & g_{6,4} \\ g_{7,1} & 0 & g_{7,3} & 0 \\ 0 & g_{8,2} & 0 & g_{8,4} \end{bmatrix}. \tag{8}$$

The rest of the construction is to determine appropriate values of the non-zero entries in $\mathbf{G}$, such that the maximally recoverable property holds.

The construction takes an iterative approach. First, we determine the coding coefficients for $\mathbf{c}_1$, such that the $(5, 4)$ subcode formed by the four data blocks and $\mathbf{c}_1$ holds the maximally recoverable property. Next, we determine the coding coefficients for $\mathbf{c}_2$, without changing those for $\mathbf{c}_1$. Again, we ensure the $(6, 4)$ subcode still holds the MR property. Then, we determine the coding coefficients for $\mathbf{c}_3$, and so on, while always ensuring the MR property. In matrix terms, this is equivalent to starting with an identity matrix and expanding the generator matrix $\mathbf{G}$ one row vector in each iteration, while ensuring the MR property.

5.2.1. Expanding Generator Matrix. Now we elaborate on how to expand the generator matrix while ensuring the MR property. Let $\mathbf{G}^{j-1}$ and $\mathbf{G}^{j}$, respectively, denote the generator matrix before and after adding $\mathbf{g}_{k+j}$, the coding vector for $\mathbf{c}_j$. Hence, the size of $\mathbf{G}^{j-1}$ is $(k + j - 1) \times k$ and that of $\mathbf{G}^{j}$ is $(k + j) \times k$. Also, $\mathbf{G}^{j^T} = [\mathbf{G}^{{j-1}^T}\ \mathbf{g}_{k+j}{}^T]$.

To ensure the MR property after adding $\mathbf{g}_{k+j}$, we need to guarantee that any submatrix of the generator matrix (of size $k \times k$) is nonsingular. For generator submatrices, not including $\mathbf{g}_{k+j}$, this should always be true, since the MR property is already

[6] MDS codes hold the maximally recoverable property by nature.

ensured when constructing $\mathbf{G}^{j-1}$ in earlier iterations. Therefore, during the iteration for adding $\mathbf{g}_{k+j}$, we simply need to guarantee any generator submatrix, including $\mathbf{g}_{k+j}$ nonsingular. Denote such a generator submatrix as $\mathbf{G}_s^j$. Then, the submatrix needs to be full-rank, or $\mathrm{rank}(\mathbf{G}_s^j) = k$.[7]

In addition, we consider the $(k-1)\times k$ submatrix, obtained by removing $\mathbf{g}_{k+j}$ from $\mathbf{G}_s^j$. Denote the submatrix as $\mathbf{G}_s^j\backslash\mathbf{g}_{k+j}$. Clearly, a full-rank $\mathbf{G}_s^j$ implies that $\mathrm{rank}(\mathbf{G}_s^j\backslash\mathbf{g}_{k+j}) = k-1$. Also, let $\mathrm{null}(\mathbf{G})$ denote the *transpose* of $\mathbf{G}$'s *nullspace*. Then, $\mathrm{rank}(\mathbf{G}_s^j\backslash\mathbf{g}_{k+j}) = k-1$ dictates that $\mathrm{null}(\mathbf{G}_s^j\backslash\mathbf{g}_{k+j})$ is a single row vector. Finally, a full-rank $\mathbf{G}_s^j$ requires $\mathbf{g}_{k+j}$ *not* orthogonal to the nullspace row vector. Therefore, the dot product of the two should be nonzero, or $\mathrm{null}(\mathbf{G}_s^j\backslash\mathbf{g}_{k+j})\cdot\mathbf{g}_{k+j}\neq 0$.

In summary, to ensure the MR property, we need to choose $\mathbf{g}_{k+j}$ such that for any generator submatrix $\mathbf{G}_s^j$ where $\mathrm{rank}(\mathbf{G}_s^j\backslash\mathbf{g}_{k+j}) = k-1$, $\mathbf{g}_{k+j}$ satisfies $\mathrm{null}(\mathbf{G}_s^j\backslash\mathbf{g}_{k+j})\cdot\mathbf{g}_{k+j}\neq 0$. Since the nullspace of each generator submatrix is a single row vector, we can store all the nullspace vectors in an auxiliary matrix, denoted as $\mathbf{U}$, to facilitate the search of $\mathbf{g}_{k+j}$. To that end, $\mathbf{g}_{k+j}$ is accepted as long as it is not orthogonal to any row vector (denoted as $\mathbf{u}$) in $\mathbf{U}$, or $\mathbf{u}\cdot\mathbf{g}_{k+j}\neq 0$.

Now, we walk through the example of adding $\mathbf{g}_5$, the coding coefficients for $\mathbf{c}_1$. Here, a generator submatrices $\mathbf{G}_s^1$ is of size 4×4 and consists of three vectors from $\mathbf{g}_1$ to $\mathbf{g}_4$ and $\mathbf{g}_5$. Therefore, submatrix $\mathbf{G}_s^1\backslash\mathbf{g}_5$ consists of three vectors from $\mathbf{g}_1$ to $\mathbf{g}_4$. We calculate the rank of each submatrix. For all the submatrices with rank 3, we compute the nullspace and store the nullspace vector in $\mathbf{U}$. There are $\binom{4}{3}$ such submatrices and we obtain the following nullspace matrix

$$\mathbf{U} = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}. \tag{9}$$

We iterate through each vector $\mathbf{u}$ in the nullspace matrix $\mathbf{U}$ to determine $\mathbf{g}_5$ such that $\mathbf{u}\cdot\mathbf{g}_{k+j}\neq 0$. Note that $\mathbf{g}_5$ contains only two nonzero entries: $g_{5,1}$ and $g_{5,2}$. Hence, $\mathbf{u}_1\cdot\mathbf{g}_5\equiv 0$ and $\mathbf{u}_2\cdot\mathbf{g}_5\equiv 0$. Taking $\mathbf{u}_1$ as an example, $\mathbf{u}_1\cdot\mathbf{g}_5\equiv 0$ implies that it is impossible to recover d_4 from d_1, d_2, d_3, and c_1. This can be verified from the coding dependency in Figure 8. $\mathbf{u}_i\cdot\mathbf{g}_{k+j}\equiv 0$ is equivalent to a case that is impossible to recover. Therefore, we simply skip such $\mathbf{u}_i$s when choosing $\mathbf{g}_{k+j}$. Using the algorithm, which we will describe next, we choose $g_{5,1} = 1$ and $g_{5,2} = 142$.[8]

5.2.2. An Algorithm for Choosing $\mathbf{g}_m$. Given $\mathbf{U}$, we now describe a general algorithm to choose $\mathbf{g}_m$, such that $\forall\ \mathbf{u}\in\mathbf{U}$, $\mathbf{u}\cdot\mathbf{g}_m\neq 0$. The algorithm starts with random values in nonzero entries. It checks the dot product of $\mathbf{u}_1$ and $\mathbf{g}_m$. If $\mathbf{u}_1\cdot\mathbf{g}_m\neq 0$, then keep $\mathbf{g}_m$ and move on to $\mathbf{u}_2$. The process continues until it encounters the first vector $\mathbf{u}_j$, which satisfies $\mathbf{u}_j\cdot\mathbf{g}_m = 0$. As discussed before, if $\mathbf{u}_j\cdot\mathbf{g}_m\equiv 0$ (i.e., nonzero entries of $\mathbf{g}_m$ always correspond to zero entries of $\mathbf{u}_j$), $\mathbf{u}_j$ is simply skipped. Otherwise, the algorithm *augments* $\mathbf{g}_m$ (make it $\mathbf{g}'_m$) such that the following two conditions are satisfied: (1) $\mathbf{u}_j\cdot\mathbf{g}'_m\neq 0$; and (2) all previous $\mathbf{u}$'s are *not* affected, that is, $\mathbf{u}_i\cdot\mathbf{g}'_m\neq 0$ $(i<j)$ still hold.

[7] Note that there are failure cases that don't satisfy the matching condition, and are thus not recoverable. The generator submatrices for these cases are always singular despite $\mathbf{g}_{k+j}$. We skip these submatrices.

[8] Again, we use integer notation for the elements in the finite field $GF(2^8)$ [Blahut 2003, Ch. 4].

```
$\mathbf{G} := I_{k\times k}, \mathbf{U} := I_{k\times k}$
for $m = k+1 : n$ do
  // $\mathbf{g}_m^{zero}$: boolean array marking constant zero entries
  // $t$: index of the $t^{th}$ entry in $\mathbf{g}_m$
  for $t = 1 : k$ do
    $\mathbf{g}_m[t]$ := random value in the field
    if $\mathbf{g}_m^{zero}[t] = true$ then
      $\mathbf{g}_m[t] := 0$
  $\mathbf{ug}$ := null // $\mathbf{ug}$: all dot products of $\mathbf{u}_i \cdot \mathbf{g}_m$
  for $j = 1 : |\mathbf{U}|, \mathbf{u}_j \in \mathbf{U}$ do
    if $\mathbf{u}_j \cdot \mathbf{g}_m \neq 0$ then
      $\mathbf{ug}[j] := \mathbf{u}_j \cdot \mathbf{g}_m$, repeat
    if $\mathbf{u}_j \cdot \mathbf{g}_m \equiv 0$ then
      $\mathbf{ug}[j] := 0$, repeat
    // $\mathcal{E}_{bad}$: all bad $\epsilon_i$'s, $\mathbf{uu}$: all dot products of $\mathbf{u}_i \cdot \mathbf{u}_j$
    $\mathcal{E}_{bad}$ := null, $\mathbf{uu}$ := null
    for $i = 1 : j-1$ do
      $\mathbf{uu}[i] := \mathbf{u}_i \cdot \mathbf{u}_j$
      if $\mathbf{uu}[i] = 0$ then
        repeat
      $\mathcal{E}_{bad} := \mathcal{E}_{bad} + \{\mathbf{ug}[i]/\mathbf{uu}[i]\}$
    $\epsilon$ = random value out of $\mathcal{E}_{bad}$
    // argument $\mathbf{g}_m$ and update $\mathbf{ug}$
    $\mathbf{g}_m := \mathbf{g}_m + \epsilon\mathbf{u}_j$
    for $i = 1 : j$ do
      $\mathbf{ug}[i] := \mathbf{ug}[i] + \epsilon\mathbf{uu}[i]$
    for $t = 1 : k, \mathbf{g}_m^{zero}[t] = true$ do
      $\mathbf{g}_m[t] := 0$
    $\mathbf{ug}[j] := \mathbf{u}_j \cdot \mathbf{g}_m$
    // update $\mathbf{U}$ and add $\mathbf{g}_m$ to $\mathbf{G}$
    for $S' = \{$k-2 rows in $\mathbf{G}\}$ do
      $S = S' + \{\mathbf{g}_m\}$
      if rank$(S) = k-1$ then
        $\mathbf{u}$ := null space vector of $S$
        $\mathbf{U} := \mathbf{U} + \{\mathbf{u}\}$
    $\mathbf{G} := \mathbf{G} + \{\mathbf{g}_m\}$
return
```

Fig. 9. Constructing generalized pyramid codes.

The first condition can be satisfied by setting $\mathbf{g}'_m = \mathbf{g}_m + \epsilon\mathbf{u}_j$ ($\epsilon \neq 0$), as any nonzero ϵ satisfies

$$\mathbf{u}_j \cdot \mathbf{g}'_m = \mathbf{u}_j \cdot (\mathbf{g}_m + \epsilon\mathbf{u}_j) = \epsilon\mathbf{u}_j \cdot \mathbf{u}_j \neq 0.$$

Now we simply need to find an ϵ such that the second condition is also satisfied. Formally, this involves finding an ϵ such that

$$\forall\, \mathbf{u}_i\ (1 \leq i < j),\ \mathbf{u}_i \cdot \mathbf{g}'_m \neq 0. \tag{10}$$

We compute all ϵs that violate Eq. (10) (call them *bad* ϵs) and construct a set to hold them (denote as $\mathcal{E}_{bad}$). As long as we pick an ϵ out of the set $\mathcal{E}_{bad}$, it is guaranteed to

	reconstruction read cost			fault tolerance	storage
	single-failure	double-failure	triple-failure	(unrecoverable prob.)	overhead
(9, 6) MDS	6	6	6	$\mathbf{1.2 \times 10^{-6}}$	1.50
(10, 6) Basic Pyramid	3	4	4.75	$\mathbf{5.6 \times 10^{-7}}$	1.67
(10, 6) Generalized Pyramid	3	4	4.75	$\mathbf{3.1 \times 10^{-7}}$	1.67
savings	50%	33%	21%		

Fig. 10. MDS code vs. pyramid codes. (The unrecoverable probability of GPC is 45% lower than that of BPC.)

satisfy the second condition. To construct $\mathcal{E}_{bad}$, simply compute all the bad ϵ_is ($1 \leq i < j$), where each ϵ_i satisfies $\mathbf{u}_i \cdot \mathbf{g}'_m = 0$ such that

$$\epsilon_i = \frac{\mathbf{u}_i \cdot \mathbf{g}_m}{\mathbf{u}_i \cdot \mathbf{u}_j}.$$

To this end, as long as the number of bad ϵ_is in $\mathcal{E}_{bad}$ (i.e., $|\mathcal{E}_{bad}|$) is less than the number of symbols in the finite field, a desirable ϵ is guaranteed to be found. In the worst case, all bad ϵ_is happen to be unique during the final round (i.e., finding $\mathbf{g}_n$), then $|\mathcal{E}_{bad}| = \binom{n}{k-1}$ (the number of vectors in $\mathbf{U}$). Still, as long as the field size is greater than $\binom{n}{k-1}$, the construction of GPC is guaranteed to succeed.

5.2.3. Construction Algorithm Summary. Using the simple example, we have described all the details in the construction of GPC. Here, we briefly summarize the entire procedure (refer to Figure 9 for complete details).

— *Step* 1. Start with a $k \times k$ identity matrix $\mathbf{G} = \mathbf{I}_{k\times k}$ and construct an empty null space matrix $\mathbf{U}$.
— *Step* 2. Update $\mathbf{U}$. Enumerate through all submatrices $\mathbf{S}$ formed by any $k - 1$ rows from $\mathbf{G}$. If the rank of $\mathbf{S}$ is $k - 1$, compute its null space vector and append its transpose to $\mathbf{U}$. Otherwise, skip $\mathbf{S}$.
— *Step* 3. Find a $\mathbf{g}_m$ such that $\forall\, \mathbf{u} \in \mathbf{U}$, $\mathbf{u} \cdot \mathbf{g}_m \neq 0$, which adopts the previously described algorithm. Update $\mathbf{G}$ by adding $\mathbf{g}_m$ to it.
— *Step* 4. Repeat Step 2 and 3 until the entire generator matrix $\mathbf{G}$ is completed.

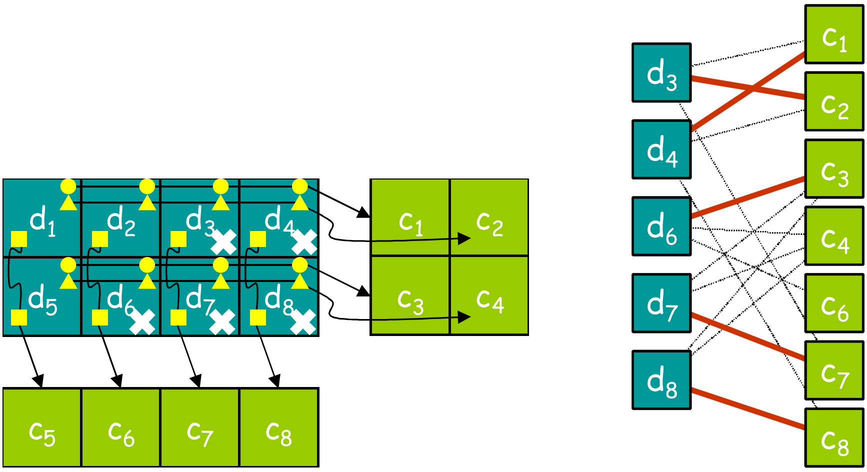
5.3. Maximally Recoverable Property

THEOREM 3. *GPC holds the maximally recoverable property. (see proof in Appendix* I*)*

5.4. Comparison with Basic Pyramid Codes

Now, we revisit the very first example (Figure 1) to illustrate the difference between BPC and GPC. Recall that the (10, 6) BPC is constructed from the (9, 6) MDS code. The code tolerates up to three arbitrary failures. In addition, it recovers 73% of four failures.

Here we construct a (10, 6) GPC, which keeps the same coding dependency as the BPC. That is, the six data blocks are divided into two groups. Each group computes one redundant block. In addition, the code contains two global redundant blocks. It is easy to verify that, when blocks fail, the reconstruction read cost is the same for both the BPC and the GPC. In addition, they both tolerate up to three arbitrary failures. The GPC, however, recovers four failures more – 86% compared to 73%. Hence, the GPC provides higher fault tolerance. Indeed, in terms of the fault tolerance metric, the unrecoverable probability of the GPC is 3.1×10^{-7}, 45% lower than that of the BPC.



(a) Failure Pattern

(b) Reduced Tanner Graph

Fig. 11. Decoding generalized pyramid codes (five block failures).

The comparison of reconstruction read cost, fault tolerance and storage overhead is summarized in Figure 10. In summary, the GPC provides higher fault tolerance than the BPC, while keeping the other three key metrics the same.

6. DECODING GENERALIZED PYRAMID CODES

A single block failure can always be reconstructed using the smallest group containing the failed block. When multiple blocks fail in GPC, however, there are typically several different sets of blocks that can be used for reconstruction (simply called a *reconstruction set*). For instance, Figure 11(a) shows a GPC with five data block failures. There are at least two different reconstruction sets: (1) one set first recovers $\mathbf{d}_3$ and $\mathbf{d}_4$ horizontally (requires four block reads), then $\mathbf{d}_6$ vertically (requires one additional block read), and finally $\mathbf{d}_3$ and $\mathbf{d}_4$ horizontally (require three additional block reads). This reconstruction set requires eight block reads. (2) Another set first recovers $\mathbf{d}_3$ and $\mathbf{d}_4$ horizontally (requires four block reads), and then $\mathbf{d}_6$, $\mathbf{d}_7$, and $\mathbf{d}_8$ all vertically (require three additional block reads). This reconstruction set requires seven block reads and thus is more efficient. In general, determining a minimum size reconstruction set can be nontrivial. The focus of this section is to develop algorithms to find minimum reconstruction set for multiple failures.

Given multiple failures (including data and redundant blocks), we are concerned with two types of reconstruction: (i) *complete reconstruction* that recovers all the failed data and redundant blocks; and (ii) *on-demand reconstruction* that recovers a single failed data block. Complete reconstruction typically happens after permanent failures, where any affected data or redundant blocks need to be reconstructed. On the other hand, on-demand reconstruction typically happens during transient failures, where read requests need to access particular failed data blocks, but not redundant ones.

6.1. Complete Reconstruction

The minimum set for complete reconstruction can be derived based on the following theorem.

```
$\mathbf{D}^e$ := failed data nodes
$\mathbf{C}^a$ := all available redundant nodes
$overhead := \infty$
for $\mathbf{C}^a_s$ = subsets of $\mathbf{C}^a$ with size $|\mathbf{D}^e|$ do
    if $\exists$ full-size matching between $\mathbf{C}^a_s$ and $\mathbf{D}^e$ then
        $c_0 := |\mathbf{C}^a_s|$ + available data blocks connected to $\mathbf{C}^a_s$
        $o_l := c_0$ + overhead to recover failed redundant blocks
        $overhead = \min(overhead,\ o_l)$

return
```

Fig. 12. Minimum cost repair.

THEOREM 4. *Given d data block and c redundant block failures in a GPC, the minimum reconstruction set always includes exactly d redundant blocks. In addition, it includes every data block, from which the failed redundant blocks are computed (see proof in Appendix* II*)*

Based on Theorem 4, we develop the following algorithm. We enumerate all subsets of redundant blocks of size d (recall that d is the number of failed data blocks). For each subset, we establish a reduced decoding Tanner graph. A full-size matching in the Tanner graph implies successful reconstruction. The reconstruction set includes the subset of redundant blocks and all the data blocks required for reconstruction. The minimum reconstruction set is discovered after enumerating through all the redundant subsets (details shown in Figure 12).

The complexity of the algorithm is exponential in terms of $n - k$ (the number of redundant blocks, but this is typically *not* high in storage systems. For instance, the example in Figure 6 contains 8 redundant blocks and 5 failed data blocks, thus there are merely $\binom{8}{5} = 56$ subsets to evaluate.

6.2. On-Demand Reconstruction

Reconstructing a single failed data block requires a different set of blocks from reconstructing all the failed blocks. An algorithm for finding the minimum on-demand reconstruction set is described as follows.

Similar to the algorithm in Figure 12, we enumerate all subsets of redundant blocks of size d (again, d is the number of failed data blocks). For each subset, we establish a reduced decoding Tanner graph. If the graph contains a full-size matching, we run a breadth-first search, starting from the *target* failed data block. When encountering a data node in the Tanner graph, the search follows only the edge in the matching to the corresponding redundant node. When encountering a redundant node, the search follows all edges in the Tanner graph to all the data nodes, which have *not* been visited before. Let $\mathbf{D}_v$ denote the set of data nodes already visited; $\mathbf{C}_v$ the set of redundant nodes already visited; and $\mathbf{D}_c$ the set of all data nodes connected to $\mathbf{C}_v$. The search stops when $\mathbf{C}_v$ becomes large enough to recover $\mathbf{D}_v$ (i.e., $|\mathbf{D}_v| \leq |\mathbf{C}_v|$ **and** $\mathbf{D}_c \subseteq \mathbf{D}_v$). (Refer to Figure 13 for details.)

The minimum on-demand reconstruction set is discovered after enumerating through all the redundant subsets. The complexity is comparable to the algorithm in Figure 12.

We now go through an example to reconstruct a single failed block $\mathbf{d}_7$ in Figure 6. Since there are five failed data blocks, we enumerate all subsets of redundant blocks of size 5. The reduced decoding Tanner graph corresponding to one particular redundant

```
$Q$ := null // queue used for the breadth first search
$\mathbf{D}_v$ := null, $\mathbf{C}_v$ := null, $\mathbf{D}_c$ := null
$M$ := find a maximum matching
if $|M|$ is less than failed data blocks then
  return
$Q$.enqueue($n_0$) // $n_0$: the target failed data block
while $|Q| > 0$ do
  $n$ := $Q$.dequeue
  if $n$ is a data node then
    if $n \in \mathbf{D}_v$ then
      repeat
    $\mathbf{D}_v$ := $\mathbf{D}_v + \{n\}$
    $Q$.enqueue($M[n]$) // follow the edge in the matching
  else
    $\mathbf{C}_v$ := $\mathbf{C}_v + \{n\}$
    // follow all edges to data nodes
    for $n_d$ := data nodes connected to $n$ do
      if $n_d \notin \mathbf{D}_v$ then
        $Q$.enqueue($n_d$)
      if $n_d \notin \mathbf{D}_c$ then
        $\mathbf{D}_c$ := $\mathbf{D}_c + \{n_d\}$
    if $|\mathbf{D}_v| \leq |\mathbf{C}_v|$ and $\mathbf{D}_c \subseteq \mathbf{D}_v$ then
      last // found an access path for $n_0$
$o_l$ := $|\mathbf{C}_v|$ + available data blocks connected to $\mathbf{C}_v$
$overhead$ := min($overhead$, $o_l$)
return
```

Fig. 13. Minimum cost reconstruction read.

subset ($\{\mathbf{c}_1, \mathbf{c}_2, \mathbf{c}_3, \mathbf{c}_7, \mathbf{c}_8\}$) is depicted in Figure 6. To reconstruct block $\mathbf{d}_7$, the breadth-first search starts from $\mathbf{d}_7$, goes to $\mathbf{c}_7$, then $\mathbf{d}_3$, $\mathbf{c}_2$, $\mathbf{d}_4$, and stops at $\mathbf{c}_1$. In the end, the on-demand reconstruction set requires five blocks.

Each redundant subset might contain multiple full-size matchings, while the breadth-first search only explores one of them. Nevertheless, the following theorem states that this is sufficient to obtain the minimum on-demand reconstruction set.

THEOREM 5. *Given a failure pattern and a redundant subset, the algorithm in Figure* 13 *always yields the same* $\mathbf{C}_v$ *even following different full-size matchings. (see proof in Appendix* III*)*

7. CONCLUSION

In this article, we design flexible schemes to explore the tradeoffs between storage space and access efficiency in reliable data storage systems. We present two novel classes of codes: basic pyramid codes (BPC) and generalized pyramid codes (GPC). Both schemes require slightly more storage space than MDS codes, but significantly improve the critical performance of read during failures and unavailability.

We establish a necessary matching condition to characterize the limit of failure recovery, that is, unless the matching condition is satisfied, a failure case is impossible to recover. In addition, we define a maximally recoverable (MR) property. For all ERC schemes holding the MR property, the matching condition becomes sufficient, that is, all failure cases satisfying the matching condition are indeed recoverable. We show that GPC is the *first* class of non-MDS schemes holding the MR property.

APPENDIX I. Proof of Theorem 3

PROOF. We prove the theorem by induction on the construction algorithm (detailed in Figure 9). The base case is to show that a GPC with single redundant block holds the MR property. When the construction algorithm terminates, the generator matrix $\mathbf{G}$ is of size $(k+1)\times k$, where $\mathbf{g}_1$ through $\mathbf{g}_k$ form a identity matrix $\mathbf{I}_{k\times k}$. Without loss of generality, assume $\mathbf{d}_k$ is used to compute the single redundant block $\mathbf{c}_1$. Now consider the case with one single failure of $\mathbf{d}_k$. Its reduced decoding Tanner graph contains one edge, between $\mathbf{d}_k$ and $\mathbf{c}_1$. This is a full-size matching and satisfies the matching condition. We need to show this failure case is recoverable.

For the failure case, we examine the generator submatrix $\mathbf{G}_s^1 = [\mathbf{g}_1, \mathbf{g}_2, \cdots, \mathbf{g}_{k-1}, \mathbf{g}_{k+1}]^T$. Because $\mathbf{g}_1$ through $\mathbf{g}_{k-1}$ are from the identify matrix, we know rank$(\mathbf{G}_s^1 \backslash \mathbf{g}_{k+1}) = k-1$. Therefore, null$(\mathbf{G}_s^1 \backslash \mathbf{g}_{k+1})$ is a single row vector and must exist in the auxiliary matrix $\mathbf{U}$. The construction algorithm ensures that $\mathbf{g}_{k+1}$ is not orthogonal to the row vector. Hence, $\mathbf{G}_s^1$ is full-rank and the failure case is recoverable.

Now assume a GPC with $m-1$ $(m = n-k)$ redundant blocks holds the MR property. Next, we need to show that a GPC with m redundant blocks still holds the property. We consider a case with m failures where the last redundant block $\mathbf{c}_m$ is available. Otherwise, we can drop $\mathbf{c}_m$, which is also called *puncturing* the code, to include only $m-1$ redundant blocks. Based on the induction assumption, the MR property holds.

For the case with m failures, we need to show the following: if its reduced decoding Tanner graph contains a full-size matching M_f of size m, the case is recoverable. This is equivalent to showing that its generator submatrix $\mathbf{G}_s$ is full-rank. Without loss of generality, assume $\mathbf{d}_k$ is the failed data block connected to $\mathbf{c}_m$ in M_f.

Now modify the failure case and create a new one, by setting $\mathbf{d}_k$ as available and $\mathbf{c}_m$ as failed. As argued before, we can drop $\mathbf{c}_m$ and consider the new case with $m-1$ failures in the punctured code. Its reduced decoding Tanner graph contains a full-size matching of size $m-1$ (simply take M_f and remove the edge between $\mathbf{d}_k$ and $\mathbf{c}_m$). Based on the induction assumption, this case is recoverable, and thus its generator submatrix $\mathbf{G}'_s$ is full-rank, that is, rank$(\mathbf{G}'_s) = k$. Excluding $\mathbf{g}_k$, the row vector in the generator matrix for $\mathbf{d}_k$, we get rank$(\mathbf{G}'_s \backslash \mathbf{g}_k) = k-1$. Therefore, null$(\mathbf{G}'_s \backslash \mathbf{g}_k)$ is a single row vector and must exist in the auxiliary matrix $\mathbf{U}$. Again, the construction algorithm ensures it is not orthogonal to $\mathbf{g}_{k+m}$, the row vector in the generator matrix for $\mathbf{c}_m$. Hence, rank$(\mathbf{G}'_s \backslash \mathbf{g}_k + \{\mathbf{g}_{k+m}\}) = k$. Now realize that excluding $\mathbf{g}_k$ from $\mathbf{G}'_s$ and then including $\mathbf{g}_{k+m}$ yields exactly $\mathbf{G}_s$ – the generator submatrix for the case with m failures. Hence, we have shown that $\mathbf{G}_s$ is full-rank and the failure case is recoverable. The proof is complete. □

APPENDIX II. Proof of Theorem 4

PROOF. To recover d failed data blocks, at least d redundant blocks are needed. Hence, the minimum reconstruction set includes at least d redundant blocks. Next, we prove that including more than d redundant blocks will only increase the number of blocks required.

Both failed data and redundant blocks need to be recovered. We first show that to recover the d failed data blocks (denoted as $\mathbf{D}^e = \{\mathbf{d}_1^e, \cdots, \mathbf{d}_d^e\}$), the minimum reconstruction set should include exactly d available redundant blocks. We prove the statement by contradiction and assume the minimum reconstruction set in fact includes r available redundant blocks (denoted as $\mathbf{C}^a = \{\mathbf{c}_1^a, \cdots, \mathbf{c}_r^a\}$) and $r > d$. Further, assume the minimum reconstruction set includes s additional available data blocks (denoted as $\mathbf{D}_0^a = \{\mathbf{d}_1^a, \cdots, \mathbf{d}_s^a\}$), which is shown to the right of the Tanner graph in Figure 14. Since $\mathbf{D}^e$ is recoverable, there must exist a full-size matching between $\mathbf{D}^e$ and $\mathbf{C}^a$.

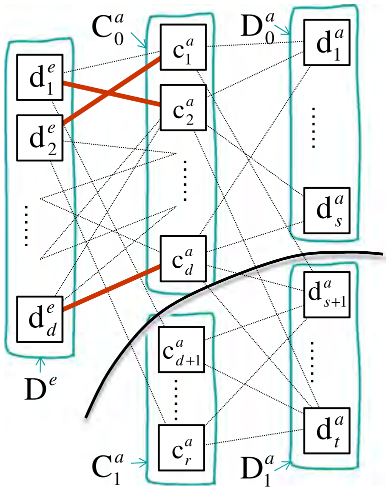


Fig. 14. Minimum cost recovery for failed data blocks.

Without loss of generality, assume the matching connects the first d nodes in $\mathbf{C}^a$, denoted as $\mathbf{C}_0^a = \{\mathbf{c}_1^a, \cdots, \mathbf{c}_d^a\}$. Then, find another reconstruction set, which includes only redundant blocks in $\mathbf{C}_0^a$. Of course, this reconstruction set needs to include additional available data blocks (denoted as $\mathbf{D}_1^a = \{\mathbf{d}_{s+1}^a, \cdots, \mathbf{d}_t^a\}$). Based on the assumption, the recovery overhead of this reconstruction set is *not* minimum (the path includes $\mathbf{C}_0^a$, $\mathbf{D}_0^a$, and $\mathbf{D}_1^a$). Hence, $d + t > r + s$ (i.e., $|\mathbf{C}_1^a| < |\mathbf{D}_1^a|$). Note that each node in $\mathbf{D}_1^a$ is connected to at least one node in $\mathbf{C}_0^a$. Since $\mathbf{D}_1^a$ is *not* included in the minimum reconstruction set, their values must have been canceled out by $\mathbf{C}_1^a$ during decoding. For this reason, each node in $\mathbf{C}_1^a$ should connect to at least one node in $\mathbf{D}_1^a$.

Now we consider only nodes in $\mathbf{C}_1^a$, $\mathbf{D}_1^a$, and edges between them. We claim that there must exist a full-size matching between $\mathbf{C}_1^a$ and $\mathbf{D}_1^a$. Assuming this is *not* true, then, the maximum matching size will be less than $|\mathbf{C}_1^a|$, as the size of the corresponding minimum node cover N_c (recall that maximum matching and minimum node covers are equivalent in the bipartite graph). Denote $\mathbf{C}_1^{a\prime}$ as those nodes in $\mathbf{C}_1^a$ while not in N_c, and denote $\mathbf{D}_1^{a\prime}$ as those nodes in $\mathbf{D}_1^a$ while also in N_c. Based on the property of node cover, each node in $\mathbf{C}_1^{a\prime}$ is connected to at least one node in $\mathbf{D}_1^{a\prime}$. On the other hand, $|\mathbf{C}_1^{a\prime}| > |\mathbf{D}_1^{a\prime}|$ (based on the assumption). Now that nodes in $\mathbf{C}_1^{a\prime}$ do not connect to other node in $\mathbf{D}_1^d$, at least one of them can be removed from the minimum access path without affecting recoverability. This means that the cost of the minimum reconstruction set can be reduced further, which is certainly a contradiction. Therefore, there must exist a full-size matching between $\mathbf{C}_1^a$ and $\mathbf{D}_1^a$.

Without loss of generality, assume this matching connects $\mathbf{C}_1^a$ to the first $r - d$ nodes in $\mathbf{D}_1^a$ (denote them as $\mathbf{D}_1^{a\prime\prime}$). We now consider a new failure case, which consists of $\mathbf{D}^e$, $\mathbf{D}_1^{a\prime\prime}$ and one more node from $\mathbf{D}_1^a$ (say $\mathbf{d}_t^a$). Using the redundant block in $\mathbf{C}^a$ ($\mathbf{C}_0^a$ and $\mathbf{C}_1^a$) and the data blocks in $\mathbf{D}_0^a$, it is clear that $\mathbf{D}^e$ can be recovered. Next, we examine the remaining Tanner graph. It contains a full-size matching, which consists of a matching of size $r - d$ between $\mathbf{C}_1^a$ and $\mathbf{D}_1^{a\prime\prime}$, together with an additional edge between $\mathbf{d}_t^a$ and at least one node in $\mathbf{C}_0^a$. Therefore, the rest of the failed data blocks can also be decoded. To this end, we have demonstrated a case, where $d + (r - d) + 1 = r + 1$ failed data blocks are recovered from only r redundant blocks. This creates a contradiction. Therefore, the minimum reconstruction set should include exactly d available redundant blocks.

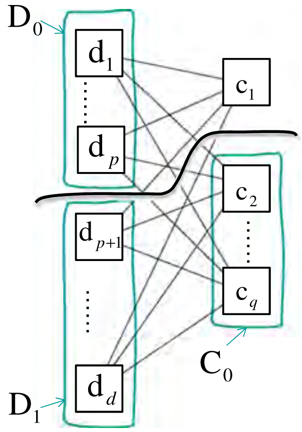


Fig. 15. Minimum cost recovery for failed redundant blocks.

In the second part of the proof, we show that to recover the failed redundant blocks, the minimum reconstruction set includes every data block from which these redundant blocks are originally computed. In other words, no redundant block can be computed from the combination of data blocks and redundant blocks with less overhead. Using a contradictory argument, we assume this claim is not true on one particular redundant block $\mathbf{c}_1$. Instead of computing from d data blocks (say $\mathbf{d}_1, \cdots, \mathbf{d}_d$), assume $\mathbf{c}_1$ can instead be computed with a minimum overhead from p data blocks (denoted as $\mathbf{D}_0 = \{\mathbf{d}_1, \cdots, \mathbf{d}_p\}$) together with $(q-1)$ redundant blocks (denoted as $\mathbf{C}_0 = \{\mathbf{c}_2, \cdots, \mathbf{c}_q\}$), where $p + (q-1) < d$ (shown in Figure 15). Under this assumption, there must exist a full-size matching between the rest $(d-p)$ data blocks (denoted as $\mathbf{D}_1$) and $\mathbf{C}_0$. (Otherwise, we can examine the corresponding minimum node cover and show that at least one node in $\mathbf{C}_0$ could be computed from $\mathbf{D}_0$ and the rest of the blocks in $\mathbf{C}_0$. This means $\mathbf{c}_1$ can be computed even if this node is removed from $\mathbf{C}_0$, which further implies even less overhead to compute $\mathbf{c}_1$.) Hence, the maximum matching size between $\mathbf{D}_1$ and $\mathbf{C}_0$ is $(q-1)$, and denote the $(q-1)$ matching nodes from $\mathbf{D}_1$ as $\mathbf{D}_1{}' = \{\mathbf{d}_{p+1}, \cdots, \mathbf{d}_{p+q-1}\}$. We now consider a particular failure case of q failed data blocks, which include all the $(q-1)$ nodes in $\mathbf{D}_1{}'$ and $\mathbf{d}_d$. This failure case should be recoverable using all the $(q-1)$ redundant blocks in $\mathbf{C}_0$ together with $\mathbf{c}_1$. This is because there exists a full-size matching in the corresponding Tanner graph (a matching of size $(q-1)$ between $\mathbf{D}_1{}'$ and $\mathbf{C}_0$, together with an edge between $\mathbf{d}_d$ and $\mathbf{c}_1$). On the other hand, $\mathbf{c}_1$ can be computed from $\mathbf{D}_0$ and $\mathbf{C}_0$, and thus is *not* an effective redundant block (or $\mathbf{c}_1$ is linear dependent on $\mathbf{D}_0$ and $\mathbf{C}_0$). Hence, $\mathbf{c}_1$ should be removed. To this end, it is impossible to recover q failed data blocks from $(q-1)$ redundant blocks. This creates a contradiction. In summary, the proof is complete with the combination of the above two parts. □

APPENDIX III. Proof of Theorem 5

PROOF. It is easy to show that $|\mathbf{D}_v| = |\mathbf{C}_v|$, when the algorithm terminates. Now we prove the theorem by contradiction. Assume the algorithm yields two different results (denoted as $\mathbf{D}_{v0}$, $\mathbf{C}_{v0}$ and $\mathbf{D}_{v1}$, $\mathbf{C}_{v1}$, respectively) when following two different matchings. It is clear that $\mathbf{D}_{v0}$ and $\mathbf{D}_{v1}$ share at least the target data block. Then, $\mathbf{C}_{v0}$ and $\mathbf{C}_{v1}$ share at least one redundant block as well. Otherwise, failed data blocks in neither $\mathbf{D}_{v0}$ nor $\mathbf{D}_{v1}$ will *not* be recoverable, because they have to be decoded from redundant blocks not in $\mathbf{C}_{v0}$ or $\mathbf{C}_{v1}$, but there are fewer redundant blocks than failed data blocks. On the other hand, any data blocks, which are connected to the shared redundant block between $\mathbf{C}_{v_0}$ and $\mathbf{C}_{v_1}$, have to be shared by $\mathbf{D}_{v_0}$ and $\mathbf{D}_{v_1}$. Hence, following the same logic and using induction argument, we can show that $\mathbf{D}_{v_0}$ and $\mathbf{D}_{v_1}$ can *not* overlap. Then, one has to contain the other. Without loss of generality, assume $\mathbf{D}_{v_0}$ contains $\mathbf{D}_{v_1}$ (then $\mathbf{C}_{v_0}$ also contains $\mathbf{C}_{v_1}$). If that's the case, in the matching between $\mathbf{D}_{v_0}$ and $\mathbf{C}_{v_0}$, at least one node in both $\mathbf{D}_{v_0}$ and $\mathbf{D}_{v_1}$ should *not* be connected to $\mathbf{C}_{v_1}$. Based on the existence of the full-size matching, at least one node in $\mathbf{C}_{v_1}$ should connect to a node in $\mathbf{D}_{v_0}$ while *not* in $\mathbf{D}_{v_1}$. This implies the algorithm would *not* have terminated with $\mathbf{D}_{v_1}$ and $\mathbf{C}_{v_1}$. Hence, neither is it possible for $\mathbf{D}_{v_0}$ to contain $\mathbf{D}_{v_1}$. In summary, this is a contradiction and the proof is complete. □

ACKNOWLEDGMENTS

The authors would like to thanks Cha Zhang, Yunnan Wu and Philip A. Chou at Microsoft Research for very helpful and inspiring discussions on various parts during this work.

REFERENCES

Abd-El-Malek, M., W. V. C. Ii, Cranor, C., Ganger, G. R., Hendricks, J., Klosterman, A. J., Mesnier, M., Prasad, M., Salmon, B., Sambasivan, R. R., Sinnamohideen, S., Strunk, J. D., Thereska, E., Wachs, M., and Wylie, J. J. 2005. Ursa minor: Versatile cluster-based storage. In *Proceedings of USENIX Conference on File and Storage Technologies*.

Aguilera, M. K., Janakiraman, R., and Xu, L. 2005. Using erasure codes for storage in a distributed system. In *Proceedings of IEEE International Conference on Dependable Systems and Networks*. IEEE, Los Alamitos, CA.

Blahut, R. E. 2003. *Algebraic Codes for Data Transmission*. Cambridge University Press.

Blaum, M. and Roth, R. M. 1999. On lowest-density MDS codes. *IEEE Trans. Inf. Theory*.

Blaum, M., Brady, J., Bruck, J., and Menon, J. 1995. EVENODD: An efficient scheme for tolerating double disk failures in RAID architectures. *IEEE Trans.Computers*.

Blomer, 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, ICSI, Berkeley, CA.

Borthakur, D., Schmidt, R., Vadali, R., Chen, S., and Kling, P. 2010. HDFS RAID. Hadoop User Group Meeting.

Cadambe, V. R., Huang, C., and Li, J. 2011. Permutation code: Optimal exact-repair of a single failed node in MDS code based distributed storage systems. In *Proceedings of IEEE International Symposium on Information Theory*. IEEE, Los Alamitos, CA.

Cadambe, V. R., Huang, C., Li, J., and Mehrotra, S. 2011. Polynomial length MDS codes with optimal repair in distributed storage. In *Proceedings of the Asilomar Conference on Signals, Systems and Computers*.

Calder, B., Wang, J., Ogus, A., Nilakantan, N., Skjolsvold, A., McKelvie, S., Xu, Y., Srivastav, S., Wu, J., Simitci, H., Haridas, J., Uddaraju, C., Khatri, H., Edwards, A., Bedekar, V., Mainali, S., Abbasi, R., Agarwal, A., Ul Haq, M. F., Ul Haq, M. I., Bhardwaj, D., Dayanand, S., Adusumilli, A., McNett, M., Sankaran, S., Manivannan, K., and Rigas, L. 2011. Windows azure storage: A highly available cloud storage service with strong consistency. In *Proceedings of the ACM Symposium on Operating Systems Principles*. ACM, New York.

Chang, F., Ji, M., Leung, S.-T., Maccormick, J., Perl, S., and Zhang, L. 2002. Myriad: Cost-effective disaster tolerance. In *Proceedings of the USENIX Conference on File and Storage Technologies*.

Chen, M., Huang, C., and Li, J. 2007. On the maximally recoverable property for multi-protection group codes. In *Proceedings of the IEEE International Symposium on Information Theory*.

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*.

Chen, Y., Edler, J., Goldberg, A., Gottlieb, A., Sobti, S., and Yianilos, P. 1999. A prototype implementation of archival intermemory. In *Proceedings of the ACM Conference on Digital Libraries*. ACM, New York.

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 USENIX Conference on File and Storage Technologies*.

Dimakis, A. G., Godfrey, P. B., Wu, Y., Wainwright, M., and Ramchandran, K. 2010. Network coding for distributed storage systems. *IEEE Trans. Inf. Theory*.

Dingledine, R., Freedman, M. J., and Molnar, D. 2000. The free haven project: Distributed anonymous storage service. In *Proceedings of the Workshop on Design Issues in Anonymity and Unobservability*.

Dubnicki, C., Gryz, L., Heldt, L., Kaczmarczyk, M., Kilian, W., Strzelczak, P., Szczepkowski, J., Ungureanu, C., and Welnicki, M. 2009. Hydrastor: A scalable secondary storage. In *Proceedings of the USENIX Conference on File and Storage Technologies*.

Fikes, A. 2010. Storage architecture and challenges. Google Faculty Summit.

Ford, D., Labelle, F., Popovici, F. I., Stokely, M., Truong, V.-A., Barroso, L., Grimes, C., and Quinlan, S. 2010. Availability in globally distributed storage systems. In *Proceedings of the USENIX Symposium on Operating Systems Design and Implementation*.

Gallager, R. G. 1963. Low-density parity-check codes. MIT Press, Cambridge, MA.

Gantenbein, D. 2012. A better way to store data. Microsoft Res. Featured Stories. http://research.microsoft.com/en-us/news/features/erasurecoding-090512.aspx.

Ghemawat, S., Gobioff, H., and Leung, S.-T. 2003. The Google file system. In *Proceedings of the ACM Symposium on Operating Systems Principles*. ACM, New York.

Gopalan, P., Huang, C., Simitci, H., and Yekhanin, S. 2011. On the locality of codeword symbols. In *Proceedings of the Allerton Conference on Communication, Control, and Computing*.

Greenan, K. M., Li, X., and Wylie, J. J. 2010. Flat XOR-based erasure codes in storage systems: Constructions, efficient recovery, and tradeoffs. In *Proceedings of the IEEE Mass Storage Systems and Technologies*. IEEE, Los Alamitos, CA.

Greenan, K. M., Long, D. E., Miller, E. L., Schwarz, T. J. E., and Wylie, J. J. 2008. A spin-up saved is energy earned: Achieving power-efficient, erasure-coded storage. In *Proceedings of the USENIX Workshop on Hot Topics in System Dependability*.

Greenan, K. M., Miller, E., and Wylie, J. J. 2008. Reliability of flat XOR-based erasure codes on heterogeneous devices. In *Proceedings of the IEEE International Conference on Dependable Systems and Networks*.

Grolimund, D. 2007. P2P online storage. In *Proceedings of the Web 2.0 Expo*.

Haeberlen, A., Mislove, A., and Druschel, P. 2005. Glacier: Highly durable, decentralized storage despite massive correlated failures. In *Proceedings of the USENIX Symposium on Networked Systems Design and Implementation*.

Hafner, J. L. 2005. Weaver codes: Highly fault tolerant erasure codes for storage systems. In *Proceedings of the USENIX Conference on File and Storage Technologies*.

Hafner, J. L. and Rao, K. 2006. Notes on reliability models for non-MDS erasure codes. IBM Tech. rep. RJ10391.

Hafner, J. L., Deenadhayalan, V. W., Rao, K., and Tomlin, J. A. 2005. Matrix methods for lost data reconstruction in erasure codes. In *Proceedings of the USENIX Conference on File and Storage Technologies*.

Hamilton, J. 2007. An architecture for modular data centers. In *Proceedings of the Conference on Innovative Data Systems Research*.

Hosekote, D. K., He, D., and Hafner, J. L. 2007. REO: A generic RAID engine and optimizer. In *Proceedings of the USENIX Conference on File and Storage Technologies*.

Huang, C. and Xu, L. 2003. Fast software implementation of finite field operations. Tech. rep., Washington University, St. Louis, MO.

Huang, C. and Xu, L. 2008. Star: An efficient coding scheme for correcting triple storage node failures. *IEEE Trans. Computers*.

Huang, C., Chen, M., and Li, J. 2007. Pyramid codes: Flexible schemes to trade space for access efficiency in reliable data storage systems. In *Proceedings of the IEEE International Symposium on Network Computing and Applications*. IEEE, Los Alamitos, CA.

Huang, C., Li, J., and Chen, M. 2007. On optimizing XOR-based codes for fault-tolerant storage applications. In *Proceedings of the IEEE Information Theory Workshop*. IEEE, Los Alamitos, CA.

Huang, C., Simitci, H., Xu, Y., Ogus, A., Calder, B., Gopalan, P., Li, J., and Yekhanin, S. 2012. Erasure Coding in Windows Azure Storage. In *Proceedings of the USENIX Annual Technical Conference*.

Khan, O., Burns, R., Plank, J., and Huang, C. 2011. In search of I/O-optimal recovery from disk failures. In *Proceedings of the USENIX Workshop on Hot Topics in Storage and File Systems*.

Khan, O., Burns, R., Plank, J., Pierce, W., and Huang, C. 2012. Rethinking erasure codes for cloud file systems: Minimizing I/O for recovery and degraded reads. In *Proceedings of the USENIX Conference on File and Storage Technologies*.

Kubiatowicz, J., Bindel, D., Chen, Y., Czerwinski, S., Eaton, P., Geels, D., Gummadi, R., Rhea, S., Weatherspoon, H., Weimer, W., Wells, C., and Zhao, B. 2000. Oceanstore: An architecture for global-scale persistent storage. In *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems*.

Luby, M. G., Mitzenmacher, M., Shokrollahi, A., and Spielman, D. A. 2001. Efficient erasure correcting codes. *IEEE Trans. Inf. Theory*.

Luo, J., Xu, L., and Plank, J. S. 2009. An efficient XOR-scheduling algorithm for erasure codes encoding. In *Proceedings of the IEEE International Conference on Dependable Systems and Networks*.

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

Maymounkov, P. and Mazieres, D. 2003. Rateless codes and big downloads. In *Proceedings of the International Workshop on Peer-To-Peer Systems*.

Papailiopoulos, D. S., Luo, J., Dimakis, A. G., Huang, C., and Li, J. 2012. Simple regenerating codes: Network coding for cloud storage. In *Proceedings of the IEEE INFOCOM Mini-Conference*. IEEE, Los Alamitos, CA.

Plank, J. S. 1997. A tutorial on reed-solomon coding for fault-tolerance in RAID-like systems. *Softw. Pract. Exper*.

Plank, J. S. 2008. The RAID-6 liberation codes. In *Proceedings of the USENIX Conference on File and Storage Technologies*.

Plank, J. S. and Xu, L. 2006. Optimizing cauchy reed-solomon codes for fault-tolerant network storage applications. In *Proceedings of the IEEE International Symposium on Network Computing and Applications*.

Reed, I. S. and Solomon, G. 1960. Polynomial codes over certain finite fields. *J. Soc. Industrial Appl. Math*.

Rhea, S., Eaton, P., Geels, D., Weatherspoon, H., Zhao, B., and Kubiatowicz, J. 2003. Pond: The oceanstore prototype. In *Proceedings of the USENIX Conference on File and Storage Technologies*.

Rowstron, A. and Druschel, P. 2001. Storage management and caching in past, a large-scale, persistent peer-to-peer storage utility. In *Proceedings of the ACM Symposium on Operating Systems Principles*. ACM, New York.

Saito, Y., Frolund, S., Veitch, A., Merchant, A., and Spence, S. 2004. FAB: Building distributed enterprise disk arrays from commodity components. In *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems*.

Schrijver, A. 2003. Combinatorial optimization, polyhedra and efficiency. *Alg. Combinatorics*.

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 USENIX Conference on File and Storage Technologies*.

Shvachko, K., Kuang, H., Radia, S., and Chansler, R. 2010. The hadoop distributed file system. In *Proceedings of the IEEE Symposium on Massive Storage Systems and Technologies*. IEEE, Los Alamitos, CA.

Suh, C. and Ramchandran, K. 2011. Exact regeneration codes for distributed storage repair using interference alignment. *IEEE Trans. Inf. Theory*.

Tanner, R. M. 1981. A recursive approach to low complexity codes. *IEEE Trans. Inf. Theory*.

Ungureanu, C., Atkin, B., Aranya, A., Gokhale, S., Rago, S., Calkowski, G., Dubnicki, C., and Bohra, A. 2010. Hydrafs: A high-throughput file system for the hydrastor content-addressable storage system. In *Proceedings of the USENIX Conference on File and Storage Technologies*.

Wang, Z., Tamo, I., and Bruck, J. 2011. On codes for optimal rebuilding access. Tech. rep. ETR111, Caltech.

Weatherspoon, H. and Kubiatowics, J. 2001. Erasure coding vs. replication: A quantitative comparison. In *Proceedings of the International Workshop on Peer-To-Peer Systems*.

Welch, B., Unangst, M., Abbasi, Z., Gibson, G., Mueller, B., Small, J., Zelenka, J., and Zhou, B. 2008. Scalable performance of the Panasas parallel file system. In *Proceedings of the USENIX Conference on File and Storage Technologies*.

Wildani, A., Schwarz, T. J. E., Miller, E. L., and Long, D. E. 2009. Protecting against rare event failures in archival systems. In *Proceedings of the IEEE International Symposium on Modelling, Analysis, and Simulation of Computer and Telecommunication Systems*.

Received October 2011; revised August 2012; accepted September 2012