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

Cheng Huang, Minghua Chen, and Jin Li
Microsoft Research, Redmond, WA 98052

Abstract

To flexibly explore the trade-offs between storage space and access efficiency in reliable data storage systems, we describe two classes of erasure resilient coding schemes: basic *and* generalized Pyramid Codes. *The basic Pyramid Codes can be simply derived from any existing codes, and thus all known efficient encoding/decoding techniques directly apply. The generalized Pyramid Codes are radically advanced new codes, which can further improve access efficiency and/or reliability upon the basic Pyramid Codes.*

We also establish a necessary condition for any failure pattern to be ever recoverable, and show that the generalized Pyramid Codes are optimal in failure recovery (i.e., the necessary condition is also sufficient, and any failure pattern that is ever recoverable can indeed be recovered).

1 Introduction

A promising direction in building large scale storage systems is to harness the collective storage capacity of massive commodity computers. While many practical systems demand high reliability (such as five 9s), individual components can rarely live up to that standard. Interestingly, a recent study of disk drives [20] shows that the real-world reliability is much lower than expected.

On the other hand, large scale production systems (e.g., GFS [7]) have successfully demonstrated the feasibility of building reliable data storage using much less reliable commodity components. To ensure high reliability, these systems often use replication schemes, where data blocks are replicated a number of 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. Another reason is because replication schemes often 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 space than the data collection itself. In data centers, storage overhead directly translates into costs in hardware (disk drives and associated equipments), as well as costs to operate, which include building space, power, cooling, and maintenance, etc. As a matter of fact, it is recently reported that over 55% of the cost of a typical data center (providing Microsoft's Windows Live services in this case) is due to building, power distribution and equipments [9]. In wide area storage applications, the storage overhead also means much less effective usage of allocated space. For instance, WheelFS [21] proposes to build a distributed storage system using PlanetLab machines, where users often have storage quotas. Shrinking the effective storage usage to $1/3$ (3-replication) or even less (due to lower reliability of individual PlanetLab machines than that in data centers, a higher replication ratio is often required) will *not* appear as an attractive solution.

Naturally, many *Erasure Resilient Coding* (ERC) based schemes (e.g. Oceanstore [12]) are proposed to reduce the storage overhead. In a typical ERC scheme, a certain mathematical transform maps k data blocks into n total blocks (k original data and $n - k$ redundant). Blocks often have the same size and can be physically mapped to bytes, disk sectors, hard drives, and computers, etc. When failures happen, failed blocks (or simply *erasures*) can be recovered using other available data and redundant blocks (with proper mathematical transforms). Such an ERC scheme is often called an (n, k)-ERC scheme.

Here is a comparison. Assuming each block fails with an independent probability of 0.01, then a $(16, 12)$-ERC scheme and a 3-replication scheme provide the same level of reliability (five 9s). Clearly, the ERC scheme requires only 16 blocks in total, compared to 36 blocks by the replication scheme, both for 12 data blocks. Hence, the storage savings of ERC schemes are superior. However, beyond RAID systems [4], ERC schemes are yet to see any large scale production level adoption. We believe there are two fundamental obstacles. First, it's very difficult to get ERC schemes right. Consistency issue has long been a huge concern. And despite of numerous efforts to address the problem, all solutions remain quite complicated. It's very challenging to design and implement, not even mention to verify, such schemes. Sometimes, the complexity simply scares engineering efforts away. Second, ERC schemes often suffer greatly on the I/O performance. In the $(16, 12)$-ERC scheme, writing a data block takes quite a number of operations (5 reads of old blocks – 1 data and 4 redundant, 5 diffs to compute deltas, and another 5 writes to update all the related blocks [1]). Reading takes 12 reads when hitting a failed data block (in order to perform a proper mathematical transform).

These obstacles, however, can be largely overcome by exploring practical application needs. Many production services have already been successfully built on top of *append-only* storage systems, where write operations only append to the end of existing data and data is rarely modified once written. This is viable as many data collections are by large static (notable examples are those ever exploding media content collections, due to the boom of portable music devices and Internet videos). Based on such observation, we envision a hybrid approach, where ERC schemes are combined together with replication schemes. Fresh data written into the system are first replicated to ensure reliability. In this way, the write performance is the same as pure replication schemes. ERC is later applied to *completed* data blocks. After redundant blocks are created, replicas of the original data blocks are deleted. This usually happens as a background process, e.g., when the system utilization enters a valley period. Clearly, most data blocks will be protected by ERC, and only a small number of *active* blocks will be in replication. Hence, the storage overhead is very close to pure ERC schemes. Moreover, since ERC is only applied to completed data blocks that rarely change, the consistency issue is greatly alleviated. In summary, several goals can be achieved simultaneously: 1) simplified design, implementation and verification; 2) low storage overhead; and 3) much improved write performance.

To this end, the remaining issue is how to achieve good read performance. Indeed, the read performance has great impact on the overall system performance, since it dictates peak system load and/or peak bandwidth usage. Further, as most systems incur many more reads than writes, the read performance is certainly a primary design concern. However, in the hybrid approach, most blocks are covered by ERC, and hence reads are almost as difficult as in pure ERC schemes. Instead of jumping from ERC schemes directly to replication schemes, which *do* improve the read performance, but at the cost of significantly higher storage overhead, we describe schemes that can achieve much better read performance with only moderately increased storage overhead. If traditional ERC schemes and replication schemes are regarded as two ends of trading storage space for access efficiency, our schemes allow flexible exploration of the entire spectrum.

Specifically, we describe two classes of erasure resilient coding schemes: *basic* and *generalized Pyramid Codes*. The basic Pyramid Codes can be simply derived from any existing codes, and thus all known efficient encoding/decoding techniques directly apply. The generalized Pyramid Codes are radically advanced new codes, which can further improve access efficiency and/or reliability upon the basic Pyramid Codes. We also establish a necessary condition for any failure pattern to be ever recoverable, and show that the generalized Pyramid Codes are optimal in failure recovery (i.e., the necessary condition is also sufficient, and any failure pattern that is ever recoverable can indeed be recovered).

The rest of the paper is organized as follows. Section 2 describes the basic Pyramid Codes and Section 3 focuses on the generalized Pyramid Codes. Section 4 lists some additional related work. We make concluding remarks in Section 5, and more importantly, raise a few open issues.

2 Basic Pyramid Codes

Let a distributed storage system be composed of n blocks, where k blocks are *data blocks*, and the other $m = n - k$ blocks are *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.

2.1 Brief primer on MDS codes

Before presenting Pyramid Codes, let us briefly review maximum distance separable (MDS) [14] erasure resilient coding, which attracts particular attention in distributed storage system design. When an ERC scheme applies a (n, k) MDS code in a distributed storage system, $m = n-k$ redundant blocks are computed from k original data blocks. The MDS property guarantees that all the original data are accessible as long as any k among the n blocks are functional. That is, the system is resilient to $n - k$ arbitrary failures.

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 a $(k + 1, k)$ MDS code. The replication scheme, which creates r replicas for each data block, is indeed a $(1 + r, 1)$ MDS code. Reed-Solomon codes [18] are a class of the most widely used MDS codes.

2.2 Basic Pyramid Codes: an example

Now we use an example to describe the Pyramid Codes, which can significantly improve the read performance. Our example constructs from a $(11, 8)$ MDS code, which could be a Reed-Solomon code, or other MDS code, such as STAR [11]. (Note that MDS codes are *not* required, but

Pyramid Codes constructed from MDS codes *do* have certain good properties, which will become clear later.)

For an $(11, 8)$ MDS code, the 8 data blocks are separated into two equal size groups $S_1 = \{\mathbf{d}_1, \mathbf{d}_2, \mathbf{d}_3, \mathbf{d}_4\}$ and $S_2 = \{\mathbf{d}_5, \mathbf{d}_6, \mathbf{d}_7, \mathbf{d}_8\}$. Two of the three redundant blocks are kept unchanged (say $\mathbf{c}_2$ and $\mathbf{c}_3$). They are now called *global* redundant blocks, which cover all the 8 data blocks. Then, a new redundant block is computed for group S_1, which is denoted as *group (or local)* redundant block $\mathbf{c}_{1,1}$. The computation is done as if computing $\mathbf{c}_1$ in the original MDS code, except for setting all the data blocks in S_2 to 0. Similarly, group redundant block $\mathbf{c}_{1,2}$ is computed for S_2. It is easy to see that group redundant blocks are only affected by data blocks in the corresponding groups and *not* by other groups at all. This yields a $(12, 8)$ Pyramid Code.

Algebraically, each data or redundant block can be represented as many *symbols* (or *elements*) in finite fields (or rings) [13]. 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 called *decoding* (or *recovery*). Without loss of generality and yet to keep the presentation simple, we assume each block is merely one symbol. Most ERC schemes apply linear block codes, where the redundant blocks are *linear* combinations of the data blocks. For instance, in the $(11, 8)$ MDS code, the redundant block $\mathbf{c}_1$ satisfies

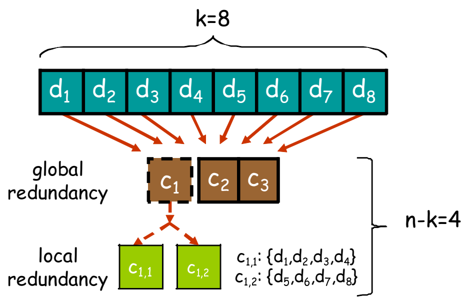
$$\mathbf{c}_1 = \sum_{i=1}^{8} \alpha_i \mathbf{d}_i,$$

where α_i's are symbols in the same field (or ring). Based on this representation, the new redundant blocks in the Pyramid Code satisfy

$$\mathbf{c}_{1,1} = \sum_{i=1}^{4} \alpha_i \mathbf{d}_i, \quad \mathbf{c}_{1,2} = \sum_{i=5}^{8} \alpha_i \mathbf{d}_i.$$

Hence, $\mathbf{c}_{1,1} + \mathbf{c}_{1,2} = \mathbf{c}_1$ (all $\sum$'s and $+$'s are binary sum). To this end, the group redundant blocks can be interpreted as the *projection* of the original redundant block in the MDS code onto each group (by setting the data blocks in all other groups to 0). Alternatively, given the group redundant blocks, they can be combined (again, binary sum) to compute the original redundant block in the MDS code. The Pyramid Code constructed in this example is shown in Figure 1(a). For convenience, we define the concept of *configuration*, which represents all data block subsets used to compute the redundant blocks. For instance, the configuration of this code is $\mathbf{c}_{1,1} : S_1$, $\mathbf{c}_{1,2} : S_2$, and $\mathbf{c}_2, \mathbf{c}_3 : S_1 \cup S_2$.

Now, we examine interesting properties of the Pyramid Code. First of all, it has the same write overhead as the original MDS code. Whenever any data block is updated, the Pyramid Code needs to update 3 redundant blocks (both



(a) construction

# of failed blocks		0	1	2	3	4
MDS code (11, 8)	recoverability (%)	100	100	100	100	0
	avg. read overhead	1.0	1.64	2.27	2.91	-
Pyramid Code (12, 8)	recoverability (%)	100	100	100	100	68.89
	avg. read overhead	1.0	1.25	1.74	2.37	2.83

(b) comparison

Figure 1. Construction of $(12, 8)$ Pyramid Code and comparison with $(11, 8)$ MDS code.

$\mathbf{c}_2$, $\mathbf{c}_3$, plus either $\mathbf{c}_{1,1}$ or $\mathbf{c}_{1,2}$), while the MDS code also updates 3 redundant blocks ($\mathbf{c}_1$, $\mathbf{c}_2$ and $\mathbf{c}_3$).

Secondly, we claim that the $(12, 8)$ Pyramid Code can also recover 3 arbitrary erasures, the same as the original $(11, 8)$ MDS code. To show this, assume there are 3 arbitrary erasures out of the 12 total 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 is unavailable. In the first case, $\mathbf{c}_1$ can be computed from $\mathbf{c}_{1,1}$ and $\mathbf{c}_{1,2}$. Then, it becomes recovering 3 erasures from the original $(11, 8)$ MDS code, which is certainly doable. 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 2 failed blocks. Hence, from the perspective of the original MDS code, there are at most 3 failures ($\mathbf{c}_1$ and two other failed blocks) and thus is decodable.

Third, the Pyramid Code is superior in terms of the read overhead. When any data block fails, the Pyramid Code can decode using local redundant blocks, which leads to read overhead of 4, compared to 8 in the MDS code. Finally, note that the Pyramid Code improves the read performance at the cost of using one additional redundant block. Hence, this example literally demonstrates the *core* concept of how the Pyramid Codes can trade storage space for access efficiency.

Next, we show detailed comparisons between the two codes. Two performance metrics are used throughout the paper. When the number of failed blocks (either data or redundant) is r, there are $\binom{n}{r}$ possible failure cases. The first metric, *recoverability*, represents the ratio between the number of recoverable cases and the total cases. It can be easily linked with the reliability of the overall system using failure probability models (we do *not* expand along this direction, as it is *not* the focus of this paper). The second metric, *average read overhead*, represents the average overhead to access each data block. Consider an example of 1 block failure in the $(11, 8)$ MDS code. If the failure is a redundant block (3/11 chance), then all the data blocks can be accessed directly, so the average read overhead is 1. Otherwise, the failure is a data block (8/11 chance), then the read overhead is 8 for the failed data block and 1 for the remaining 7 data blocks. Hence, the average read overhead is $(8 + 7)/8$. Altogether, the average read overhead is $1 \times 3/11 + (8+7)/8 \times 8/11 = 1.64$. Detailed comparisons on these two metrics are shown in Figure 1(b). We observe that the additional redundant block in Pyramid Code helps to reduce the read overhead under all failure patterns, compared to the MDS code. Moreover, it also helps the Pyramid Code to battle additional failures (4 failure in here).

2.3 Basic Pyramid Codes: construction

Formally, a basic Pyramid Code can be constructed as follows. It starts with a (n, k) code (preferably a MDS code), and separates the data blocks into L disjoint groups (denoted as S_l, $l = 1, \cdots, L$), where group S_l contains k_l blocks (i.e., $|S_l| = k_l$). Next, it keeps m_1 out of the m redundant blocks unchanged, and computes $m_0 = m - m_1$ new redundant blocks for each group S_l. The j^{th} group redundant block for group S_l (denoted as $\mathbf{c}_{j,l}$) is simply a projection of the j^{th} redundant block in the original 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 original code, but simply setting all groups other than S_l to 0. Again, the combination of all $\mathbf{c}_{j,l}$'s for the same l yields the redundant block $\mathbf{c}_j$ in the original code. Moreover, if a Pyramid Code is constructed from a MDS code, it satisfies the following property.

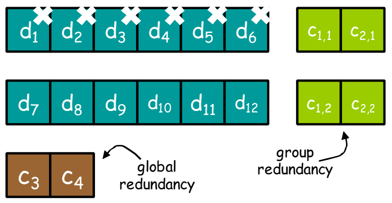
Theorem 1 *A basic Pyramid Code constructed from a (n, k) MDS code can recover $m = n - k$ arbitrary erasures (proof skipped, please refer to [10]).*

So far, we focus on Pyramid Codes with two-level hierarchy (global level and group level). Clearly, It should *not* be difficult to extend the results to multiple hierarchies. And regardless of levels, to achieve the best read performance, the decoding of a Pyramid Code will always start with the lowest level and gradually move to the global level. *This is very similar to climbing up a Pyramid, just as the name of the codes suggests.* (please see [10] for more details).

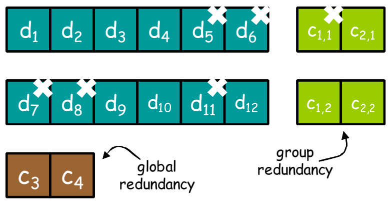
3 Generalized Pyramid Codes

In this section, we describe *generalized Pyramid Codes*, which are *not* trivial generalizations of the basic Pyramid Codes, but rather radically advanced new ERC schemes. They also go beyond the structure of the basic Pyramid Codes, where groups lower in the hierarchy are always nested in upper ones. In the generalized Pyramid Codes, groups may overlap with each other. Nevertheless, we use the common name *Pyramid Codes* to categorize both classes of codes, as they both aim at the same goal of trading storage space for access efficiency, and also follow the same failure recovery philosophy (i.e., the decoding scope is gradually broadened).

3.1 Motivation



(a) inherently unrecoverable



(b) unrecoverable as a basic Pyramid Code

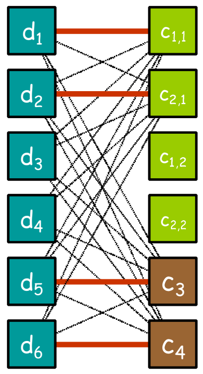
Figure 2. Examples to motivate the generalized Pyramid Codes (6 erasures each, marked by "x").

We use an example to explain the need to investigate beyond the basic Pyramid Codes. Figure 2 shows a configuration of a $(18, 12)$ basic Pyramid Code, which is constructed from a $(16, 12)$ MDS code. The code has 2 groups. Within each group, 6 data blocks are protected by 2 redundant blocks. Additionally, there are 2 global redundant blocks which protect the entire 12 data blocks. From the previous section, we know that the code can recover 4 arbitrary erasures. Since it has 6 redundant blocks, interesting questions to ask are: 1) what 5-erasure and 6-erasure patterns can it recover? Clearly, due to information theory limits, the code can *not* recover more than 6 erasures; and 2) more generally, can the recoverability be further improved?

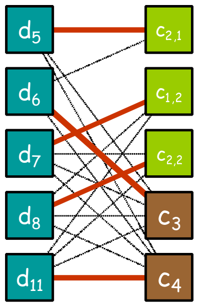
In particular, we examine the two erasure patterns. The first pattern has 6 erasures and is shown in Figure 2(a). There are 6 data blocks and 2 redundant blocks available in the second group. Hence, those 2 redundant blocks are *not* useful for recovery and can be removed. Now, we are left with 6 erasures, but only 4 redundant blocks. Therefore, this erasure pattern is inherently unrecoverable. The second erasure pattern is shown in Figure 2(b). It turns out that the basic Pyramid Code can *not* recover this pattern either. The decoding procedure can *not* make progress within each group, where there are more failed data blocks than available redundant blocks. Hence, it moves to the global level, where it computes one more redundant block $\mathbf{c}_2$ from $\mathbf{c}_{2,1}$ and $\mathbf{c}_{2,2}$. Still, on the global level, there are only 3 available redundant blocks and yet 5 data erasures. Hence, it can *not* proceed either. But, is there a way in which we can recover this pattern? In the following, we give a positive answer by presenting the generalized Pyramid Codes.

We first present a necessary condition of recoverability. Then, we describe the construction of the generalized Pyramid Codes, where the condition also becomes sufficient (i.e., the codes are optimal in failure recovery).

3.2 Necessary condition of recoverability



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



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

Figure 3. Tanner graphs (bold edges show maximum matchings).

The recoverability of *any* ERC scheme (not just Pyramid Codes) can be easily verified using a Tanner graph, which is a tool frequently used in the study of erasure resilience coding [13]. 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 corresponding redundant block is computed from the corresponding data block. Given an erasure pattern, a *simplified* Tanner graph (denoted as T) can be plotted to show only the *failed* data blocks and the *available* redundant blocks. For instance, the Tanner graphs corresponding to the erasure patterns in Figure 2 are shown in Figure 3.

Furthermore, we define *matching* (denoted by M) as a set of edges in a 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 to the number of data nodes, such a matching is called a *full-size matching* (denoted by M_f). For example, the Tanner graph in Figure 3(b) contains a full-size matching, while the one in Figure 3(a) does *not*. With these definitions, the necessary condition of recoverability is stated in the following theorem. (Note that when there is no ambiguity, blocks and nodes are used interchangeably, so as the recovery of an erasure pattern and the recover of a Tanner graph.)

Theorem 2 *For any linear ERC scheme (not just Pyramid Codes), an erasure pattern is recoverable* only if *the corresponding Tanner graph contains a full-size matching.*

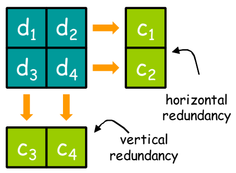
Proof We prove this theorem by contradiction. Examining an arbitrary *recoverable* erasure pattern, whose corresponding Tanner graph T consists of r_d data nodes and r_c redundant nodes. (Again, this means the erasure pattern has r_d failed data blocks and r_c available redundant blocks.) Obviously, $r_d \leq r_c$. Now, let's assume T does *not* contain a full-size matching. Then, the size of its maximum matching M_m is less than r_d, i.e., $|M_m| < r_d$. Based on the König-Egerváry Theorem [19] 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, i.e., $|N_c| = |M_m|$. 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 $n_d \leq |M_m| < r_d$.

Now let us assume all the data blocks in N_c are somehow known (not erasures any more), then we can deduce a new erasure pattern 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 less redundant nodes than the data nodes, and thus T' is not recoverable. Therefore, T should not be recoverable either, which contradicts with the assumption. $\square$

For interested readers, the same condition is also studied using an alternative set representation in a companion paper [3] (called *Maximally Recoverable* property there). Next, we present the generalized Pyramid Codes, which are optimal as the necessary condition also becomes sufficient.

3.3 Construction and optimality of generalized Pyramid Codes

We use an exemplary configuration (shown in Figure 4(a)) to briefly illustrate the construction of a generalized Pyramid Code. The code can be represented using a matrix form, as $\mathbf{C} = \mathbf{G} \times \mathbf{D}$, where $\mathbf{C} = [\mathbf{d}_1, \mathbf{d}_2, \mathbf{d}_3, \mathbf{d}_4, \mathbf{c}_1, \mathbf{c}_2, \mathbf{c}_3, \mathbf{c}_4]^T$, $\mathbf{D} = [\mathbf{d}_1, \mathbf{d}_2, \mathbf{d}_3, \mathbf{d}_4]^T$ and $\mathbf{G}$ is a *generator matrix* (shown in Figure 4(b)). The construction is to fill



(a) configuration

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

(b) generator matrix

Figure 4. A construction example.

all the non-zero entries of $\mathbf{G}$, such that the code is optimal in failure recovery.

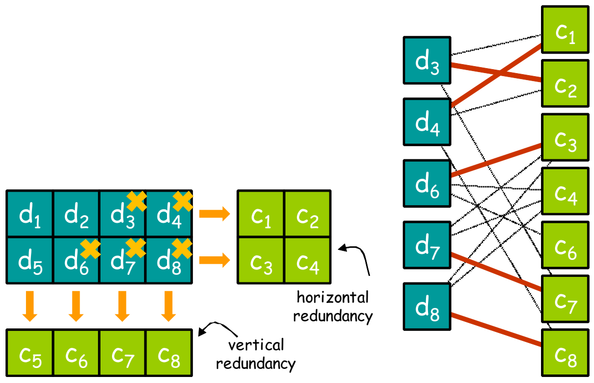
The algorithm works as follows. It starts with an identity matrix $\mathbf{G} = \mathbf{I}_{4\times4}$ and adds new rows (say $\mathbf{g}_i$) *one at a time*. The physical meaning of adding $\mathbf{g}_i$ is to define the computation of a new redundant block. To achieve optimal recovery, it is desirable that any failed data block should be recoverable from the new redundant block together with any other 3 blocks. Rows of $\mathbf{G}$ that correspond to such blocks form a submatrix, and the recoverability requires the submatrix to be invertible. This requires that $\mathbf{g}_i$ should be linearly independent of the other 3 rows (denote the corresponding subspace as $\mathbf{S}$). Hence, $\mathbf{g}_i$ should *not* be contained in $\mathbf{S}$. Or equivalently, $\mathbf{g}_i$ should *not* be orthogonal to the *null space* of $\mathbf{S}$ (a single vector here, as $\mathbf{S}$ has rank 3). Finally, the orthogonality boils down to require that the dot product of $\mathbf{g}_i$ and the null space vector to be non-zero. Due to space limitation, we only present the very high level concept here. Please refer to [10] for complete details. Similarly, the following theorem on the existence and optimality of the generalized Pyramid Codes is simply presented without proofs.

Theorem 3 *When the finite field size is greater than $\binom{n}{k-1}$, the construction of a generalized Pyramid Code is guaranteed. In such a code, any recoverable erasure pattern that is ever recoverable (i.e., its corresponding Tanner graph contains a full-size matching) can indeed be recovered.*

3.4 Optimal decoding

When block failures happen in ERC schemes, two types of recovery could be triggered: 1) recovery of all the failed blocks, including data and redundant blocks; and 2) recovery of a particular data block being actively accessed. Correspondingly, the *access overhead* can also be categorized into: 1) recovery overhead; and 2) read overhead.

Given an erasure pattern, we define an *access path* as a sequence of blocks to be accessed in order to recover the desirable blocks. Different access paths often bear different overheads. For instance, Figure 5(a) shows a configuration of a generalized Pyramid Code, as well as an erasure pattern with 5 failed data blocks. If it is desirable to recover data block $\mathbf{d}_6$, there are at least two viable access paths: 1) recover $\mathbf{d}_6$ directly from $\mathbf{d}_2$ and $\mathbf{c}_6$; or 2) first



(a) erasure pattern (b) Tanner graph

Figure 5. Decoding of generalized Pyramid Codes.

recover $\mathbf{d}_3$ from $\mathbf{d}_1$, $\mathbf{d}_2$, $\mathbf{c}_1$ and $\mathbf{c}_2$, then recover $\mathbf{d}_7$ from $\mathbf{d}_3$ and $\mathbf{c}_7$, and finally recover $\mathbf{d}_6$ from $\mathbf{d}_5$, $\mathbf{d}_7$, $\mathbf{c}_3$ and $\mathbf{c}_4$. Clearly, these two access paths have significantly different overheads. Similarly, if it is desirable to recover all the failed data blocks, there might also be a few access paths with different overheads. In this section, we describe algorithms to find access paths with either minimum recovery overhead or minimum read overhead. This is contrast to the basic Pyramid Codes, where finding access path with minimum overhead is straightforward, because decoding should always start from the lowest level in the hierarchy and gradually move up.

3.4.1 Minimum recovery overhead

Theorem 4 *In the generalized Pyramid Codes, to recover the failed blocks in an erasure pattern with d failed data blocks and c failed redundant blocks, the minimum access path will include exactly d available redundant blocks. It will also include every available data block, from which the failed redundant blocks are originally computed (proof skipped, please refer to [10]).*

Based on Theorem 4, it is straightforward to design a decoding algorithm with the minimum recovery overhead. Given any erasure pattern, we choose subsets of redundant blocks, such that the size of each subset simply equals to the number of failed data blocks. If the recovery can succeed (again, the corresponding Tanner graph contains a full-size matching), the recovery (data + redundant) overhead is computed. After enumerating through all the redundant subsets, the minimum recovery overhead can be readily derived. In practice, the number of available redundant blocks in the Tanner graph will not be many more than the number of failed data blocks, so the complexity of the algorithm should *not* be high. For instance, the Tanner graph in Figure 5(b) contains 7 redundant blocks and 5 failed data blocks, thus there are merely $\binom{7}{5} = 21$ subsets to compute.

3.4.2 Minimum read overhead

The recovery of a single data block in general requires smaller overhead than the recovery of all failed blocks, and their respective access paths could be rather different as well. An algorithm to find an access path with the minimum read overhead is described as follows.

Similar to the previous case, we choose subsets of the available redundant blocks, whose size equals to the total number of failed data blocks. If the corresponding Tanner graph does *not* contain a full-size matching, this subset is simply skipped. Otherwise, a breadth first search is carried out, starting from the target failed data block. If the search encounters a data node in the Tanner graph, it follows only the edge in the matching to the corresponding redundant node. If the search encounters a redundant node, it follows all edges in the Tanner graph to all the data nodes, which have *not* been visited before. The search stops when the set of visited redundant nodes is large enough to recover all the visited data nodes. After enumerating through all subsets, the minimum read overhead can be easily derived. The complexity is comparable to the previous case.

Very careful readers might challenge that given a redundant subset, there could exist more than one full-size matching in the Tanner graph (i.e., data and redundant nodes could be matched differently, while the sizes of matchings are the same). The breadth first search only explores one of them, which might happen to be *not* minimum. Nevertheless, the following theorem relieves such concern and the algorithm indeed guarantees to find the minimum read overhead (proof skipped, please refer to [10]).

Theorem 5 *In the generalized Pyramid Codes, given an erasure pattern and a redundant subset, the bread first search algorithm will always yield the same read overhead, even following different full-size matchings.*

3.5 Comparisons

This subsection compares the basic and generalized Pyramid Codes. We first use the same configuration, as shown in Figure 2. Both codes (denoted as *basic* and *gen. I*, respectively) are $(18, 12)$ codes and guarantee the recovery of 4 arbitrary failures. Figure 6 compares their recoverability beyond 4 failures, as well as the recovery and read overhead. It is quite obvious that the generalized Pyramid Code has higher recoverability when the number of failures exceeds 4. Moreover, this improvement of recoverability comes at the cost of increased read overhead (mostly prominent when there are 6 failures).

Next, we modify the configuration slightly and create a new generalized Pyramid Code, where the global redundant blocks are removed and replaced by $\mathbf{c}_3$: $\{\mathbf{d}_1, \mathbf{d}_2, \mathbf{d}_3, \mathbf{d}_7, \mathbf{d}_8, \mathbf{d}_9\}$ and $\mathbf{c}_4$: $\{\mathbf{d}_4, \mathbf{d}_5, \mathbf{d}_6, \mathbf{d}_{10}, \mathbf{d}_{11}, \mathbf{d}_{12}\}$. (Note that this configuration is *not* valid for a basic Pyramid Code, since horizontal and vertical groups now overlap.) The performance of this code is also shown in Figure 6. We observe that its recoverability is slightly reduced, as it no longer guarantees the recovery of 4 arbitrary erasures. On the other hand, both its recovery and read overhead are reduced as well. This again demonstrates the flexibility of Pyramid Codes in trading storage space for access efficiency, with different choices of configurations.

4 Additional related work

There are a few works, which bear a similar concept of trading storage space for access efficiency. For example, one scheme in [8] can improve the read overhead by using twice as much storage space as the data collection itself. [16] uses slightly more storage space than MDS codes to improve access efficiency in wide area storage networks. Compared to these schemes, Pyramid Codes are much more flexible and can explore a much wider range of the trade-offs. Moreover, the generalized Pyramid Codes have optimal recovery performance, while none of the other existing schemes does.

There are also significant efforts in trying to improve the encoding/decoding performance of ERC schemes. In particular, lots of them advocate using pure XOR operations, such as EVENODD [2], X-Code [22], RDP [5], codes based on CPM [6], etc. As mentioned before, if these codes are used to derive the basic Pyramid Codes, then all optimizations direct apply. As for the generalized Pyramid Codes, some generic optimization concepts, such as [17], are still applicable.

5 Concluding remarks

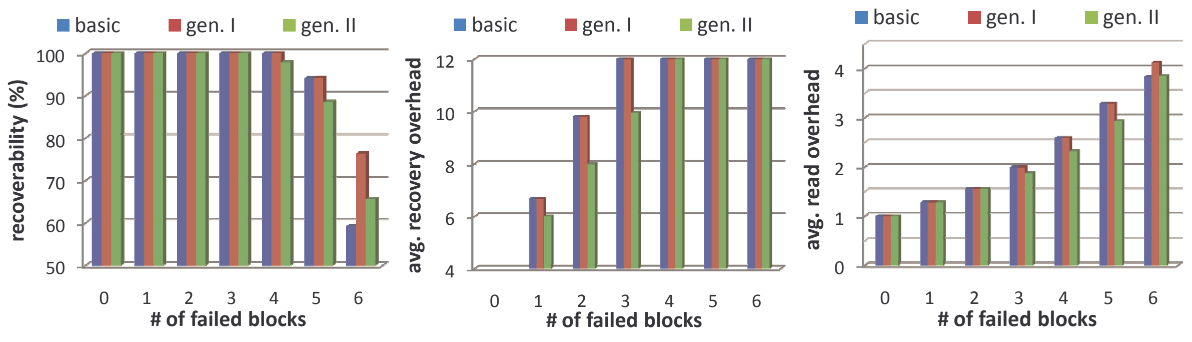
In this paper, we describe two classes of Pyramid Codes to allow flexible exploration of the trade-offs between storage space and access efficiency in reliable data systems. The basic Pyramid Codes can be simply derived from any existing codes, while the generalized Pyramid Codes are radically advanced new codes. We also establish a necessary condition of recoverability and show that the generalized Pyramid Codes are optimal in failure recovery.

References

[1] M. K. Aguilera, R. Janakiraman, and L. Xu, “Using erasure codes for storage in a distributed system”, *DSN 2005*, Yokohama, Japan, June. 2005.

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

[3] M. Chen, C. Huang, and J. Li, “’On the Maximally Recoverable Property for Multi-Protection Group



(a) recoverability (b) recovery overhead (c) read overhead

Figure 6. Comparisons of three $(18, 12)$ **Pyramid Codes.**
(*basic* stands for a basic Pyramid Code with the configuration shown in Figure 2; *gen. I* is a generalized Pyramid Code with the same configuration; and *gen. II* is a generalized Pyramid Code, where the global redundant blocks are removed and replaced by two group redundant blocks, covering $\{\mathbf{d}_1, \mathbf{d}_2, \mathbf{d}_3, \mathbf{d}_7, \mathbf{d}_8, \mathbf{d}_9\}$ and $\mathbf{c}_4 : \{\mathbf{d}_4, \mathbf{d}_5, \mathbf{d}_6, \mathbf{d}_{10}, \mathbf{d}_{11}, \mathbf{d}_{12}\}$, respectively.)

Codes”, (to appear) *ISIT 2007*, Nice, France, Jun. 2007.

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

[5] P. Corbett, B. English, A. Goel, T. Grcanac, S. Kleiman, J. Leong, and S. Sankar, “Row-Diagonal Parity for Double Disk Failure Correction”, *FAST 2005*, San Francisco, CA, Dec. 2005.

[6] G.-L. Feng, R. H. Deng, F. Bao, and J.-C. Shen, “New Efficient MDS Array Codes for RAID Part I: Reed-Solomon-Like Codes for Tolerating Three Disk Failures”, *IEEE Trans. on Computers*, 54(9), Sep. 2005.

[7] S. Ghemawat, H. Gobioff, and S.-T. Leung,“The Google File System”, *SOSP 2003*, Lake George, NY, October, 2003.

[8] J. L. Hafner, “WEAVER Codes: Highly Fault Tolerant Erasure Codes for Storage Systems”, *FAST 2005*, San Francisco, CA, Dec. 2005.

[9] J. Hamilton, “An Architecture for Modular Data Centers”, *CIDR 2007*, Jan. 2007

[10] C. Huang, M. Chen, and J. Li, “Pyramid Codes: Flexible Schemes to Trade Space for Access Efficiency in Reliable Data Storage Systems”, *Microsoft Research Technical Report MSR-TR-2007-25*, Mar. 2007.

[11] C. Huang, and L. Xu, “STAR: an Efficient Coding Scheme for Correcting Triple Storage Node Failures”, *FAST 2005*, San Francisco, CA, Dec. 2005.

[12] J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells, and B. Zhao, “OceanStore: an Architecture for Global-Scale Persistent Storage”, *ASPLOS 2000*, Cambridge, MA, Nov., 2000.

[13] S. Lin, and D. J. Costello, “Error Control Coding, Fundamentals and Applications”, Prentice Hall Press, 2004.

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

[15] J. S. Plank, “A tutorial on Reed-Solomon Coding for Fault-Tolerance in RAID-like Systems”, *Software – Practice & Experience*, 27(9), 995-1012, Sep. 1997.

[16] J. S. Plank, and M. G. Thomason, “A practical analysis of low-density parity-check erasure codes for wide-area storage applications”, *DSN 2004*, Florence, Italy, Jun. 2004.

[17] J. S. Plank, and L. Xu, “Optimizing Cauchy Reed-Solomon Codes for Fault-Tolerant Network Storage Applications,” *NCA 2006*, Cambridge, MA, Jul., 2006.

[18] I. S. Reed, and G. Solomon, “Polynomial Codes over Certain Finite Fields”, *J. Soc. Indust. Appl. Math.*, 8(10), 300-304, 1960.

[19] A. Schrijver, “Combinatorial Optimization, Polyhedra and Efficiency”, *Algorithms and Combinatorics*, Springer, vol. A, 2003.

[20] B. Schroeder, and G. A. Gibson, “Disk Failures in the Real World: What does an MTTF of 1,000,000 Hours Mean to You?”, *FAST 2007*, San Francisco, CA, Feb. 2007.

[21] J. Stribling, E. Sit, M. F. Kaashoek, J. Li, and R. Morris, “Don’t Give Up on Distributed File Systems”, *IPTPS 2007*, Bellevue, WA, Feb. 2007.

[22] L. Xu, and J. Bruck, “X-code: MDS array codes with optimal encoding”, *IEEE Trans. on Information Theory*, vol. 45, no. 1, 1999.