Leveled Product Codes for Optimal Block Repairs in Geo-distributed Storage Systems

Si Wu[1], Guantian Lin[2], Patrick P. C. Lee[3], Yinlong Xu[2]
[1]*Shandong University* [2]*University of Science and Technology of China*
[3]*The Chinese University of Hong Kong*

***Abstract*—To provide fault tolerance with low storage overhead, modern geo-distributed storage systems use erasure coding to stripe data redundancy across geographical regions. The prevalence of node, rack, and region failures motivates the needs for both single-block and multi-block repairs, yet block repairs trigger substantial cross-rack and cross-region data transfers. We propose a new family of erasure codes, Leveled Product Codes (LPCs), by adapting the classical Product Codes designed for disk arrays into geo-distributed storage systems. LPCs localize single-block repairs within racks and optimize multi-block repairs with the minimum sum of cross-rack and cross-region data transfers, while providing fault tolerance against node, rack, and region failures. We theoretically prove the optimality of LPCs, and further implement LPCs in a distributed storage prototype. Our numerical analysis and testbed evaluation show that LPCs significantly reduce the single-block and multi-block repair times of state-of-the-art hierarchy-aware erasure codes.**

I. Introduction

Geo-distributed storage systems store massive amounts of data across geographical regions to provide proximal and scalable user access, while ensuring data availability even in the face of catastrophic failures such as earthquakes and power outages. To safeguard data storage against node and rack failures within a region, as well as region-wide failures, modern geo-distributed storage systems employ *erasure coding* to stripe low-cost redundancy across geographical regions [2], [6], [11], [21], [28]. At a high level, an erasure code encodes a group of data blocks into parity blocks, in which the data and parity blocks collectively form a *stripe*, such that any subset of a sufficient number of data and parity blocks within a stripe can reconstruct all original data blocks. Compared to replication, erasure coding significantly reduces the amount of redundancy, while offering much higher reliability measured by mean-time-to-data-loss [35]. This makes erasure coding particularly well-suited for exascale data management in geo-distributed storage systems.

To maintain data availability, storage systems frequently perform data repair operations due to the prevalence of node, rack, and region failures [9], [25], [29]. In traditional erasure-coded storage, a majority of stripes with failures (e.g., more than 98%) experience a single failed block [25], and hence existing erasure codes often focus on optimizing single-block repairs (e.g., [7], [15], [29]). However, in catastrophic events [9] or zone maintenance [16], rack and region failures can occur, thereby causing multiple blocks to become unavailable (such failed blocks can belong to the same stripe). Also, wide stripes, which comprise a large number of data blocks and a small number of parity blocks, are increasingly studied and deployed to achieve ultra-low storage redundancy [1], [4], [12], [16], and the likelihood of having multiple failed blocks in wide stripes is non-negligible [12], [16]. Thus, both single-block and multi-block repairs are critical in large-scale erasure-coded storage and should be simultaneously optimized.

Achieving optimal single-block and multi-block repairs for erasure-coded storage is non-trivial, as they trigger substantial network transfers in order to retrieve multiple available blocks of the same stripe for reconstruction of any failed blocks. The repair penalty becomes more prominent in geo-distributed storage systems, in which substantial cross-rack and cross-region data transfers are necessary for block repairs. Even though the literature has proposed various repair-efficient erasure codes, such as Regenerating Codes [7], Locally Repairable Codes (LRCs) [14], [15], [17], [29], and hierarchy-aware erasure codes for rack-based data centers [12], [13], [16], [31], [34], [37], they still fall short in optimizing both single-block and multi-block repairs for geo-distributed storage systems with nodes, racks, and regions (see Section III-B for details).

We propose a new family of erasure codes, namely *Leveled Product Codes (LPCs)*, to achieve optimal single-block and multi-block repairs in geo-distributed storage systems. LPCs build on the classical Product Codes [10], [19], which are designed for disk arrays, and adapt them into geo-distributed storage systems. An LPC stripe comprises a two-dimensional logical array of blocks encoded by row-based and column-based codes. It incorporates (i) a node-level column-based code to localize single-block repairs within a rack, (ii) a rack-level row-based code to tolerate rack failures while optimizing multi-block repairs with the minimum sum of cross-rack and cross-region transfers, and (iii) a region-level data placement strategy to tolerate region failures. To summarize, our paper makes the following contributions:

- We design the code construction and data placement for LPCs in geo-distributed storage systems. In particular, to minimize the sum of cross-rack and cross-region transfers in multi-block repairs, LPCs adopt a progressive approach by first repairing only a subset of failed blocks, followed by locally repairing remaining failed blocks within a rack.
- We formally prove the optimality guarantees for single-block and multi-block repairs of LPCs; note that the optimal multi-block repairs also apply to rack and region repairs.
- We implement a distributed storage prototype to realize our LPCs as well as the state-of-the-art erasure codes, including

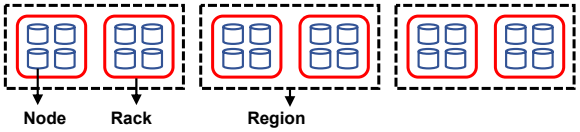


Figure 1. Geo-distributed storage system; a rectangle with black-dotted lines indicates a region and a red rectangle means a rack.

hierarchy-aware RS Codes and LRCs as well as Multi-Level Erasure Codes (MLEC) [34].

- We conduct experiments on a local cluster that simulates up to 240 nodes. LPCs reduce the single-block repair times by up to 90.6%, 80.8%, and 62.8%, and the multi-block repair times by up to 70.0%, 73.2%, and 57.5%, compared with hierarchy-aware RS Codes, hierarchy-aware LRCs, and MLEC, respectively, while all codes share similar storage overhead with up to 11.1% differences.

The source code of our prototype is available at: **https://github.com/hhlgt/leveled-product-codes**.

II. Background

A. Geo-distributed Storage Systems

We consider a geo-distributed storage system with a hierarchical topological structure as observed in production (e.g., Facebook f4 [21] and Microsoft Giza [6]), as shown in Figure 1. It organizes multiple *nodes* in a *rack*, and further organizes multiple racks in a *region*, such that the entire storage system spans a number of nodes, racks, and regions. Cross-rack data transfers are typically much slower than inner-rack data transfers due to limited cross-rack bandwidth [5], [33], and cross-region data transfers are also slow as they traverse different physical areas. Thus, both cross-rack and cross-region transfers are the performance bottlenecks in data accesses.

B. Erasure Coding

Our work focuses on two widely deployed families of erasure codes: Reed-Solomon (RS) Codes [27] and Locally Repairable Codes (LRCs) [14], [15], [17], [29], both of which have also been applied to hierarchical topologies (Section III).

RS Codes. An RS Code, denoted by $\mathrm{RS}(k,m)$, is configured by two parameters k and m. $\mathrm{RS}(k,m)$ encodes k original data blocks (denoted by $D_1, D_2, \cdots, D_k$) into m additional *parity blocks* (denoted by $P_1, P_2, \cdots, P_m$), such that any k out of the $k+m$ blocks can recover all k original data blocks; in other words, it tolerates the failures of any m out of the $k+m$ blocks. The $k+m$ blocks that are encoded together collectively form a *stripe*. A geo-distributed storage system stores many stripes that are independently encoded. Figure 2 shows an example of RS(6,3) with $k=6$ data blocks and $m=3$ parity blocks.

The encoding and decoding of RS Codes are based on the arithmetic in Galois Field $GF(2^w)$ in w-bit words [22]. Each data/parity block (say B) in $\mathrm{RS}(k,m)$ is encoded or decoded by computing a linear combination of k data/parity blocks (say $B_1, B_2, \cdots, B_k$) of the same stripe as $B = \sum_{i=1}^{k} \alpha_i B_i$ for some coding coefficients α_i's, where additions are bitwise-XORs. Linear combinations satisfy *additive associativity*, meaning that the terms $\alpha_i B_i$'s can be grouped in arbitrary order in additions.

RS Codes minimize the storage redundancy (i.e., $1+\frac{m}{k}$ times the original data size) to tolerate the loss of any m out of $k+m$ blocks; this property is called *maximum distance separable (MDS)*. However, RS Codes have high repair penalty: the repair of any lost block in $\mathrm{RS}(k,m)$ needs to retrieve k surviving blocks of the same stripe (i.e., k times the block size). For example, in Figure 2, the repair of D_1 in $\mathrm{RS}(6,3)$ needs to retrieve D_2, D_3, D_4, D_5, D_6, and P_1.

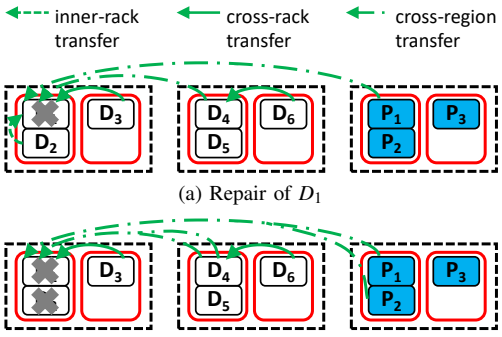
LRCs. An LRC, denoted by $\mathrm{LRC}(k,l,g)$, is configured by three parameters k, l, and g. $\mathrm{LRC}(k,l,g)$ encodes k data blocks into l *local parity blocks* (denoted by $L_1, L_2, \cdots, L_l$) and g *global parity blocks* (denoted by $G_1, G_2, \cdots, G_g$), such that the $k+l+g$ blocks form a stripe. There are various code constructions for LRCs [15]–[17], [29], among which Azure's Local Reconstruction Codes [15] can tolerate the loss of most blocks under the same storage redundancy [12]. In this work, we focus on Azure's code construction for $\mathrm{LRC}(k,l,g)$. Specifically, suppose that k is divisible by l. $\mathrm{LRC}(k,l,g)$ divides k data blocks evenly into l local groups and computes a local parity block based on bitwise-XORs of the $\frac{k}{l}$ data blocks in each local group. It further encodes all k data blocks into g global parity blocks as in RS Codes [27]. For example, Figure 3 shows an example of LRC(6,2,2) with $k=6$ data blocks, $l=2$ local parity blocks, and $g=2$ global parity blocks.

LRCs exploit *stripe locality* to improve repair efficiency. In $\mathrm{LRC}(k,l,g)$, the repair of a lost data block or local parity block only retrieves the remaining $\frac{k}{l}$ surviving blocks within the same local group, while the repair of a lost global parity block still retrieves k out of the $k+g-1$ surviving data blocks and global parity blocks. For example, in Figure 3, the repair of D_1 in LRC(6,2,2) retrieves D_2, D_3, and L_1, while the repair of G_1 retrieves $k=6$ blocks from D_1 to D_6, and G_2.

Other erasure codes. Some erasure codes have been proposed to mitigate the repair traffic of RS Codes, such as Regenerating Codes [7] and Piggybacking Codes [26], by performing block repairs at the granularity of sub-blocks. However, they incur non-contiguous I/Os and may negate the repair efficiency [32]. Also, their constructions do not address hierarchical topologies.

C. Fault Tolerance Requirements

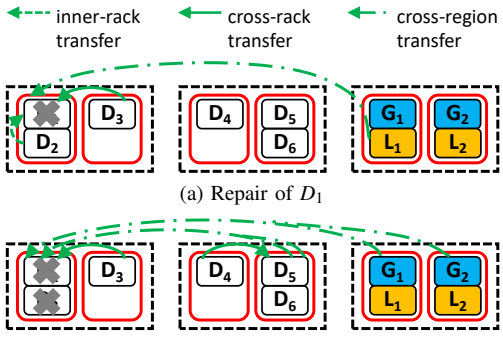
A geo-distributed storage system should place the blocks of a stripe across distinct nodes for *multi-node fault tolerance*. It should also tolerate whole-region failures in case of catastrophic events. As whole-region failures happen much less rarely than node failures in practice [21], we focus on *single-region fault tolerance* (a similar concept, called single-cluster fault tolerance, is found in prior studies [21], [36], [37]). As a region comprises multiple racks, single-region fault tolerance also implies *multi-rack fault tolerance*. Under single-region fault tolerance, $\mathrm{RS}(k,m)$ can place up to m blocks of an RS stripe in one region, while $\mathrm{LRC}(k,l,g)$ can place up to $g+j$ $(1 \le j \le l)$ blocks of an LRC stripe that span j local groups in one region [37] since the number of local/global parity blocks that can be used for decoding is $g+j$. For example, in Figure 2, RS(6,3) places three blocks in one region (i.e., three regions in total for an RS stripe), while in Figure 3, LRC(6,2,2) places every



(a) Repair of D_1

(b) Repair of D_1 and D_2

Figure 2. Block repairs in RS$(6,3)$. The repair of D_1 (figure (a)) transfers two cross-rack blocks and two cross-region blocks. The repair of D_1 and D_2 (figure (b)) transfers two cross-rack blocks and four cross-region blocks.



(a) Repair of D_1

(b) Repair of D_1 and D_2

Figure 3. Block repairs in LRC$(6,2,2)$. The repair of D_1 (figure (a)) transfers one cross-rack block and one cross-region block. The repair of D_1 and D_2 (figure (b)) transfers two cross-rack blocks and four cross-region blocks.

$g+1=3$ data blocks in one region and all parity blocks in a separate region (i.e., three regions in total for an LRC stripe). Both examples achieve single-region fault tolerance.

III. Data Repair Problem

A. Problem Statement

In this work, we focus on both single-block and multi-block repairs, which reconstruct single and multiple failed blocks of a stripe, respectively, by retrieving other available blocks of the same stripe. We assume that cross-rack and cross-region transfers are the performance bottlenecks in repairs (Section II-A). Our goal is to *minimize the sum of the amounts of traffic in cross-rack and cross-region transfers for a repair*.

Note that other design goals may be considered. One possible goal is to minimize the *weighted* sum of cross-rack and cross-region transfers, by accounting for the more limited cross-region bandwidth than the cross-rack bandwidth and assigning a higher weight to cross-region transfers than cross-rack transfers. Another possible goal is to minimize the maximum load of cross-rack (or cross-region) transfers when the cross-rack (or cross-region) bandwidth is heterogeneous. We pose the analysis of different goals as future work.

Failures can occur at the node, rack, and region levels. To mitigate the risk of encountering additional failures that lead to data loss, any failed node, rack, or region should be repaired as fast as possible. For a single node failure, its repair can be viewed as a set of single-block repairs, each of which reconstructs the failed block of a stripe that covers the failed node. Similarly, for a failed rack or region, their repairs can be viewed as a set of single-block and multi-block repairs across all affected stripes. Thus, our work on optimizing single-block and multi-block repairs can be applied to the repair of a failed node, rack, and region.

B. Limitations of Existing Hierarchy-aware Erasure Codes

There exist erasure coding designs with hierarchical awareness in the literature [12], [13], [16], [31], [37]. However, we argue that they still cannot perform efficient repairs in geo-distributed storage systems. Specifically, they cannot localize block repairs within racks and will trigger substantial cross-rack and cross-region transfers for block repairs, due to the lack of stripe locality in the code construction or improper dispersion of a local group of blocks among racks.

Limitations of hierarchy-aware RS Codes. CAR [31] and DoubleR [13] extend RS Codes [27] and Minimum-storage Regenerating (MSR) Codes [7] with hierarchical awareness, while maintaining the MDS property (i.e., minimum storage redundancy). We use RS Codes as an example to explain how hierarchical awareness is incorporated. The key idea is to place every m blocks of a stripe in a region so as to limit the number of regions spanned by the stripe, while maintaining single-region fault tolerance (if a region contains more than m blocks, a failed region will cause data loss). As a region comprises multiple racks, we assume that the m blocks are randomly distributed to the racks in a region. Cross-region transfers are provably minimized by exploiting the additive associativity of RS Codes (Section II-B) to perform *partial decoding* on the surviving blocks in each region (i.e., adding parts of terms of a linear combination in decoding) [13], [31]. As RS Codes lack stripe locality, both single-block and multi-block repairs always retrieve k surviving blocks for decoding, and inevitably incur cross-region transfers for $k \geq m$.

Figure 2 shows the block repairs of RS$(6,3)$. In Figure 2(a), the repair of D_1 retrieves $k=6$ blocks. As there is one block D_2 within the same rack as D_1, the repair of D_1 needs to retrieve D_3 from another rack, and D_4, D_5, D_6, and P_1 from other regions. Here, we can apply partial decoding to D_4, D_5, and D_6, in which we transfer D_6 to the rack that stores D_4 and D_5, and then compute and transfer a partially decoded block to the region that stores D_1. The repair of D_1 transfers two cross-rack blocks and two cross-region blocks. In Figure 2(b), the repair of both D_1 and D_2 still needs $k=6$ blocks. This leads to two cross-rack blocks and four cross-region blocks. Here, we apply partial decoding to D_4, D_5, and D_6 to transfer two partially decoded blocks to repair two failed blocks.

Limitations of hierarchy-aware LRCs. LRCs incorporate stripe locality, in which the repair of a data block or local parity block retrieves only $\frac{k}{l}$ blocks with the same local group and incurs less repair traffic than RS Codes. Prior studies [12], [37] place every $g+1$ blocks of each local group in one region to minimize the number of regions spanned by each local group while maintaining single-region fault tolerance [37].

Furthermore, all global parity blocks are placed in a dedicated region for fault tolerance. Here, we again assume the random placement of the blocks to the racks within a region.

If $\frac{k}{l} > g$, then each local group spans more than one region. Thus, the repair of a single data block or local parity block incurs cross-region transfers. Also, the repair of a single global parity block retrieves k blocks from the surviving data blocks and global parity blocks. The same issue holds for the repair of multiple blocks within the same local group, as the decoding within a local group is no longer feasible. Thus, cross-region transfers are inevitable as the k blocks span multiple regions.

Figure 3 shows the block repairs of LRC$(6,2,2)$. In Figure 3(a), the repair of D_1 retrieves $\frac{k}{l} = 3$ blocks from the same local group. As D_2 resides in the same rack as D_1, the repair of D_1 accesses D_3 from another rack and L_1 from another region. In Figure 3(b), the repair of both D_1 and D_2 retrieves $k = 6$ surviving blocks. Similar to the multi-block repair in RS Codes in Figure 2(b), the repair of D_1 and D_2 transfers two cross-rack blocks and four cross-region blocks.

IV. Leveled Product Codes

A. Main Idea

We design a new family of erasure codes called *Leveled Product Codes (LPCs)*), which incorporate hierarchical awareness and stripe locality. LPCs comprise three levels: (i) *node level*: LPCs realize a column-based RS Code across nodes inside a rack to localize single-block repairs and tolerate node failures; (ii) *rack level*: LPCs realize a row-based RS Code across racks to optimize multi-block repairs and tolerate rack failures; (iii) *region level*: LPCs realize region-aware data placement for single-region fault tolerance. LPCs localize single-block repairs inside racks. Also, LPCs decompose multi-block repairs into a minimum number of *row repairs* and a maximum number of *column repairs*, in which row repairs are done by retrieving only a subset of blocks, so that column repairs are later done locally within racks.

LPCs adapt Product Codes [10], [19] into geo-distributed storage systems. We address two key challenges: (i) how to design the data placement of LPCs in geo-distributed storage systems so as to provide fault tolerance against node, rack, and region failures, and (ii) how to provide optimality guarantees for both single-block and multi-block repairs.

B. Construction

Definition. We construct an LPC with five parameters k_1, m_1, k_2, m_2, and r, denoted by LPC(k_1,m_1,k_2,m_2,r). It comprises RS(k_1,m_1) in the row direction and RS(k_2,m_2) in the column direction (note that RS Codes can be replaced by any MDS codes). Thus, an LPC stripe forms a $(k_2+m_2)\times(k_1+m_1)$ two-dimensional logical array with storage redundancy $(1+\frac{m_1}{k_1})(1+\frac{m_2}{k_2})$. Each LPC stripe is distributed across r regions. The LPC construction comprises three steps.

Step 1. We organize k_1k_2 data blocks, denoted by D_1, D_2, $\cdots$, $D_{k_1k_2}$, into a $k_2\times k_1$ logical array in column-major order. We encode each of the k_1 columns of k_2 data blocks to form m_2 *column parity blocks*, and label all k_1m_2 column parity blocks (i.e., a $m_2\times k_1$ logical array) by $C_1,C_2,\cdots,C_{k_1m_2}$ in column-major order.

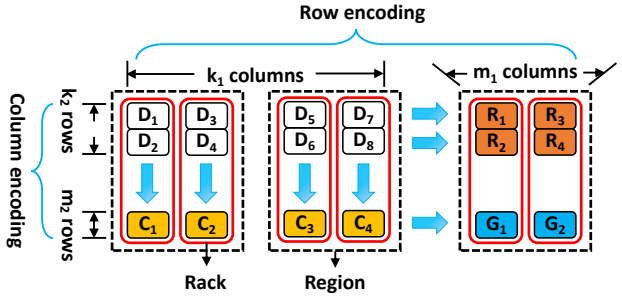


Figure 4. Construction of LPC(k_1,m_1,k_2,m_2,r), where $k_1=4$, $m_1=2$, $k_2=2$, $m_2=1$, and $r=3$.

Step 2. We encode each of the k_2 rows of k_1 data blocks to form m_1 *row parity blocks*, and label all k_2m_1 row parity blocks (i.e., a $k_2\times m_1$ logical array) by $R_1,R_2,\cdots,R_{k_2m_1}$ in column-major order. We also encode each of the m_2 rows of k_1 column parity blocks to form m_1 *global parity blocks*, and label all m_1m_2 global parity blocks (i.e., a $m_2\times m_1$ logical array) by $G_1,G_2,\cdots,G_{m_1m_2}$ in column-major order.

Step 3. We distribute each column of k_2+m_2 blocks into k_2+m_2 distinct nodes, where each column resides in a distinct rack. We also co-locate every m_1 of the k_1+m_1 racks in one region, so an LPC stripe is stored in $\lceil\frac{k_1}{m_1}\rceil+1$ regions.

Figure 4 shows an LPC$(4,2,2,1,3)$ stripe with $k_1k_2=8$ data blocks, $m_1=2$ row parity blocks for each row of data blocks, and $m_2=1$ column parity block for each column of data blocks. The stripe spans $r=3$ regions.

While each global parity block is generated by encoding the column parity blocks in the same row using the row-based RS Code, Theorem 1 shows that it can also be generated by encoding the row parity blocks in the same column using the column-based RS Code.

Theorem 1. *Each column of an LPC(k_1,m_1,k_2,m_2,r) is an RS(k_2,m_2) stripe.*

Proof. Without loss of generality, we focus on the generation of the global parity block G_1. Each row parity block R_i $(1\le i\le k_2)$ is generated by RS(k_1,m_1) as $R_i=\sum_{j=1}^{k_1}\alpha_jD_{(j-1)k_2+i}$, where α_j's $(1\le j\le k_1)$ denote the encoding coefficients of RS(k_1,m_1). Also, we have $G_1=\sum_{j=1}^{k_1}\alpha_jC_{(j-1)m_2+1}$.

We examine the first row of column parity blocks C_1, C_{m_2+1}, C_{2m_2+1}, $\cdots$, $C_{(k_1-1)m_2+1}$. Each column parity block $C_{(j-1)m_2+1}$ $(1\le j\le k_1)$ is generated by RS(k_2,m_2) as $C_{(j-1)m_2+1}=\sum_{i=1}^{k_2}\beta_iD_{(j-1)k_2+i}$, where β_i's $(1\le i\le k_2)$ denote the encoding coefficients of RS(k_2,m_2).

We can rewrite G_1 as $G_1=\sum_{j=1}^{k_1}\alpha_j\sum_{i=1}^{k_2}\beta_iD_{(j-1)k_2+i}=\sum_{i=1}^{k_2}\beta_i\sum_{j=1}^{k_1}\alpha_jD_{(j-1)k_2+i}=\sum_{i=1}^{k_2}\beta_iR_i$. This implies that G_1 can be generated from R_i's $(1\le i\le k_2)$ using RS(k_2,m_2). □

C. Fault Tolerance

We analyze the fault tolerance guarantees of LPCs.

Lemma 1. *LPC(k_1,m_1,k_2,m_2,r) can tolerate any m_1 failed blocks in each row or any m_2 failed blocks in each column.*

Proof. For each row (column) of an LPC (k_1, m_1, k_2, m_2, r) stripe, if there are up to m_1 (m_2) failed blocks, the failed blocks can be repaired by retrieving k_1 (k_2) surviving blocks from the same row (column) via RS decoding. □

Lemma 1 implies that LPC(k_1, m_1, k_2, m_2, r) can tolerate the failed blocks that span no more than m_2 rows or m_1 columns.

Lemma 2. *LPC(k_1, m_1, k_2, m_2, r) can tolerate any $f = im_1 + jm_2 - m_1m_2$ block failures that span i ($m_2 < i \leq k_2 + m_2$) rows and j ($m_1 < j \leq k_1 + m_1$) columns if and only if the failed blocks do not contain any $(m_2+1) \times (m_1+1)$ block matrix.*

Proof. **If part:** We first consider that there is no $(m_2+1) \times (m_1+1)$ block matrix formed among the failed blocks. Suppose that among the existing failed blocks, there are x rows with $m_1 < f_1, f_2, \cdots, f_x \leq j$ failed blocks, and $i-x$ rows with $0 < f_{x+1}, f_{x+2}, \cdots, f_i \leq m_1$ failed blocks. Since the failed blocks in the last $i-x$ rows can be repaired based on Lemma 1, we focus on the fault tolerance of the first x rows.

We consider two cases. If $x \leq m_2$, which means that each column has no more than m_2 failed blocks, then each column in the x rows can be repaired by column-based RS decoding. If $x > m_2$, since there is no $(m_2+1) \times (m_1+1)$ block matrix among the failed blocks, then there are at most m_1 columns that have more than m_2 (but no more than x) failed blocks each, while the remaining columns have no more than m_2 failed blocks. For the former columns, they can be repaired by row-based RS decoding as there are at most m_1 failed blocks in each row, while for the latter columns, they can be repaired by column-based RS decoding.

Based on the above arguments, we compute the maximum number of tolerable failed blocks. If $x \leq m_2$, the number of tolerable failed blocks is maximized at $x = m_2$. Then, we have $f_1 = \cdots = f_x = j$ and $f_{x+1} = \cdots = f_i = m_1$, meaning the maximum number of tolerable failed blocks is $\sum_{h=1}^{i} f_h = im_1 + jm_2 - m_1m_2$. If $x > m_2$, then among the first x rows, the columns with more than m_2 failed blocks have at most m_1x failed blocks in total, while the remaining columns have at most $(j-m_1)m_2$ failed blocks in total. Among the remaining $i-x$ rows, there are at most $(i-x)m_1$ failed blocks in total. Thus, the maximum number of tolerable failed blocks is also $im_1 + jm_2 - m_1m_2$.

Only-if part: If there is a $(m_2+1) \times (m_1+1)$ block matrix among the failed blocks, then the failed blocks in the block matrix cannot be repaired from neither the row-based RS Code nor the column-based RS Code. □

Remarks. Product Codes with parameters k_1, m_1, k_2, and m_2 are shown to tolerate any $m_1m_2 + m_1 + m_2$ failed blocks [19]. This can be derived by setting $i = m_2 + 1$ and $j = m_1 + 1$ in Lemma 2, where $m_1m_2 + m_1 + m_2$ failed blocks do not contain any $(m_2+1) \times (m_1+1)$ block matrix. Lemma 2 covers more general scenarios than the findings in [19] by proving fault tolerance against more than $m_1m_2 + m_1 + m_2$ failed blocks. Our findings are critical for analyzing fault tolerance in geo-distributed storage systems with a much larger number of failed blocks (under rack or region failures) in catastrophic events.

Algorithm 1 Progressive multi-block repair

Input: A set of failed blocks
Output: Repair policy for the failed blocks
1: Check reparability from Lemma 2;
2: **if** the failed blocks cannot be repaired **then**
3: Return "data loss";
4: **end if**
5: **while** there exist failed blocks **do**
6: // Execute as many column repairs as possible
7: **for** all columns with no more than m_2 failed blocks **do**
8: Execute column repairs in parallel;
9: **end for**
10: **if** some rows have no more than m_1 failed blocks **then**
11: // Execute one row repair
12: Select the row with maximum number of failed blocks;
13: Execute one row repair;
14: **end if**
15: **end while**

Lemma 3. *LPC(k_1, m_1, k_2, m_2, r) tolerates m_1 rack failures.*

Proof. If m_1 racks fail, there are $k_2 + m_2$ rows of m_1 failed blocks. From Lemma 1, the failed blocks can be repaired. □

Lemma 4. *LPC(k_1, m_1, k_2, m_2, r) tolerates a region failure.*

Proof. If a region fails, the m_1 racks in this region fail. From Lemma 3, the failed region can be repaired. □

D. Block Repairs

We describe the details of single-block and multi-block repairs in LPCs. We finally prove their optimality.

Single-block repair. As LPCs place each column of blocks inside a rack, any single-block repair and the repair of up to m_2 failed blocks within the same column can be done by retrieving any k_2 surviving blocks in the same column (i.e., same rack) without any cross-rack or cross-region transfer. For example, in Figure 5(a), the repair of D_1 accesses D_2 and C_1 within a rack.

Multi-block repair. Conventional wisdom suggests that if there are more than m_2 failed blocks within the same column, a column repair is infeasible. Nevertheless, we develop a *progressive* approach to allow LPCs to perform a multi-block repair even with more than m_2 failed blocks in the same column. Specifically, LPCs execute row repairs to decode some failed blocks across the affected columns, so that column repairs can locally decode the remaining failed blocks within racks. As a column repair can be executed completely inside a rack while a row repair involves block accesses across racks and regions, our idea is to execute the maximum possible number of column repairs and the minimum possible number of row repairs in a multi-block repair. Such a progressive multi-block repair approach *provably* minimizes the sum of cross-rack and cross-region transfers.

Algorithm 1 shows the steps of a progressive multi-block repair. We first check if a set of failed blocks can be repaired based on Lemma 2 and return "data loss" if the set of failed blocks cannot be repaired (Lines 1-4). We next find all columns

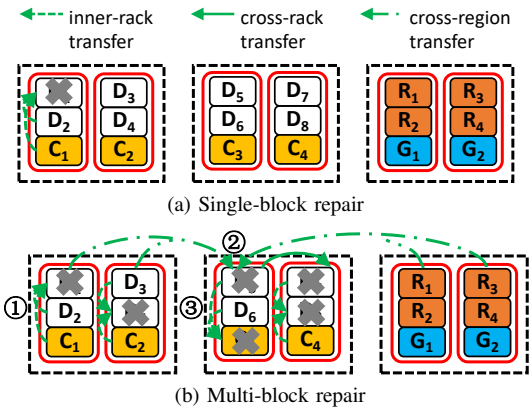


Figure 5. Block repairs in LPC$(4,2,2,1,3)$. The single-block repair of D_1 (figure (a)) is done within a rack. The multi-block repair of D_1, D_4, D_5, D_7, D_8, C_3 (figure (b)) transfers one cross-rack block and four cross-region blocks.

with no more than m_2 failed blocks and execute column repairs in parallel (Lines 6-9). We then find the rows with no more than m_1 failed blocks, select the row with the maximum number of failed blocks, and execute a row repair on the selected row (Lines 10-14). We repeat the executions of column repairs and a row repair until all failed blocks are repaired.

We elaborate on the details of a row repair, assuming that there are f failed blocks in the row:

- *Step 1: Selecting the main region and the main rack.* We randomly select a region and a rack therein as the *main region* and the *main rack*, respectively.
- *Step 2: Selecting helper regions.* We select the smallest number of helper regions, such that the total number of surviving blocks in this row in all helper regions and the main region is at least k_1.
- *Step 3: Transferring blocks for decoding.* We first consider $f = 1$. Then, for each helper region, if the number of surviving blocks is one, then we directly transfer the surviving block to the main rack; otherwise, we select the rack that stores at least one surviving block as the *relay*, which performs partial decoding and transfers a partially decoded block [12], [13], [31], [36], [37]. Specifically, the relay collects other surviving blocks across racks within the same region, computes a partially decoded block, and transfers the partially decoded block to the main rack. If $f > 1$, then for each helper region, we directly transfer the surviving blocks to the main rack.
- *Step 4: Decoding.* We decode all failed blocks in this row in the main rack.
- *Step 5, Redistributing decoded blocks.* We redistribute the decoded blocks from the main rack to other racks and regions.

We now analyze the cross-rack and cross-region transfers of a row repair. To simplify our analysis, we assume that k_1 is divisible by m_1, such that each region stores exactly m_1 blocks in a row. In the main region, suppose that there are x failed blocks and $m_1 - x$ surviving blocks; in other words, in the helper regions, there are $f - x$ failed blocks.

- (i) We start with $f > 1$. Suppose that the main rack contains one failed block. Then, we access $m_1 - x$ surviving blocks across racks in the main region and $k_1 - m_1 + x$ surviving blocks across regions from the helper regions for decoding. After decoding, we re-distribute $x - 1$ decoded blocks across racks in the main region and $f - x$ decoded blocks to other regions. Thus, the numbers of cross-rack and cross-region transfers are $m_1 - 1$ and $k_1 - m_1 + f$ blocks, respectively. Note that we can obtain the same numbers even if the main rack contains one surviving block. This implies that the random selection of the main region and main rack has no impact on the numbers of cross-rack and cross-region transfers.
- (ii) From (i), the sum of cross-rack and cross-region transfers is fixed as $k_1 + f - 1$ blocks. We now explain why we exclude partial decoding for $f > 1$. In Step 3, suppose there are $s > f$ surviving blocks in one helper region. In our design, we directly transfer the s blocks across regions. However, if we apply partial decoding, then the relay accesses $s - 1$ blocks across racks, computes and transfers f blocks corresponding to the failed blocks. This means that the number of cross-region blocks decreases by $s - f$, but the number of cross-rack blocks increases by $s - 1$, i.e., the sum of cross-rack and cross-region transfers increases.
- (iii) We now consider $f = 1$. There are $\frac{k_1}{m_1} - 1$ helper regions with m_1 surviving blocks (and a last helper region with one surviving block). In Step 3, in each such helper region, the number of cross-region blocks decreases by $m_1 - 1$ and the number of cross-rack blocks increases by $m_1 - 1$ by applying partial decoding (i.e., the sum keeps unchanged) From (i), the numbers of cross-rack and cross-region blocks are $k_1 - \frac{k_1}{m_1}$ and $\frac{k_1}{m_1}$, respectively.

Thus, the numbers of cross-rack and cross-region blocks are

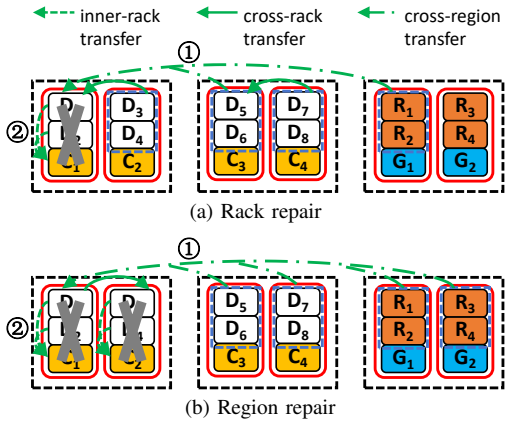
$$\#(\text{cross-rack blocks}) = \begin{cases} k_1 - \frac{k_1}{m_1}, & \text{if } f = 1; \\ m_1 - 1, & \text{if } f > 1, \end{cases} \quad (1)$$

$$\#(\text{cross-region blocks}) = \begin{cases} \frac{k_1}{m_1}, & \text{if } f = 1; \\ k_1 - m_1 + f, & \text{if } f > 1, \end{cases} \quad (2)$$

where f is the number of failed blocks in a row.

If k_1 is not divisible by m_1, the sum of cross-rack and cross-region transfers stays unchanged (i.e., $k_1 + f - 1$) and is not affected by the number of blocks in a region. However, the individual numbers of cross-rack and cross-region blocks may be different. For $f = 1$, the two numbers are decided by the numbers of surviving blocks (which may be diverse) in the helper regions, while for $f > 1$, the two numbers are determined by the number of blocks in the main region. We pose the analysis of general k_1 and m_1 as future work.

For example, in Figure 5(b), the multi-block repair of D_1, D_4, D_5, D_7, D_8, and C_3 works as follows. We first execute two column repairs for D_1 and D_4. We then select the first row and execute a row repair to decode D_5 and D_7. Specifically, we select the second region and the first rack therein as the main region and the main rack, respectively. We transfer D_1, D_3, R_1, and R_3 to the main rack to decode D_5 and D_7. We re-distribute D_7 to another rack. Finally, we execute two column repairs to decode C_3 and D_8. The multi-block repair transfers one cross-rack block and four cross-region blocks.



(a) Rack repair

(b) Region repair

Figure 6. Rack and region repairs in LPC$(4,2,2,1,3)$. The rack repair (figure (a)) transfers four cross-rack and four cross-region blocks. The region repair (figure (b)) transfers two cross-rack and eight cross-region blocks.

Rack repair. A failed rack implies a column of failed blocks (including k_2 data blocks and m_2 column parity blocks). Using the progressive multi-block repair, we first perform k_2 row repairs for the k_2 data blocks (which can be executed in parallel), so that we can later perform a column repair to decode the m_2 column parity blocks locally within the failed rack. Note that there is only one failed block in each row repair, so we apply partial decoding. From Equations (1) and (2), the numbers of cross-rack and cross-region blocks are

$$\#(\text{cross-rack blocks}) = k_1k_2 - \frac{k_1k_2}{m_1}, \tag{3}$$

$$\#(\text{cross-region blocks}) = \frac{k_1k_2}{m_1}. \tag{4}$$

For example, in Figure 6(a), in order to repair the blocks of the first rack, we perform two row repairs to decode D_1 and D_2. Then, we perform a column repair to decode C_1 locally in the failed rack. The rack repair transfers four cross-region blocks and four cross-rack blocks.

Region repair. A failed region implies m_1 failed columns (i.e., k_2m_1 data blocks and m_1m_2 column parity blocks). Using the progressive multi-block repair, we first perform k_2 row repairs for the k_2m_1 data blocks, and later perform m_1 column repairs to decode the m_1m_2 column parity blocks. We exclude partial decoding as there are m_1 failed blocks in each row repair. From Equations (1) and (2), the numbers of cross-rack and cross-region blocks are

$$\#(\text{cross-rack blocks}) = m_1k_2 - k_2, \tag{5}$$

$$\#(\text{cross-region blocks}) = k_1k_2. \tag{6}$$

For example, in Figure 6(b), to repair all blocks in the first region, we perform two row repairs to decode D_1 to D_4. Then, we perform two column repairs to decode C_1 and C_2 locally within each rack. The region repair transfers eight cross-region blocks and two cross-rack blocks.

Optimality. We now show that all types of block repairs minimize the sum of cross-rack and cross-region transfers.

First, each single-block repair and the repair of up to m_2 blocks within the same column are optimal as they can be locally done within a rack without any cross-rack and cross-region transfer.

Next, we show the optimality of multi-block repair. In each round of the while loop in Algorithm 1, we always execute the most column repairs but only one row repair with the maximum number of repairable failed blocks. The selection of a row in a round enables more column repairs in the next round. Overall, we execute the maximum number of column repairs and the minimum number of row repairs in a multi-block repair.

If we do not select the row with the maximum number of repairable failed blocks in a round, then we need at least two row repairs to maintain the same number of subsequent column repairs. From Equations (1) and (2), the sum of cross-rack and cross-region transfers increases (the number of row repairs also increases). Thus, we can deduce that all row repairs minimize the sum of cross-rack and cross-region transfers. To elaborate, we revisit the example in Figure 5(b). After executing two column repairs for D_1 and D_4, we select the first row and perform a row repair to decode D_5 and D_7. This single row repair transfers one cross-rack block and four cross-region blocks. If we do not select the first row, then we need to execute two row repairs to decode C_3 and D_8, so that we can perform two column repairs for D_5 and D_7. The two row repairs transfer four cross-rack blocks and four cross-region blocks (i.e., more transferred blocks).

Finally, both rack repair and region repair are in essence multi-block repairs. The optimality of multi-block repairs can be applied to both rack and region repairs.

V. Evaluation

We compare via numerical analysis and testbed experiments our LPC(k_1,m_1,k_2,m_2,r) with three baselines: (i) hierarchy-aware RS(k,m), which places every m blocks of a stripe in one region and applies partial decoding [13], [31], (ii) hierarchy-aware LRC(k,l,g), which places every $g+1$ blocks of each local group in one region and applies partial decoding [12], [37], (iii) Multi-Level Erasure Codes (denoted by MLEC(k_1,m_1,k_2,m_2)) [34], which also builds upon Product Codes but places a column of blocks in one region. For fair comparisons, we also apply partial decoding to MLEC.

We summarize the overall results as follows. From numerical analysis, LPCs completely eliminate cross-rack and cross-region transfers for single-block repair and reduce the sum of cross-rack and cross-region transfers of RS Codes, LRCs, and MLEC by up to 93.4%, 92.9%, and 87.4%, respectively, for multi-block repair (Section V-A). From testbed experiments, LPCs reduce the single- and multi-block repair times of RS Codes, LRCs, and MLEC by up to 90.6%, 80.8%, and 62.8%; and 70.0%, 73.2%, and 57.5%, respectively (Section V-B).

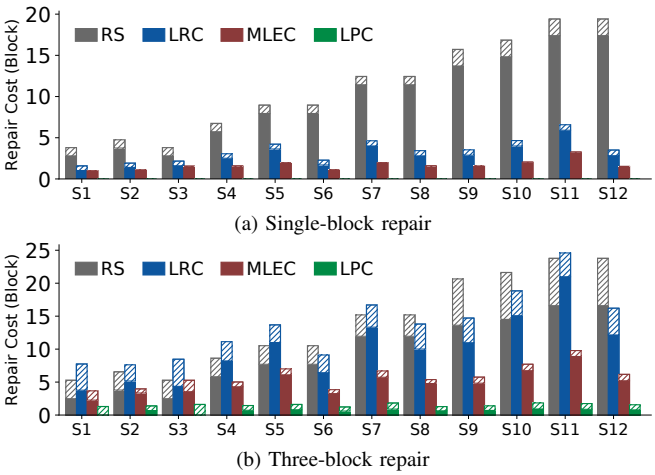
A. Numerical Analysis

We measure the cross-rack transfers and cross-region transfers individually, as well as their sum, for both single-block and multi-block repairs.

Settings. We evaluate LPC(k_1,m_1,k_2,m_2,r) under $2 \le k_1 \le 12$, $2 \le k_2 \le 6$, and $1 \le m_1,m_2 \le 2$. We then configure the

Table I
PARAMETER SETTINGS FOR FOUR ERASURE CODES.

	LPC	RS	LRC	MLEC
S1	(3,1,2,1,4)	(6,6)	(6,3,2)	(3,1,2,1)
S2	(4,2,2,1,3)	(8,8)	(8,4,4)	(4,2,2,1)
S3	(2,1,3,1,3)	(6,6)	(6,2,3)	(2,1,3,1)
S4	(4,2,3,1,3)	(12,12)	(12,4,6)	(4,2,3,1)
S5	(4,2,4,1,3)	(16,16)	(16,4,8)	(4,2,4,1)
S6	(8,2,2,1,5)	(16,16)	(16,8,4)	(8,2,2,1)
S7	(6,2,4,1,4)	(24,16)	(24,6,8)	(6,2,4,1)
S8	(8,2,3,1,5)	(24,16)	(24,8,6)	(8,2,3,1)
S9	(10,2,3,1,6)	(30,15)	(30,10,6)	(10,2,3,1)
S10	(8,2,4,1,5)	(32,16)	(32,8,8)	(8,2,4,1)
S11	(6,2,6,1,4)	(36,18)	(36,6,12)	(6,2,6,1)
S12	(12,2,3,1,7)	(36,18)	(36,12,6)	(12,2,3,1)



(a) Single-block repair

(b) Three-block repair

Figure 7. Repair traffic for single- and multi-block repairs, in terms of numbers of cross-rack blocks (solid bars) and cross-region blocks (shaded bars).

parameters of RS Codes, LRCs, and MLEC to keep the same number of data blocks in a stripe (by setting $k_1 \times k_2 = k$), while maintaining similar storage overhead and fault tolerance. We consider 12 parameter settings listed in Table I. For single-block repairs, we average the amounts of transfers over all blocks in a stripe. For multi-block repairs, we choose 10 random patterns of two or three failed blocks in a stripe and apply the same failure pattern to each erasure code, and derive the average amounts of transfers. We consider a topology with sufficient regions and nodes, and each region has two racks.

Results for single-block repairs. Figure 7(a) shows the cross-rack and cross-region transfers for single-block repairs under the 12 parameter settings. In all settings, LPCs require no cross-rack or cross-region transfer. RS Codes need k blocks to repair a failed block, and hence have the highest sum of cross-rack and cross-region transfers. LRCs have $\frac{k}{l} \leq g$ under all settings, meaning that LRCs always place a local group entirely in one region. Thus, the repair of a data block or local parity block incurs no cross-region transfer, but the repair of a global parity block still incurs cross-region transfers. MLEC places a column of blocks in one region, so it has no cross-region transfer, but still incurs cross-rack transfers.

Results for multi-block repairs. Figure 7(b) shows the repair traffic for repairing three failed blocks (similar results are observed for repairing two failed blocks). LPCs reduce the sum of cross-rack and cross-region transfers of RS Codes, LRCs, and MLEC by 69.6%-93.4%, 81.1%-92.9%, and 64.9%-87.4%, respectively. LPCs significantly reduce the cross-rack transfers of RS Codes, LRCs, and MLEC, and even have no cross-rack transfer when $m_1 = 1$; in this case, LPCs store one column/rack in one region, so a multi-block repair has no cross-rack transfer (Equation (1)). LPCs also significantly reduce the cross-region transfers.

B. Testbed Experiments

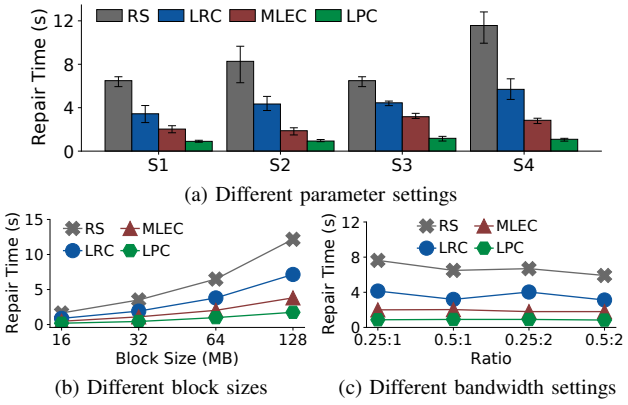
Implementation. We built a distributed storage prototype, in which we implemented LPCs, RS Codes, LRCs and MLEC based on Jerasure [23] and block repairs. Our prototype comprises a coordinator, multiple clients, and multiple regions. Each region contains a proxy and multiple racks, where each rack contains multiple nodes. The coordinator maintains the metadata, generates data placements, and coordinates the proxies in different regions to perform block repairs. Our prototype is implemented in C++ with around 8000 SLoC.

Setup. We deploy our prototype on a local cluster with 15 physical machines, each of which runs CentOS 7.9.2009 with 2 dodeca-core 2.20 GHz Intel(R) E5-2650 v4 CPUs, 64 GB RAM, a Seagate ST1000NM0023 7200 RPM 1 TB SATA hard disk, and 10 Gbps bandwidth. We deploy the coordinator and clients in one machine, assign two machines as network cores to simulate cross-rack and cross-region transfers, respectively. We use each of the remaining machines to emulate a region. In each region, we deploy two racks with 10 nodes each (i.e., a total of 240 nodes). We use the WonderShaper tool [3] to control the cross-rack and cross-region bandwidth. Such cluster settings have been explored in prior studies [12], [20], [40].

Methodology. We consider the four parameter settings, S1 to S4, from Table I, where all erasure codes share similar storage redundancy with up to 11.1% differences. By default, we set the block size as 64 MB, the cross-rack bandwidth as 1 Gbps, the cross-region bandwidth as 0.5 Gbps, and eight stripes in total (i.e., the total storage size is around 12 GB).

In RS Codes, LRCs, and MLEC, we distribute the blocks randomly to the two racks in a region to preserve single-region fault tolerance (Section III). For a single-block repair, we repair the failed block in each stripe one by one and record the average time. For multi-block repair, we randomly trigger 10 patterns of two or three failed blocks in a stripe and apply the same failure pattern to each erasure code; we again obtain the average repair time. We further average the results of each experiment over five runs.

Single-block repair performance. Figure 8(a) shows the single-block repair performance under different settings (the error bars show the maximum and minimum results across five runs). The four erasure codes have larger repair time under larger parameters. Overall, LPCs reduce the single-block repair time of RS Codes, LRCs, and MLEC by 81.6%-90.6%, 73.5%-80.8%, and 50.5%-62.8%, respectively, across the four parameter settings.

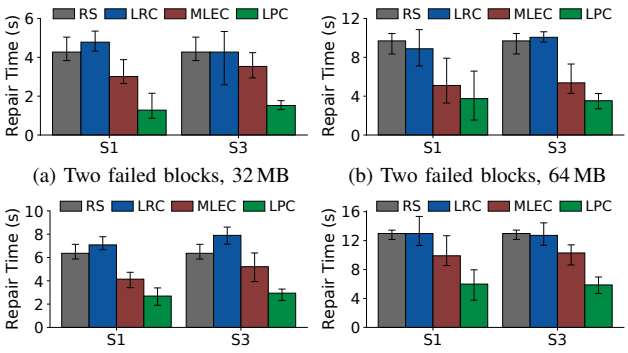


(a) Different parameter settings

(b) Different block sizes

(c) Different bandwidth settings

Figure 8. Performance for single-block repairs.



(a) Two failed blocks, 32 MB

(b) Two failed blocks, 64 MB

(c) Three failed blocks, 32 MB

(d) Three failed blocks, 64 MB

Figure 9. Performance for multi-block repairs.

We also evaluate the impact of block size, varied from 16 MB to 128 MB, on single-block repair performance. We focus on the setting S1. Figure 8(b) shows the results. LPCs reduce the single-block repair times of RS Codes, LRCs, and MLEC by 81.6%-88.5%, 73.8%-78.4%, and 51.2%-60.0%, respectively. LPCs have larger gains with a larger block size.

We further evaluate the impact of cross-region and cross-rack bandwidth in four combinations: 0.25 & 1 Gbps, 0.5 & 1 Gbps, 0.25 & 2 Gbps, and 0.5 & 2 Gbps. We focus on the setting S1 with a block size of 64 MB. Figure 8(c) shows the results. LPCs reduce the single-block repair times of RS Codes, LRCs, and MLEC by 81.6%-88.6%, 71.5%-79.0%, and 53.3%-56.5%, respectively. LPCs have larger gains with more scarce cross-rack (cross-region) bandwidth, as single-block repairs in LPCs are locally performed and are not affected by the cross-rack (cross-region) bandwidth, as opposed to other baselines.

Multi-block repair performance. We now evaluate multi-block repair performance with two and three failed blocks. We adopt the settings S1 and S3, and also consider two block sizes, 32 MB and 64 MB. Figure 9 shows the results (the error bars show the maximum and minimum results across five runs). LPCs achieve the smallest multi-block repair time. Note that LPCs have a higher performance gain with a smaller number of failed blocks, as the sum of cross-rack and cross-region transfers (i.e, $k_1 + f - 1$, where f is the number of failed blocks) increases with f, while RS Codes and LRCs need k blocks for multi-block repairs. For example, with two failed blocks with a block size of 32 MB, LPCs reduce the multi-block repair times of RS Codes, LRCs, and MLEC by 64.4%-70.0%, 64.4%-73.2%, and 56.9%-57.5%, respectively.

VI. Related Work

Erasure coding in geo-distributed storage systems. Erasure codes have been deployed in geo-distributed data centers for low-cost redundancy [6], [11], data security [28], region-level fault tolerance [2], [21]. These studies do not consider how to mitigate the significant cross-region transfers in block repairs. We design LPCs to effectively mitigate cross-region transfers in data repairs, and provide the optimality guarantees for LPCs.

Repair optimizations in hierarchical settings. Some studies focus on minimizing cross-rack transfers for single-block repairs in RS Codes [30], [31] and regenerating codes [13], [24]. Other studies consider LRCs and place each local group into a minimum number of racks to minimize cross-rack transfers for single-block repairs [20], [37], [40]. The studies [12], [39] address the high repair penalty in wide stripes, and apply LRCs [12] and XOR-Hitchhiker-RS Codes [39] to mitigate cross-rack transfers for single-block repairs. Our work complements existing studies by optimizing both single-block and multi-block repairs in geo-distributed storage systems.

Product Codes. Explicit constructions of Product Codes [10], [19] have been studied for high fault tolerance against disk failures in disk arrays. The idea has been extended for distributed storage. For example, CORE [8] provides cross-object redundancy with XOR-based column parities for high repair efficiency, but lacks support for hierarchical geo-distributed settings. R-STAIR codes [18] address node-level and rack-level fault tolerance, but focus only on single-block repairs. HACFS [38] transitions between fast and compact Product Codes to balance storage overhead and access performance. MLEC [34] builds on Product Codes to realize inner-rack single-block repairs for data centers. LPCs enhance MLEC in several aspects. First, LPCs consider a geo-distributed setting and optimize cross-region transfers, where MLEC shows suboptimal performance from our evaluation. Second, LPCs address multi-block repairs but not in MLEC. Third, LPCs are studied with both theoretical analysis and prototype evaluation, MLEC is studied with simulations only.

VII. Conclusion

We study both single-block and multi-block repairs in geo-distributed storage systems. We propose Leveled Product Codes (LPCs), which optimize block repair operations by localizing single-block repairs within racks and performing multi-block repairs with the provably minimum sum of cross-rack and cross-region transfers, while tolerating node, rack, and region failures. Our numerical analysis and testbed experiments validate the efficiency of LPCs in both single-block and multi-block repairs.

Acknowledgements. This work is supported by NSFC (62202440), Research Grants Council of Hong Kong (AoE/P-404/18). The corresponding author is Patrick P. C. Lee.

REFERENCES

[1] Backblaze. Erasure coding used by Backblaze. https://www.backblaze.com/blog/reed-solomon/.

[2] CubeFS - Erasure Code Subsystem. https://cubefs.readthedocs.io/en/latest/design/blobstore.html.

[3] The Wonder Shaper 1.4. https://github.com/magnific0/wondershaper.

[4] VastData. https://vastdata.com/providing-resilience-efficiently-part-ii/.

[5] F. Ahmad, S. T. Chakradhar, A. Raghunathan, and T. Vijaykumar. ShuffleWatcher: Shuffle-aware scheduling in multi-tenant Mapreduce clusters. In *Proc. of USENIX ATC*, 2014.

[6] Y. L. Chen, S. Mu, J. Li, C. Huang, J. Li, A. Ogus, and D. Phillips. Giza: Erasure coding objects across global data centers. In *Proc. of USENIX ATC*, 2017.

[7] A. G. Dimakis, P. B. Godfrey, Y. Wu, M. J. Wainwright, and K. Ramchandran. Network coding for distributed storage systems. *IEEE Trans. on Information Theory*, 56(9):4539–4551, 2010.

[8] K. S. Esmaili, L. Pamies-Juarez, and A. Datta. CORE: Cross-object redundancy for efficient data repair in storage systems. In *Proc. of IEEE BigData*, 2013.

[9] D. Ford, F. Labelle, F. Popovici, M. Stokely, V.-A. Truong, L. Barroso, C. Grimes, and S. Quinlan. Availability in globally distributed storage systems. In *Proc. of USENIX OSDI*, 2010.

[10] J. Hafner. HoVer erasure codes for disk arrays. In *Proc. of IEEE/IFIP DSN*, 2006.

[11] R. Halalai, P. Felber, A.-M. Kermarrec, and F. Taïani. Agar: A caching system for erasure-coded data. In *Proc. of IEEE ICDCS*, 2017.

[12] Y. Hu, L. Cheng, Q. Yao, P. P. C. Lee, W. Wang, and W. Chen. Exploiting combined locality for wide-stripe erasure coding in distributed storage. In *Proc. of USENIX FAST*, 2021.

[13] Y. Hu, X. Li, M. Zhang, P. P. C. Lee, X. Zhang, P. Zhou, and D. Feng. Optimal repair layering for erasure-coded data centers: From theory to practice. *ACM Trans. on Storage*, 13(4):33, 2017.

[14] C. Huang, M. Chen, and J. Li. Pyramid Codes: Flexible schemes to trade space for access efficiency in reliable data storage systems. *ACM Trans. on Storage (TOS)*, 9(1):1–28, 2013.

[15] C. Huang, H. Simitci, Y. Xu, A. Ogus, B. Calder, P. Gopalan, J. Li, and S. Yekhanin. Erasure coding in Windows Azure Storage. In *Proc. of USENIX ATC*, 2012.

[16] S. Kadekodi, S. Silas, D. Clausen, and A. Merchant. Practical design considerations for wide Locally Recoverable Codes (LRCs). In *Proc. of USENIX FAST*, 2023.

[17] O. Kolosov, G. Yadgar, M. Liram, I. Tamo, and A. Barg. On fault tolerance, locality, and optimality in Locally Repairable Codes. In *Proc. of USENIX ATC*, 2018.

[18] M. Li and P. P. C. Lee. Relieving both storage and recovery burdens in big data clusters with r-stair codes. *IEEE Internet Computing*, 22(4):15–26, 2018.

[19] M. Li, J. Shu, and W. Zheng. GRID codes: Strip-based erasure codes with high fault tolerance for storage systems. *ACM Trans. on Storage*, 4(4):1–22, 2009.

[20] S. Ma, S. Wu, C. Li, and Y. Xu. Repair-optimal data placement for locally repairable codes with optimal minimum hamming distance. In *Prof. of ACM ICPP*, 2022.

[21] S. Muralidhar, W. Lloyd, S. Roy, C. Hill, E. Lin, W. Liu, S. Pan, S. Shankar, V. Sivakumar, L. Tang, et al. f4: Facebook's warm BLOB storage system. In *Proc. of USENIX OSDI*, 2014.

[22] J. Plank and C. Huang. Tutorial: Erasure coding for storage applications. In *Slides presented at USENIX FAST*, 2013.

[23] J. S. Plank, J. Luo, C. D. Schuman, L. Xu, Z. Wilcox-O'Hearn, et al. A performance evaluation and examination of open-source erasure coding libraries for storage. In *Proc. of USENIX FAST*, 2009.

[24] N. Prakash, V. Abdrashitov, and M. Médard. The storage versus repair-bandwidth trade-off for clustered storage systems. *IEEE Trans.on Information Theory*, 64(8):5783–5805, Aug 2018.

[25] K. Rashmi, N. B. Shah, D. Gu, H. Kuang, D. Borthakur, and K. Ramchandran. A solution to the network challenges of data recovery in erasure-coded distributed storage systems: A study on the Facebook warehouse cluster. In *Proc. of USENIX HotStorage*, 2013.

[26] K. V. Rashmi, N. B. Shah, and K. Ramchandran. A piggybacking design framework for read-and download-efficient distributed storage codes. *IEEE Trans. on Information Theory*, 63(9):5802–5820, 2017.

[27] I. Reed and G. Solomon. Polynomial codes over certain finite fields. *Journal of the Society for Industrial and Applied Mathematics*, 8(2):300–304, 1960.

[28] J. K. Resch and J. S. Plank. AONT-RS: Blending security and performance in dispersed storage systems. In *Proc. of USENIX FAST*, 2011.

[29] M. Sathiamoorthy, M. Asteris, D. Papailiopoulos, A. G. Dimakis, R. Vadali, S. Chen, and D. Borthakur. XORing Elephants: Novel erasure codes for big data. In *Proc. of VLDB Endowment*, pages 325–336, 2013.

[30] Z. Shen, J. S. amd Zhijie Huang, and Y. Fu. ClusterSR: Cluster-aware scattered repair in erasure-coded storage. In *Proc. of IEEE IPDPS*, 2020.

[31] Z. Shen, J. Shu, and P. P. C. Lee. Reconsidering single failure recovery in clustered file systems. In *Proc. of IEEE/IFIP DSN*, 2016.

[32] K. Tang, K. Cheng, H. H. W. Chan, X. Li, P. P. C. Lee, Y. Hu, J. Li, and T.-Y. Wu. Balancing repair bandwidth and sub-packetization in erasure-coded storage via elastic transformation. In *Proc. of IEEE INFOCOM*, 2023.

[33] A. Vulimiri, C. Curino, P. B. Godfrey, T. Jungblut, J. Padhye, and G. Varghese. Global analytics in the face of bandwidth and regulatory constraints. In *Proc. of USENIX NSDI*, 2015.

[34] M. Wang, J. Mao, R. Rana, J. Bent, S. Olmez, A. George, G. W. Ransom, J. Li, and H. S. Gunawi. Design considerations and analysis of multi-level erasure coding in large-scale data centers. In *Proc. of ACM SC*, 2023.

[35] H. Weatherspoon and J. D. Kubiatowicz. Erasure coding vs. replication: A quantitative comparison. In *Proc. of IPTPS*, 2002.

[36] S. Wu, Q. Du, P. P. C. Lee, Y. Li, and Y. Xu. Optimal data placement for stripe merging in Locally Repairable Codes. In *Proc. of IEEE INFOCOM*, 2022.

[37] S. Wu, Z. Shen, and P. P. C. Lee. On the optimal repair-scaling trade-off in Locally Repairable Codes. In *Proc. of IEEE INFOCOM*, 2020.

[38] M. Xia, M. Saxena, M. Blaum, and D. A. Pease. A tale of two erasure codes in HDFS. In *Proc. of USENIX FAST*, 2015.

[39] G. Yang, H. Xue, Y. Gu, C. Wu, J. Li, M. Guo, S. Li, X. Xie, Y. Dong, and Y. Zhao. XHR-Code: An efficient wide stripe erasure code to reduce cross-rack overhead in cloud storage systems. In *Proc. of IEEE SRDS*, 2022.

[40] H. Zhao, S. Wu, H. Liu, Z. Tang, X. He, and Y. Xu. Toward optimal repair and load balance in locally repairable codes. In *Proc. of ACM ICPP*, 2023.