Harmonizing Repair and Maintenance in LRC-Coded Storage

Keyun Cheng[1], Si Wu[2], Xiaolu Li[3], and Patrick P. C. Lee[1]
[1]*The Chinese University of Hong Kong* [2]*University of Science and Technology of China*
[3]*Huazhong University of Science and Technology*

***Abstract*—Modern storage systems not only introduce data redundancy for fault tolerance, but also conduct regular maintenance operations on storage nodes for system robustness. Erasure coding provides storage-efficient redundancy and has been widely deployed in production, yet it also incurs substantial bandwidth and I/O overhead due to the repair of storage failures. In particular, maintenance operations make storage nodes temporarily unavailable and lead to data unavailability, thereby incurring repair overhead for erasure-coded storage. In this paper, we study Locally Repairable Codes (LRCs), a class of practical repair-efficient erasure codes, and show that there exists an inherent performance trade-off between the repair and maintenance operations of LRCs in data center settings, such that the repair performance in regular (i.e., no-maintenance) and maintenance modes cannot be simultaneously optimized. To this end, we design a configurable data placement scheme that operates along the trade-off subject to fault-tolerance constraints. We prototype our data placement scheme atop Hadoop HDFS and show how it balances the performance trade-off of repair and maintenance operations in real network environments.**

I. Introduction

Given the prevalence of failures [13], modern large-scale storage systems often adopt various reliability mechanisms to provide data availability and system robustness guarantees. There are two key reliability mechanisms: (i) *data redundancy* and (ii) *system maintenance*. Specifically, storage systems introduce data redundancy, via replication or erasure coding, to ensure that any stored data remains available and is recoverable from redundant data even in the presence of failures. In addition, system administrators regularly conduct maintenance on storage nodes for system stability and robustness [13], [15], [20], [29]; for example, storage nodes perform periodic kernel upgrades to keep the system up-to-date [27].

Replication is often used as a redundancy scheme by making multiple exact data copies for fault tolerance, yet it incurs significantly high storage overhead. *Erasure coding* is a storage-efficient redundancy scheme and has been widely adopted in production [1], [2], [9], [13], [24], [26]. It works by encoding a set of original uncoded *data blocks* into coded *parity blocks* (as redundant data), such that a subset of a sufficient number of available data and parity blocks can decode a block. Compared with replication, erasure coding achieves a higher degree of reliability under the same amount of redundancy [39]. On the other hand, erasure coding, albeit storage-efficient, suffers from a high repair penalty, as repairing any failed block by retrieving multiple available blocks for decoding, thereby incurring substantial network transfers and disk I/Os.

In particular, the repair overhead of erasure coding makes system maintenance more challenging. Maintenance of storage systems is often conducted in the *maintenance events* over a subset of storage nodes in a controlled manner. During the execution of maintenance events, software restarts or machine reboots of storage nodes are often necessary, thereby causing the data stored in such storage nodes inaccessible. To address data unavailability due to maintenance events for erasure-coded storage, one approach is to put restrictions on data placement, namely *maintenance-robust deployment* [20]. Specifically, storage nodes are partitioned into multiple groups called *maintenance zones* and maintenance events are only conducted on a per-zone basis. Maintenance-robust deployment carefully organizes erasure-coded blocks across multiple maintenance zones, such that any block stored in any affected maintenance zone can be reconstructed by retrieving the available blocks from other unaffected maintenance zones; in other words, any maintenance event will not make any block unavailable. However, repairing any failed block during maintenance events still incurs significant performance overhead, and how to mitigate the performance overhead of maintenance events for erasure-coded storage remains an unexplored issue.

In this paper, we study the performance tensions between repair and maintenance operations for erasure-coded storage. We focus on Locally Repairable Codes (LRCs) [19], [21], [33], [36], which are a popular family of repair-efficient erasure codes that mitigate the repair overhead over traditional erasure codes (e.g., Reed-Solomon codes [31]) at the expense of slightly higher redundancy and have been adopted by industry [19]–[21]. LRCs partition the data blocks into multiple small-size local groups, each of which is further encoded with a local parity block, such that a failed block can be repaired within a local group with much fewer available blocks. Despite the repair efficiency of LRCs, we show that there exists an inherent performance trade-off between the repair and maintenance operations, such that the repair performance in regular (no-maintenance) and maintenance modes (measured by the cross-rack bandwidth in rack-based data centers) cannot be simultaneously optimized. We make the following contributions:

- We formulate a data placement problem for the repair and maintenance operations in rack-based data centers. We characterize the feasible data placements, and obtain the optimal repair and maintenance schemes by solving integer linear programming problems. We show the performance trade-off between the repair and maintenance operations.
- We design a configurable data placement scheme that operates along the trade-off between repair and maintenance operations subject to the fault tolerance constraints.

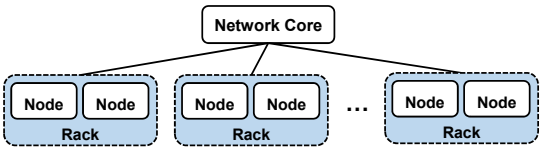


Figure 1: Example of a rack-based data center.

- We prototype different data placement schemes atop Hadoop 3.3.4 HDFS [2] and evaluate them using both numerical analysis and testbed evaluation. Our evaluation results demonstrate how the data placement schemes balance the performance trade-off between the repair and maintenance operations. For example, we can configure a repair-driven data placement scheme that reduces the degraded read time in regular mode by 68.5% compared with the optimal maintenance scheme, while we can configure a maintenance-driven data placement scheme that reduces the degraded read time in maintenance mode by 31.7% compared with the optimal repair scheme.

The source code of our prototype is available at **https://github.com/adslabcuhk/openec-lrctradeoff**.

II. Background

A. Rack-based Data Centers

We consider a large-scale storage system that is deployed as a rack-based data center [11], as shown in Figure 1. The storage system comprises multiple *nodes* that are partitioned into multiple *racks*, in which the nodes within a rack are connected by a top-of-rack switch, while multiple racks are connected by an aggregation layer of switches called the *network core*. We assume that the cross-rack network transfer is the performance bottleneck, as the cross-rack network bandwidth is much more scarce than the inner-rack bandwidth [8], [11], [38]. Such a hierarchical architecture also appears in geo-distributed erasure-coded storage [10], where cross-region bandwidth is much more limited than the inner-region bandwidth. In addition, the storage system organizes data as large fixed-sized blocks (e.g., 128 MiB in Hadoop HDFS [2] and 256 MiB in Facebook [29]); setting a large block size can effectively mitigate the I/O access overhead in networked environments.

B. Locally Repairable Codes (LRCs)

In this work, we focus on LRCs as the erasure coding construction due to their repair efficiency.

Basics of LRCs. We construct an LRC by three parameters (k,l,g), denoted by LRC(k,l,g). Specifically, LRC(k,l,g) encodes k data blocks (denoted by $D_1,D_2,\cdots,D_k$) into l *local parity blocks* (denoted by $P_1,P_2,\cdots,P_l$) and g *global parity blocks* (denoted by $Q_1,Q_2,\cdots,Q_g$) based on some encoding functions. Each set of k data blocks, l local parity blocks, and g global parity blocks collectively form a *stripe* and are distributed across $k+l+g$ nodes for fault tolerance. Large-scale storage systems typically store multiple stripes that are independently encoded. In this work, our analysis focuses on a single LRC stripe.

There exist various LRC constructions in the literature [19], [20], [33], [36]. In this work, we consider the LRC construction

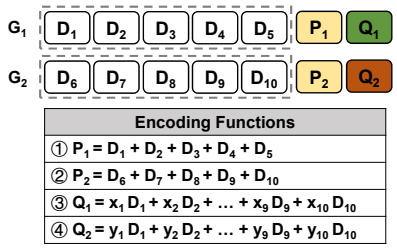


Figure 2: Example of an LRC$(10,2,2)$ stripe.

based on Azure's Local Reconstruction Codes (Azure-LRC) [19], since it achieves the highest level of fault tolerance under the same storage redundancy in rack-based data centers [16]. Specifically, it divides k data blocks into l equal-sized local groups, assuming that k is divisible by l. Each local group computes a local parity block from the bitwise-XOR-sum of its $b=\frac{k}{l}$ data blocks. All k data blocks are encoded into the g global parity blocks based on Reed-Solomon codes [31], such that any k out of the $k+g$ data blocks and global parity blocks can reconstruct all k data blocks. Figure 2 shows an example of an LRC$(10,2,2)$ stripe based on Azure-LRC, in which '+' denotes the bitwise-XOR-sum, and $x_1,x_2,\cdots,x_{10},y_1,y_2,\cdots,y_{10}$ are the encoding coefficients based on Reed-Solomon codes. The figure also lists the encoding functions (i.e., ① to ④) for computing the local and global parity blocks.

Fault tolerance. The $k+l+g$ blocks of each LRC stripe are distributed across multiple nodes and racks for fault tolerance. Specifically, we store the $k+l+g$ blocks of an LRC stripe in $k+l+g$ distinct nodes for multi-node fault tolerance. As rack failures are much rarer than node failures [24], we focus on single-rack fault tolerance [34], [40], [41]. We still distribute the blocks of an LRC stripe across racks, but we also allow multiple blocks of an LRC stripe to be stored in a rack so as to limit cross-rack data transfers. Wu et. al [41] prove that to achieve single-rack fault tolerance, each rack can store at most $g+h$ blocks of an LRC stripe that span h local groups (where $1\le h\le l$). In this case, we can decode at most $g+h$ failed data blocks with $g+h$ available parity blocks from the available racks.

Repair. There are two types of repair operations: (i) *degraded reads*, in which a read request is issued to an unavailable block, and (ii) *full-node recovery*, in which all lost blocks of a failed node are recovered. As transient failures account for the majority of all failure events (e.g., more than 90% of failure events last less than 15 minutes [13]), we focus on degraded reads to data blocks as the major repair operation in this work.

We define the *average degraded-read cost (ADC)* as the average amount of cross-rack transfers (in units of blocks) to repair an unavailable data block over all k data blocks (note that local and global parity blocks are excluded). Here, we assume that maintenance is not yet conducted, and the system operates in *regular mode* (i.e., without maintenance). For a *flat* data placement where each rack stores only one block of an LRC stripe, the ADC is b, as each data block is repaired by retrieving $b-1$ other available data blocks and the local parity block in the local group for decoding.

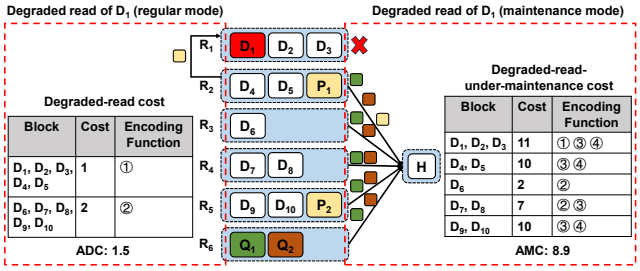


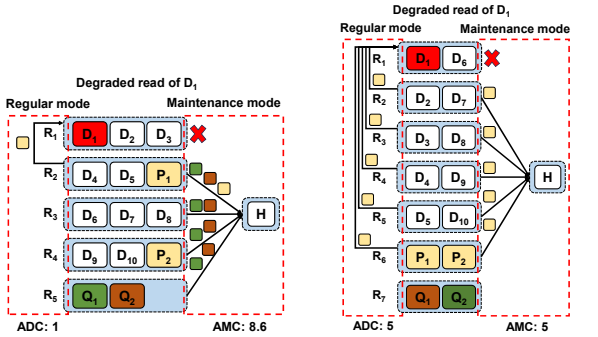
Figure 3: Example of a random data placement for LRC$(10,2,2)$.

For rack-based data centers, since multiple blocks of a local group can be stored in a rack, the ADC also decreases. Figure 3 depicts the calculation of the ADC of a random data placement for LRC$(10,2,2)$ under single-rack fault tolerance, in which the LRC stripe is distributed across 14 nodes across six racks $R_1, R_2, \cdots, R_6$. For example, D_1 can be decoded by retrieving D_2 and D_3 from the same rack R_1, and one partially decoded block $D_4 + D_5 + P_1$ from rack R_2 based on the encoding function ① in Figure 1. Thus, the amount of cross-rack transfers is one block. We can calculate the ADC by averaging the amount of cross-rack transfers over $D_1, D_2, \cdots D_{10}$, and the ADC in this case is 1.5.

Maintenance. Recall from §I that all data blocks within a maintenance zone cannot be directly accessed during maintenance. Given that maintenance events are scheduled a priori, it is feasible to design a data placement scheme that satisfies *maintenance-robust deployment* [20], such that no maintenance events cause any data block unrecoverable (i.e., any unavailable data block can still be decoded by the available blocks within the same stripe from other maintenance zones). In this work, we assume that a maintenance zone comprises one single rack, where maintenance-robust deployment implies single-rack fault tolerance, and at most one maintenance zone is under maintenance at any time [20].

We now describe the degraded reads in *maintenance mode* for a single data block; as opposed to the regular mode (see above), all blocks stored in the rack under maintenance are unavailable. Here, we conduct the repair in a *helper node*, which retrieves the available blocks from other unaffected racks for decoding. To simplify our analysis, we assume that the helper node resides in a rack that does not store any blocks of the corresponding stripe. We define the *average degraded-read-under-maintenance cost (AMC)* as the average amount of cross-rack transfers (in units of blocks) to repair an unavailable data block over all k data blocks under maintenance. For a flat data placement, the AMC is still b (i.e., the same as the ADC), as any unavailable rack is equivalent to a single block failure.

We again consider the data placement in Figure 3 and show how the AMC is calculated, where the helper node (denoted by H) is responsible for conducting the degraded reads in maintenance mode. For example, consider the degraded reads to D_1 when rack R_1 is under maintenance (i.e., D_1, D_2 and D_3 are unavailable). The repair of D_1 is done by the associated three encoding functions ①, ③ and ④ (where D_1, D_2 and D_3 are treated as the unknown variables in decoding). First, it retrieves one partially decoded block (i.e., $D_4 + D_5 + P_1$) from rack R_2 based on ①. Second, it retrieves five blocks from racks $R_2, R_3, \cdots, R_6$ based on ③ (i.e., $x_4D_4 + x_5D_5, x_6D_6, x_7D_7 + x_8D_8, x_9D_9 + x_{10}D_{10}$, and Q_1, respectively). Third, it retrieves five blocks from $R_2, R_3, \cdots, R_6$ based on ④ (i.e., $y_4D_4 + y_5D_5, y_6D_6, y_7D_7 + y_8D_8, y_9D_9 + y_{10}D_{10}$, and Q_2, respectively). Thus, the amount of cross-rack transfers for repairing D_1 is 11 blocks. The average AMC over $D_1, D_2, \cdots D_{10}$ for this data placement is 8.9.



(a) Combined locality

(b) Maintenance-robust-efficient deployment

Figure 4: Example of state-of-the-art data placements for LRC$(10,2,2)$. Combined locality shows a high AMC, while maintenance-robust-efficient deployment shows a high ADC.

C. Motivation

Extensive studies focus on minimizing the repair overhead in regular mode for LRCs [16], [21], [36], [41]. However, when maintenance is also considered, how to address the repair performance tensions in both regular and maintenance modes is unexplored. Here, we review two state-of-the-art data placement schemes for LRC-coded storage: the repair-driven scheme designed for optimizing the repair performance in regular mode [16], and the maintenance-driven scheme designed for optimizing the repair performance in maintenance mode [20]. We show via examples that they cannot minimize the degraded read overhead in both regular and maintenance modes.

Repair-driven data placements. *Combined locality* [16] is an optimal data placement that minimizes the ADC in regular mode subject to single-rack fault tolerance. It aggregates each local group across a minimum number of racks, such that the repair of each data block retrieves the minimum number of blocks from the racks spanned by the corresponding local group. Specifically, it stores an LRC stripe across $l\lceil\frac{b+1}{g+1}\rceil+1$ racks, where each local group (comprising $b+1$ blocks) is stored across $\lceil\frac{b+1}{g+1}\rceil$ racks, and the g global parity blocks are stored in a distinct rack.

While combined locality minimizes the ADC in regular mode, it incurs a high AMC in maintenance mode. Figure 4(a) shows an example of combined locality for LRC$(10,2,2)$. For example, in regular mode, the repair of D_1 incurs a cross-rack transfer of one block from R_2 to R_1. However, in maintenance mode, the repair of D_1 transfers nine blocks, where the helper node H needs to retrieve three, two, two, and two blocks from

R_2, R_3, R_4, and R_5, respectively. Overall, the ADC and AMC of combined locality are 1 and 8.6, respectively.

Maintenance-driven data placements. *Maintenance-robust-efficient deployment* [20] aims to mitigate the AMC in LRC-coded storage, while preserving the single-rack fault tolerance under maintenance-robust deployment. It spreads each local group across a maximum number of racks, such that each rack stores at most one block from any local group. Thus, the repair of each data block retrieves the blocks based on the corresponding encoding function in the local group. Maintenance-robust-efficient deployment stores an LRC stripe across $b+2$ racks, where each of the first b racks stores l data blocks from l local groups. It then stores the l local parity blocks and the g global parity blocks in the remaining two racks, respectively.

We argue that maintenance-robust-efficient deployment shows a high ADC. Figure 4(b) shows an example. The repair of D_1 in maintenance mode is five blocks (i.e., 44.4% less than that of combined locality). However, the repair of D_1 in regular mode is five blocks (i.e., 5× that of combined locality). Overall, the ADC and AMC of maintenance-robust-efficient deployment are both 5.

III. Problem and Analysis

We now characterize the feasible data placements for LRCs that are maintenance-robust (i.e., single-rack fault tolerance) and show the calculations of the ADC and AMC for a data placement. We show how we can obtain the optimal repair and maintenance schemes that minimize the ADC and AMC, respectively, by solving integer linear programming (ILP) problems. We further provide insights into the performance trade-off between the repair and maintenance operations.

A. Modeling a Data Placement

We model a data placement for an LRC(k,l,g) stripe with three sets of non-negative integer parameters, namely $d_{i,j}$, $p_{i,j}$, and q_i where $1 \le i \le k+l+g$ and $1 \le j \le l$. Let $d_{i,j}$ and $p_{i,j}$ be the numbers of data blocks and local parity blocks from the j-th local group G_j stored in rack R_i, respectively, and let q_i be the number of global parity blocks stored in rack R_i. Let $I(x)$ be the indicator function of an input integer x, where $I(x)=1$ when x is positive or $I(x)=0$ otherwise. We derive the following constraints on a feasible data placement:

$$\sum_{i=1}^{k+l+g} d_{i,j} = b \quad \text{for each } 1 \le j \le l, \tag{1}$$

$$\sum_{i=1}^{k+l+g} p_{i,j} = 1 \quad \text{for each } 1 \le j \le l, \tag{2}$$

$$\sum_{i=1}^{k+l+g} q_i = g\ , \tag{3}$$

$$q_i + \sum_{j=1}^{l}(d_{i,j}+p_{i,j}) \le g + \sum_{j=1}^{l} I(d_{i,j}+p_{i,j}) \quad \text{for each } 1 \le i \le k+l+g. \tag{4}$$

We elaborate on the constraints as follows. Equations (1), (2), and (3) limit the total numbers of data blocks, local parity blocks, and global parity blocks spanned across all racks, respectively. Equation (4) ensures that each rack stores at most $g+h$ blocks that span h local groups (where $1 \le h \le l$) to preserve single-rack fault tolerance [41] (§II-B). In Equation (4), the left-hand side calculates the total number of blocks stored in R_i, while the right-hand side calculates the maximum number of blocks that can be stored in R_i. Note that $I(d_{i,j}+p_{i,j})$ indicates whether any block from G_j is stored in R_i, and $\sum_{j=1}^{l} I(d_{i,j}+p_{i,j})$ calculates the number of local groups spanned in R_i. Figure 5 shows the modeling of the data placement shown in Figure 3, and the constraints Equations (1)-(4) are all satisfied.

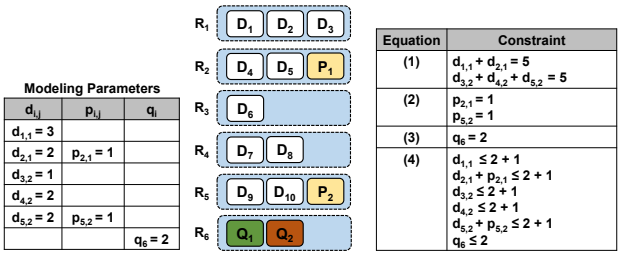


Figure 5: Example of the data placement of LRC$(10,2,2)$ in Figure 3. The data placement satisfies all constraints Equations (1)-(4).

B. Calculating the ADC

We show how the ADC of a data placement is calculated. We first calculate the amount of cross-rack transfers of a degraded read to a single block in regular mode. Suppose that we repair a data block from local group G_j stored in rack R_i. It retrieves blocks based on the encoding function of local parity block P_j for decoding. Specifically, it retrieves a total of $\delta_j - 1$ blocks, where δ_j denotes the number of racks spanned by G_j:

$$\delta_j = \sum_{i=1}^{k+l+g} I(d_{i,j}+p_{i,j}) \text{ for each } 1 \le j \le l, \tag{5}$$

where $I(d_{i,j}+p_{i,j})$ indicates whether any block of G_j is stored in R_i, and δ_j counts the number of such racks.

Let $r_{i,j}$ be the amount of cross-rack transfers (in units of blocks) to repair a data block from G_j stored in R_i in regular mode. We have $r_{i,j} = \delta_j - 1$. There are $d_{i,j}$ blocks from G_j stored in R_i, where the repair of each block retrieves $r_{i,j}$ blocks. Thus, we can calculate ADC as follows:

$$\begin{aligned} \text{ADC} &= \tfrac{1}{k}\sum_{i=1}^{k+l+g}\sum_{j=1}^{l} d_{i,j} r_{i,j} \\ &= \tfrac{1}{l}\sum_{j=1}^{l}(\delta_j - 1), \end{aligned} \tag{6}$$

where we average the amount of cross-rack transfers over the k data blocks from l local groups stored in $k+l+g$ racks. The deduction of Equation (6) is based on Equation (1).

For example, in Figure 5, the repair of D_1 in regular mode retrieves one block, which can be represented by $r_{1,1} = \delta_1 - 1$. The ADC can be calculated as $\frac{1}{2} \times ((2-1)+(3-1)) = 1.5$.

Optimality of ADC. We can obtain the optimal repair scheme that minimizes the ADC by solving an ILP problem. The optimization objective is to find an assignment of the modeling parameters (i.e., $d_{i,j}$, $p_{i,j}$, and q_i) to minimize the objective function (i.e., Equation (6)), subject to the constraints (i.e., Equations (1)-(4)). The above ILP problem can be solved with the branch-and-bound method [3].

We provide an intuition for optimizing the ADC. From Equation (6), the ADC is minimized when δ_j is minimized

for each local group G_j simultaneously. This is achievable when each local group spans the minimum number of racks. To preserve single-rack fault tolerance, each rack stores at most $g+1$ blocks from a local group (§II-B). Thus, each G_j (with $b+1$ blocks) spans $\delta_j = \lceil \frac{b+1}{g+1} \rceil$ racks. Repair-driven data placements satisfy the above property and hence minimize the ADC (§II-C).

C. Calculating the AMC

We show how to calculate the AMC of a data placement. We consider two types of repair operations, namely *local decoding* and *global decoding*, in maintenance mode. Suppose that rack R_i is under maintenance. Let $m_{i,j}$ be the amount of cross-rack transfer (in units of blocks) to repair a data block from G_j stored in R_i in maintenance mode.

Local decoding. Local decoding of a data block is done by retrieving the blocks within the same local group of the data block. It is feasible only when $d_{i,j} = 1$ and $p_{i,j} = 0$, meaning that the data block should be the only block of G_j stored in R_i. The reason is that the other data blocks of G_j and the local parity block P_j need to be available for decoding, and hence they cannot be stored in R_i. Similar to the repair in regular mode, we have $m_{i,j} = \delta_j - 1$.

For example, in Figure 3, when R_3 is under maintenance, the repair of D_6 in maintenance mode retrieves two blocks (i.e., $D_7 + D_8$ and $D_9 + D_{10} + P_2$), where local decoding is feasible (as $d_{3,2} = 1$ and $p_{3,2} = 0$). The amount of cross-rack transfers for D_6 is $m_{3,2} = \delta_2 - 1 = 2$.

Global decoding. Global decoding of a data block is done by retrieving the blocks spanning multiple local groups within the same stripe of the data block. It addresses two cases: (i) $d_{i,j} = 1$ and $p_{i,j} = 1$, and (ii) $d_{i,j} \geq 2$, meaning that multiple blocks from G_j are stored in R_i. Specifically, we form a system of equations by associating $\sum_{x=1}^{l} d_{i,x}$ encoding functions for decoding, where the $\sum_{x=1}^{l} d_{i,x}$ data blocks stored in R_i are treated as unknown variables in decoding. For each local group G_x (where $1 \leq x \leq l$), if some data blocks of G_x are stored in R_i (i.e., $d_{i,x} \geq 1$), we select $d_{i,x}$ encoding functions to form the equations. The selection of encoding functions depends on the value of $p_{i,x}$ (i.e., whether the local parity block P_x is stored in R_i), which affects the amount of cross-rack transfers for decoding. Let σ be the total number of racks spanned by all data blocks:

$$\sigma = \sum_{i=1}^{k+l+g} I(\sum_{j=1}^{l} d_{i,j}). \tag{7}$$

We elaborate on the selection of encoding functions as follows:

- When $p_{i,x} = 0$ (i.e., P_x is not stored in R_i and hence available), we select the encoding function of P_x, plus $d_{i,x} - 1$ encoding functions of $d_{i,x} - 1$ available global parity blocks. To solve an equation based on P_x, we retrieve $\delta_x - 1$ blocks for decoding. To solve an equation based on a global parity block, we retrieve σ blocks for decoding, including the global parity block and $\sigma - 1$ blocks from the unaffected racks spanned by the data blocks. In total, it retrieves $(d_{i,x} - 1)\sigma + \delta_x - 1$ blocks for decoding.
- When $p_{i,x} = 1$ (i.e., P_x is unavailable), we select $d_{i,x}$ encoding functions of $d_{i,x}$ available global parity blocks. Solving an equation based on a global parity block retrieves σ blocks for decoding. In total, it retrieves $d_{i,x}\sigma$ blocks for decoding.

We can calculate $m_{i,j} = \gamma_i$, where γ_i denotes the amount of cross-rack transfers to solve all equations (for R_i):

$$\gamma_i = \sum_{x=1}^{l} I(d_{i,x})(d_{i,x}\sigma + (1 - p_{i,x})(\delta_x - \sigma - 1)), \tag{8}$$

in which we sum up the number of blocks retrieved for each local group G_x where $d_{i,x} \geq 1$ (i.e., $I(d_{i,x}) = 1$).

For example, in Figure 3, when R_1 is under maintenance, the repair of D_1 in maintenance mode retrieves 11 blocks, which requires global decoding (as $d_{1,1} \geq 2$). We associate three encoding functions to form a system of equations, where D_1, D_2, and D_3 are treated as the unknown variables. As $d_{1,1} \geq 1$ and $p_{1,1} = 0$, we select the encoding function ① of local parity block P_1, and two encoding functions ③ and ④ of global parity blocks Q_1 and Q_2, respectively. For ①, we retrieve $\delta_1 - 1 = 1$ block. For each of ③ and ④, we retrieve $\sigma = 5$ blocks. The amount of cross-rack transfers for D_1 can be represented by $m_{1,1} = \gamma_1$. Note that the repair of both D_2 and D_3 also requires global decoding, and hence retrieves γ_1 blocks for decoding.

We can now calculate the AMC as follows by averaging the amount of cross-rack transfers over the k data blocks from l local groups stored in $k+l+g$ racks:

$$\text{AMC} \quad = \quad \frac{1}{k} \sum_{i=1}^{k+l+g} \sum_{j=1}^{l} d_{i,j} m_{i,j}, \text{ where}$$

$$m_{i,j} = \begin{cases} 0 & \text{for } d_{i,j} = 0, \\ \delta_j - 1 & \text{for } d_{i,j} = 1, p_{i,j} = 0, \\ \gamma_i & \text{for } d_{i,j} = 1, p_{i,j} = 1 \text{ or } d_{i,j} \geq 2. \end{cases} \tag{9}$$

Optimality of AMC. We can also obtain the optimal data placement that minimizes the AMC by solving an ILP problem, where the optimization objective is Equation (9), and the constraints are Equations (1)-(4). However, compared with ADC, optimizing AMC is computationally much more difficult. Intuitively, the calculation of AMC needs to consider both local decoding and global decoding; in particular, global decoding needs to solve multiple equations based on some local and global encoding functions, where the selection of encoding functions also depends on the values of modeling parameters (i.e., $d_{i,j}$ and $p_{i,j}$). Nevertheless, we provide an insight into finding the optimality of AMC. The key idea is that the amount of cross-rack transfers to repair a data block in maintenance mode can be minimized *only if* local decoding is feasible. We support the insight with the following proposition:

Proposition 1. *Suppose that $d_{i,j} \geq 1$ for some rack R_i and local group G_j (where $1 \leq i \leq k+l+g$ and $1 \leq j \leq l$). Then $m_{i,j}$ is minimized only when $d_{i,j} = 1$ and $p_{i,j} = 0$.*

Proof. First, we prove that when (i) $d_{i,j} = 1$ and $p_{i,j} = 1$, or (ii) $d_{i,j} \geq 2$, we have $m_{i,j} \geq \sigma$. It corresponds to the case that global decoding is required, and hence $m_{i,j} = \gamma_i$. Based on Equation (8), γ_i is minimized only when (i) $d_{i,j} = 1$ and $p_{i,j} = 1$ for G_j, and (ii) $d_{i,x} = 0$ for each G_x where $1 \leq x \leq l$

and $x \neq j$, meaning that R_i stores only one data block of G_j as well as local parity block P_j. In this case, we have $\gamma_i = \sigma$. Thus, we have $m_{i,j} \geq \sigma$.

Next, we discuss the opposite case when $d_{i,j} = 1$ and $p_{i,j} = 0$, where local decoding is feasible and hence $m_{i,j} = \delta_j - 1$. Now we show that $\delta_j - 1 \leq \sigma$ for any G_j (where $1 \leq j \leq l$), as the number of racks spanned by G_j (i.e., δ_j) must be no larger than the number of racks spanned by all data blocks (i.e., σ) plus one (even when local parity block P_j is not co-located with any data block of G_j). Hence, we complete the proof. □

Proposition 1 shows that to repair a data block in maintenance mode, global decoding retrieves more blocks than local decoding. It suggests that to minimize the amount of cross-rack transfers for decoding, the block should not be co-located with any other blocks from its local group in a rack, so that local decoding is feasible.

We argue that there exists a design dilemma in minimizing the AMC. Recall that local decoding of a block from G_j retrieves $\delta_j - 1$ blocks (Equation (9)). To minimize the cross-rack transfers for decoding in maintenance mode, we also need to minimize δ_j. To achieve this, the remaining blocks from G_j need to be aggregated in the minimum number of racks. However, the repair of any remaining block of G_j in maintenance mode requires global decoding, which significantly enlarges the AMC. On the other hand, we can spread the blocks of G_j across the maximum number of racks, such that local decoding is feasible to repair any block (e.g., maintenance-driven data placement). Meanwhile, as δ_j is also maximized, the cross-rack transfers for local decoding cannot be minimized. Nevertheless, we show via evaluation that maintenance-driven data placement can achieve a near-optimal AMC and effectively reduce the degraded read time in maintenance mode (§V).

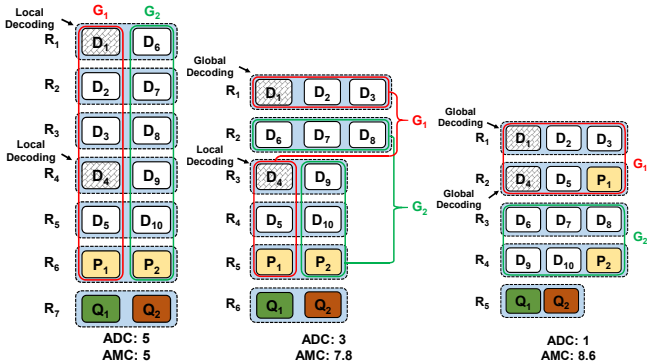
Discussion. From prior analysis, the optimization of ADC and AMC shows two inherently different directions. To minimize the ADC, each local group should span the minimum number of racks, but it enlarges the AMC, as most data blocks need global decoding. On the other hand, to minimize the AMC, each local group tends to span the maximum number of racks to enable local decoding, but it enlarges the ADC. Thus, we cannot minimize both ADC and AMC simultaneously.

IV. Configurable Data Placement Scheme

We design a configurable data placement scheme that operates along the trade-off between ADC and AMC subject to single-rack fault tolerance. By adjusting a single configuration parameter that controls the aggregation degree of data blocks, our configurable data placement scheme effectively balances the degraded read performance in both regular and maintenance modes. We first present our configurable data placement scheme with a guiding example (§IV-A), followed by the trade-off analysis based on our modeling (§IV-B).

A. Design Details

Our configurable data placement scheme generates a data placement for an LRC(k,l,g) stripe that provides single-rack fault tolerance. It uses a configuration parameter η (where $0 \leq \eta \leq \lceil \frac{b}{g+1} \rceil$) to control the aggregation degree of data blocks. Specifically, η controls the proportion of data blocks from the same local group that are aggregated within a limited number of racks, such that (i) with a small η, it generates a data placement with a low AMC and a high ADC, where a large proportion of data blocks from the same local group are spread across different racks; (ii) by increasing η, it trades AMC for ADC, where more data blocks from the same local group are aggregated in a limited number of racks. We present our configurable data placement scheme as follows:



(a) $\eta = 0$ (b) $\eta = 1$ (c) $\eta = 2$

Figure 6: Example of data placements generated by our data placement scheme for LRC$(10,2,2)$.

- *Step 1 (Co-locate data blocks of a local group):* For each local group G_j (where $1 \leq j \leq l$), we choose η new racks and put $g+1$ data blocks of G_j in each new rack.
- *Step 2 (Spread data blocks of a local group):* For each local group G_j, we spread the remaining $b - \eta(g+1)$ data blocks in $b - \eta(g+1)$ new racks. In each new rack, we co-locate l data blocks from l different local groups.
- *Step 3 (Put local parity blocks):* We choose one new rack and put l local parity blocks in the rack.
- *Step 4 (Put global parity blocks):* We choose one new rack and put g global parity blocks in the rack.

Note that there remains a corner case of Step 1. When $\eta = \lceil \frac{b}{g+1} \rceil$, if b is not divisible by $g+1$, one of the η new racks will store fewer than $g+1$ data blocks from G_j. In this case, we put the remaining b mod $(g+1)$ data blocks, together with the corresponding local parity block in the new rack. Step 3 does not need to be performed accordingly, as all local parity blocks are placed in Step 1.

Guiding example. For a better illustration of the performance trade-off, we walk through a guiding example for LRC$(10,2,2)$, as shown in Figure 6.

- First, in Figure 6(a), we set $\eta = 0$ to generate a maintenance-driven data placement, which has a low AMC and a high ADC. In Step 1, we do not co-locate any data block from the same local group. In Step 2, we choose five new racks (i.e., $R_1, R_2, \cdots, R_5$), where we spread one data block of G_1 and G_2 in each rack. In Step 3, we put P_1 and P_2 in R_6. In Step 4, we put Q_1 and Q_2 in R_7. While local decoding is feasible for each data block to reduce the amount of cross-rack transfers in maintenance mode, it enlarges the amount of cross-rack

transfers in regular mode. We take the degraded read of both D_1 and D_4 as examples. In regular mode, repairing D_1 and D_4 each retrieves five blocks, while in maintenance mode, repairing D_1 and D_4 each also retrieves five blocks, where local encoding is feasible for both blocks. Overall, the ADC and AMC of the data placement are both 5.

- Next, in Figure 6(b), we increase η to 1 to trade AMC for ADC. In Step 1, we co-locate D_1, D_2, and D_3 from G_1 in R_1, and co-locate D_6, D_7, and D_8 from G_2 in R_2. In Step 2, we put D_4 and D_9 in R_3, and put D_5 and D_{10} in R_4. In regular mode, the amount of cross-rack transfers for decoding is reduced, as each local group spans fewer racks. In maintenance mode, a proportion of data blocks now requires global decoding, such that the amount of cross-rack transfers for decoding increases. For example, in regular mode, decoding D_1 and D_4 each retrieves three blocks. In maintenance mode, decoding D_1 retrieves 11 blocks, as global decoding is required; decoding D_4 retrieves three blocks, as local decoding is feasible. Overall, the ADC decreases to 3, while the AMC increases to 7.8.
- Finally, in Figure 6(c), we set $\eta = 2$ to generate a repair-driven data placement. We co-locate all blocks of G_1 in R_1 and R_2, and co-locate all blocks of G_2 in R_3 and R_4. In regular mode, the amount of cross-rack transfers for decoding is minimized, as each local group spans the minimum number of racks. In maintenance mode, decoding a block needs a large amount of cross-rack transfers, as global decoding is required. For example, in regular mode, decoding D_1 and D_4 each retrieves one block. In maintenance mode, decoding D_1 and D_4 retrieves nine and eight blocks, respectively, where global decoding is required for both blocks. Overall, the ADC decreases to 1, while the AMC increases to 8.6.

B. Trade-off Analysis

We analyze the trade-off between AMC and ADC based on our modeling in §III-A. Specifically, for LRC(k,l,g), both ADC and AMC can be represented as a function of the configuration parameter η. We show that for most LRC parameters (k,l,g), when η increases, the ADC decreases while the AMC increases.

Modeling. Given the configuration parameter η, we show the assignment of modeling parameters of the corresponding data placement. To simplify our analysis, we first assume that b is a multiple of $g+1$, where we discuss the case when b is not divisible by $g+1$ later.

- To initialize the model, we set $d_{i,j}, p_{i,j}$, and q_i as 0 for each $i \in [1, k+l+g]$ and for each $j \in [1,l]$.
- In Step 1, for each $j \in [1,l]$, we set $d_{i,j} = g+1$ for each $i \in [(j-1)\eta+1, j\eta]$. It means that for each local group G_j, we co-locate $(g+1)\eta$ data blocks of G_j in η new racks (i.e., $R_{(j-1)\eta+1}, R_{(j-1)\eta+2}, \cdots, R_{j\eta}$), where each rack stores $g+1$ data blocks.
- In Step 2, for each $i \in [l\eta+1, (l-g-1)\eta+b]$ and for each $j \in [1,l]$, we set $d_{i,j} = 1$. It means that for each local group G_j, we spread the remaining $b-(g+1)\eta$ data blocks of G_j in $b-(g+1)\eta$ new racks (i.e., $R_{l\eta+1}, R_{l\eta+2}, \cdots, R_{(l-g-1)\eta+b}$). In each new rack, we co-locate l data blocks from l local groups.
- In Step 3, for each $j \in [1,l]$, we set $p_{(l-g-1)\eta+b+1,j} = 1$, meaning that we put the l local parity blocks in rack $R_{(l-g-1)\eta+b+1}$.
- In Step 4, we set $q_{(l-g-1)\eta+b+2} = g$, meaning that we put the g global parity blocks in rack $R_{(l-g-1)\eta+b+2}$.

When b is not divisible by $g+1$, we only discuss the case where $\eta = \lceil \frac{b}{g+1} \rceil$. In this case, for each $j \in [1,l]$, we set $d_{j\eta,j} = b \bmod (g+1)$ and $p_{j\eta,j} = 1$, meaning that we co-locate $b \bmod (g+1)$ data blocks and the local parity block of G_j in $R_{j\eta}$.

For the assignment of modeling parameters, we can verify that the constraints (Equations (1)-(4)) are all satisfied.

Analysis of ADC. We can express the ADC of the data placement as a function of η. For each local group G_j, we have $\delta_j = -g\eta + b + 1$. From Equation (6), we can derive the ADC as follows:

$$\mathrm{ADC}(\eta) = -g\eta + b. \tag{10}$$

It implies that when η increases, the ADC linearly decreases.

Analysis of AMC. Similarly, we can also express the AMC of the data placement as a function of η. Based on Equation (9), we calculate the amount of cross-rack transfers for repairing a data block in maintenance mode.

First, we focus on the data blocks placed in Step 1 (e.g., D_1 in Figure 6(b)), where repairing each data block requires global decoding, as each rack co-locates with multiple blocks from the same local group. For each rack R_i where $1 \le i \le l\eta$, it is only spanned by local group G_j, where $d_{i,j} = g+1$ and $p_{i,j} = 0$. From Equations (7) and (8), we can derive the amount of cross-rack transfers for global decoding as $\gamma_i = (l-g-2)g\eta + (g+1)b$. There are a total of $(g+1)l\eta$ data blocks stored in R_i that require global decoding, where the decoding of each block transfers γ_i blocks.

Next, we focus on the data blocks placed in Step 2 (e.g., D_4 in Figure 6(b)), where local decoding is feasible to repair each block, as each rack only stores one data block from each local group. The amount of cross-rack transfers for local decoding is $\delta_j - 1 = -g\eta + b$. There are a total of $-(g+1)l\eta + lb$ data blocks stored in R_i that are feasible for local decoding, where the decoding of each block transfers $-g\eta + b$ blocks.

Thus, we can derive the AMC as follows:

$$\mathrm{AMC}(\eta) = \tfrac{(l-g-1)(g+1)g}{b}\eta^2 + g^2\eta + b, \tag{11}$$

where the AMC is a quadratic function η, and the coding parameters (k,l,g) (where $b = \frac{k}{g}$) also affect how AMC changes with η. Let $\eta^* = \frac{bg}{2(g+1)(g+1-l)}$, where $\mathrm{AMC}(\eta)$ takes the extreme value at $\eta = \eta^*$.

- When $g = l-1$, $\mathrm{AMC}(\eta) = g^2\eta + b$, which is a linear function of η. It implies that when η increases, the AMC linearly increases.
- When $g < l-1$, as the coefficient of the quadratic term (i.e., $\frac{(l-g-1)(g+1)g}{b}$) is positive and $\eta^* < 0$, $\mathrm{AMC}(\eta)$ takes the minimum value when $\eta = \eta^*$. In our data placement scheme

where $\eta \geq 0$, it implies that when η increases, the AMC also increases.

- When $g > l-1$, as the coefficient of the quadratic term is negative and $\eta^* > 0$, AMC(η) takes the maximum value at $\eta = \eta^*$. In our data placement scheme where $0 \leq \eta \leq \lceil \frac{b}{g+1} \rceil$, if $\eta^* \geq \lceil \frac{b}{g+1} \rceil$, when η increases, the AMC also increases. Otherwise, if $\eta^* < \lceil \frac{b}{g+1} \rceil$, when η increases, the AMC first increases when $\eta \leq \eta^*$, then decreases when $\eta > \eta^*$. In this case, we can derive that $g > 2l-2$.

We argue that for most LRC coding parameters (where $g \leq 2l-2$), when η increases, the AMC increases. This is true for most LRC parameters reported in the literature, where the number of global parity blocks remains limited to suppress the storage overhead (e.g., $(12,2,2)$ in [19], $(6,3,2)$ in [21], $(20,4,2)$ in [16], $(48,4,3)$ in [20]).

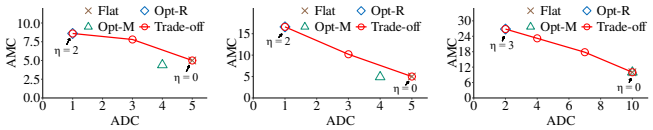
We now address the case when b is not a multiple of $g+1$. When $0 \leq \eta \leq \lceil \frac{b}{g+1} \rceil - 1$, the prior analysis still holds. Here, we only discuss how ADC and AMC vary when η increases from $\lceil \frac{b}{g+1} \rceil - 1$ to $\lceil \frac{b}{g+1} \rceil$. For each local group, the remaining b mod $(g+1)$ data blocks and the corresponding local parity block are co-located in a rack, such that δ_j decreases and hence the ADC decreases (see Equation (6)). On the other hand, for each of b mod $(g+1)$ data blocks, as more data blocks now require global decoding, the AMC increases.

V. Evaluation

We evaluate the repair performance in both regular and maintenance modes for different data placements via numerical analysis and testbed experiments. We aim to address the following questions: (i) Can our data placement scheme balance the trade-off between ADC and AMC? (ii) How do the coding parameters affect ADC and AMC? (iii) What is the degraded read performance in regular mode and maintenance mode of the data placements under different system configurations?

In our evaluation, we consider the following baseline data placements: (i) the flat data placement (denoted by *Flat*), (ii) the optimal repair scheme (denoted by *Opt-R*), and (iii) the optimal maintenance scheme (denoted by *Opt-M*). For Flat, the ADC and AMC are both b, where repairing a data block in both regular and maintenance modes retrieves blocks within the corresponding local group (§II-B). For Opt-R and Opt-M, we obtain them by solving the corresponding ILP problems (§III). We implement an ILP solver based on Gurobi [3], a highly optimized solver to address general ILP problems. Gurobi also supports solving objective functions with indicator functions (i.e., $I(.)$ in §III) [5]. Our implementation is written in Python with around 170 LoC.

We run the ILP solver on a Ubuntu 22.04 machine equipped with two 12-core 2.2 GHz Intel Xeon CPUs, 64 GiB RAM. For each optimization objective, the ILP solver finds a feasible assignment of parameters that minimizes the objective function. Note that the solving time of an ILP problem depends on the complexity of the objective function as well as the solution space. For practical consideration, we set a timeout for the solver to limit the search time, such that when the solver finishes



(a) $(10,2,2)$ (b) $(20,4,2)$ (c) $(30,3,3)$

Figure 7: Experiment A1: Trade-off analysis.

	$(10,2,2)$	$(20,4,2)$	$(30,3,3)$
ADC	0.01s	1.19s	1.97s
AMC	57.05s	>4h	>4h

Table I: Running time of the ILP solver.

before the timeout, it returns the optimal solution; otherwise, it returns the currently best feasible solution found within the time limit without guaranteed optimality [25].

For our configurable data placement scheme, we vary η from 0 to $\lceil \frac{b}{g+1} \rceil$ to generate $\lceil \frac{b}{g+1} \rceil + 1$ data placements, and denote each of them by *Trade-off-*η. We then obtain the trade-off curve between ADC and AMC. Note that Trade-off-0 corresponds to the maintenance-driven data placement, while Trade-off-$\lceil \frac{b}{g+1} \rceil$ corresponds to the repair-driven data placement.

A. Numerical Analysis

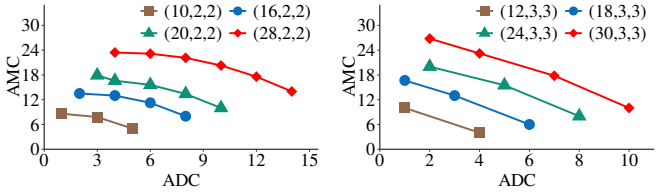
We study the ADC and AMC of LRC(k,l,g) for different data placements. For each data placement, we calculate the ADC and AMC based on Equations (6) and (9), respectively.

Experiment A1: Trade-off analysis. Figure 7 shows the results of $(k,l,g) = (10,2,2)$, $(20,4,2)$, and $(30,3,3)$. For our data placement scheme, when η increases, the AMC increases while the ADC decreases. In particular, Trade-off-2 (i.e., repair-driven) achieves the optimal ADC as Opt-R, while Trade-off-0 (i.e., maintenance-driven) has a slightly higher AMC than Opt-M. For example, for $(10,2,2)$, Trade-off-2 reduces the ADC of Opt-M by 75%, while Trade-off-0 reduces the AMC of Opt-R by 41.8% and increases the AMC of Opt-M by 13.6%. For Trade-off-1, it reduces the ADC of Opt-M by 25% and reduces the AMC of Opt-R by 9.3%, so it strives a balance between the ADC and AMC.

Table I shows the running time of the ILP solver to obtain the optimal schemes, where we set the timeout as four hours to allow the evaluation of different parameters [25]. While we can find Opt-R quickly (which takes less than 2 s), finding Opt-M takes a significantly longer time, especially when k is large. One reason is that the objective function of AMC (i.e., Equation (9)) introduces a large number of variables in the ILP solver (e.g., from the indicator functions [5]), where the variables are highly correlated (e.g., from local decoding and global decoding). This leads to long time.

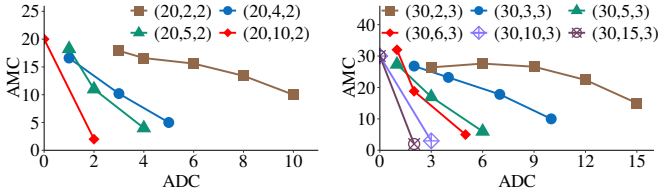
We further study the impact of coding parameters k, l, and g on ADC and AMC. For simplicity, we only consider data placements generated by our data placement scheme in the following discussions.

Experiment A2: Impact of k. We study the impact of k on the ADC and AMC. We consider $(l,g) = (2,2)$ and $(3,3)$, and vary k to up to 30. Figure 8 shows the results; note that a larger k allows more options for η, so there are more points on the curve of ADC and AMC. For the same η, both ADC



(a) $(l,g) = (2,2)$ (b) $(l,g) = (3,3)$

Figure 8: Experiment A2: Impact of k.



(a) $(k,g) = (20,2)$ (b) $(k,g) = (30,3)$

Figure 9: Experiment A3: Impact of l.

and AMC increase with k. The reason is that when k increases, each local group contains more blocks and hence spans more racks, such that a repair retrieves more blocks. For example, for $(24,3,3)$, when η increases from 0 to 2, the ADC decreases from 8 to 2, while the AMC increases from 8 to 20.

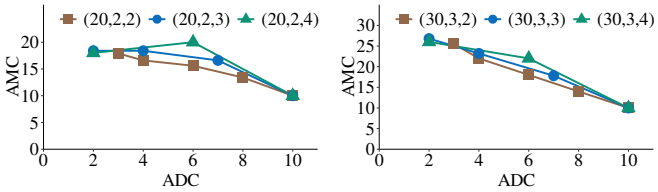
Experiment A3: Impact of l. We study the impact of l on the ADC and AMC. We consider $(k,g) = (20,2)$ and $(30,3)$ and vary l from 2 to $\frac{k}{2}$ (i.e., each local group has two data blocks). Figure 9 shows the results. For ADC, when l increases, the ADC decreases for the same η, as each local group contains fewer blocks and hence spans fewer racks, such that the degraded read in regular mode retrieves fewer blocks. For AMC, when l increases, the AMC increases for the same η, since more data blocks are aggregated in the same racks, where they require global decoding in maintenance mode. For example, for $(20,2,2)$, when η increases, the ADC decreases from 10 to 3, while the AMC increases from 10 to 17.9.

Experiment A4: Impact of g. We study the impact of g on the ADC and AMC. We consider $(k,l) = (20,2)$ and $(30,3)$ and vary g from 2 to 4. Figure 10 shows the results. When g increases, the trade-off curves appear close. When $\eta = 0$, it retrieves b blocks for both regular and maintenance modes independent of g. When $\eta = \lceil \frac{b}{g+1} \rceil$, the number of blocks retrieved for maintenance is close to k, meaning that most blocks of the stripe need to be retrieved for decoding. For example, for $(30,3,2)$, when η increases from 0 to 4, the ADC decreases from 10 to 3, while the AMC increases from 10 to 25.5. Note that for $(20,2,4)$, when η increases, the AMC first increases when $\eta = 1$, and then decreases when $\eta = 2$. It corresponds to the case when $g > 2l-2$ (§IV-B).

B. Testbed Evaluation

We present testbed evaluation results. Our goal is to show that the repair performance in a real distributed storage system conforms to our findings in numerical analysis.

Prototype implementation. We prototype different data placements atop Hadoop 3.3.4 HDFS [2]. Our implementation builds with OpenEC [22], a middleware system that runs atop HDFS



(a) $(k,l) = (20,2)$ (b) $(k,l) = (30,3)$

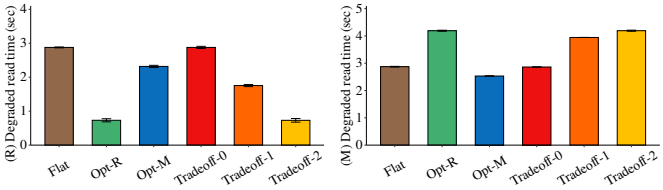
Figure 10: Experiment A4: Impact of g.

and provides a unified erasure coding interface based on direct acyclic graphs. HDFS comprises a NameNode for storage management and multiple DataNodes for data storage. HDFS organizes data in fixed-size blocks and supports rack-based settings [4]. Our implementation extends OpenEC to support the repair of LRCs in both regular and maintenance modes in rack-based settings, where we use ISA-L [6] to implement the erasure coding functionalities. It adds 3.1 K LoC to OpenEC.

Evaluation methodology. We evaluate our prototype in small-scale rack-based settings. We set up a local cluster with 17 machines, each of which has a quad-core 3.4 Ghz Intel Core i5 CPU, 16 GiB RAM, and a 7200 RPM 1 TB SATA hard disk. All machines are installed with Ubuntu 22.04 and are connected via a 10 Gbps Ethernet switch. To simulate a rack-based setting, we assign one dedicated machine to act as the network core, and direct all cross-rack traffic through the network core. We also assign one machine as the NameNode and the remaining machines as the DataNodes. We use Wondershaper tool [7] to configure the outgoing bandwidth of the network core.

We use the following default configurations. We choose LRC$(10,2,2)$ as the coding parameters (as in [21]) that fit the cluster size, and evaluate the degraded read performance in both regular and maintenance modes. We evaluate Flat, Opt-S, Opt-M, and our data placement scheme, where we vary η from 0 to 3. We set the block size as 64 MiB, the packet size (i.e., the smallest data unit for data transfer) as 1 MiB, and the cross-rack bandwidth as 1 Gbps [33] (i.e., the ratio of inner-rack to cross-rack bandwidth is 10:1). We also vary the block size and cross-rack bandwidth in our experiments.

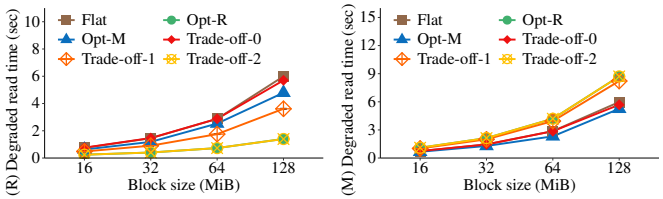
We measure the *degraded read time* in both regular and maintenance modes. The degraded read time is defined as the time from issuing a degraded read request to a failed block until the repaired block has been retrieved. For each data placement, we configure the rack topology in HDFS, and assign one DataNode as the helper node to issue a degraded read. We first write an LRC stripe to HDFS and ensure that the helper node does not store any block of the stripe. We next erase a block from the stripe. We then issue a degraded read to the erased data block, in which the helper node queries the NameNode for the DataNodes storing the available blocks and retrieves the blocks for decoding. In regular mode, we configure the helper node to reside in the rack storing the failed block. In maintenance mode, we configure the helper node to reside in a rack that does not store any block of the stripe. We measure the degraded time each of the k data blocks in a stripe, and average the results over the k data blocks. We



(a) Degraded read time in regular mode

(b) Degraded read time in maintenance mode

Figure 11: Experiment B1: Overall performance.



(a) Degraded read time in regular mode

(b) Degraded read time in maintenance mode

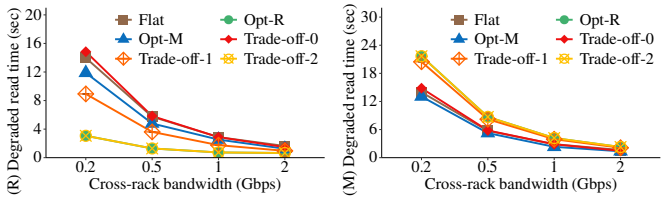
Figure 12: Experiment B2: Impact of block size.

report the results over 10 runs, with an error bar showing the 95% confidence interval based on the student's t-distribution (most error bars are invisible due to small variations).

Experiment B1: Overall performance. We first evaluate the degraded read time in both regular and maintenance modes for different data placements. Figure 11 shows the results. In maintenance mode, Trade-off-0 has a slightly higher degraded read time than Opt-M. When η decreases, the degraded read time in regular mode also decreases, while the degraded read time in maintenance mode increases, which complies with our prior analysis (§V-A). In regular mode, Trade-off-2 has almost the same repair time as Opt-R. For example, in regular mode, Trade-off-2 reduces the degraded read times of Opt-M and Flat by 68.5% and 74.6%, respectively. In maintenance mode, Trade-off-0 slightly increases the degraded read time of Opt-M by 13.1%, but reduces the degraded read time of Opt-R by 31.7%. For Trade-off-1, in regular mode, it reduces the degraded read time of Flat by 39.0%; in maintenance mode, it reduces the degraded read time of Opt-R by 5.9%.

Experiment B2: Impact of block size. We study the impact of block size on the degraded read time. We vary the block size from 16 MiB to 128 MiB. Figure 12 shows the results. When the block size increases, we observe a stable increase of degraded read time in both regular and maintenance modes, while we still observe a clear performance trade-off for different data placements. For example, when the block size is 32 MiB, Trade-off-1 reduces the degraded read time of Flat by 37.7% in regular mode, while it reduces the degraded read time of Opt-R by 6.2% in maintenance mode.

Experiment B3: Impact of cross-rack bandwidth. We study the impact of cross-rack bandwidth on the degraded read time. We vary the cross-rack bandwidth from 200 Mbps to 2 Gbps. Figure 13 shows the results. The degraded read time in both regular and maintenance modes fairly scales with the cross-rack bandwidth, which justifies our assumption that cross-rack bandwidth is the performance bottleneck (§II-A). For example, when the cross-rack bandwidth is 500 Mbps, Trade-off-1 reduces the degraded read time of Flat by 37.1% in regular mode, while it reduces the degraded read time of Opt-R by 5.6% in maintenance mode.



(a) Degraded read time in regular mode

(b) Degraded read time in maintenance mode

Figure 13: Experiment B3: Impact of cross-rack bandwidth.

VI. Related Work

Erasure coding has been reportedly deployed in data centers across geographical regions [10], [32]. Some studies focus on minimizing cross-rack transfers for the single-block repair [16]–[18], [34], [42], while some studies consider minimizing the cross-rack transfers for redundancy transitioning [40], [41]. Our work considers minimizing the cross-rack transfers for the single-block repair in both regular and maintenance modes and studies their performance trade-offs.

Some theoretical studies propose LRC constructions with different design objectives, such as minimizing the Hamming distance [35]–[37], minimizing the repair I/Os [14], [30], and minimizing the amount of network transfers in redundancy transitioning [23]. On the applied side, LRCs have been reportedly deployed in Azure [19], Facebook [33], Ceph [21], and Google [20]. Our work focuses on data placement designs of LRCs that balance the repair performance in regular and maintenance modes.

Several studies explore the design trade-offs in erasure coding from different aspects. Some studies focus on the trade-off between repair performance and storage redundancy (e.g., in regenerating codes [12], [28], and LRCs [21]). Wu et al. [41] study the trade-off between repair and redundancy transitioning performance. Our work considers the repair performance trade-off between regular and maintenance modes.

VII. Conclusion

We analyze the performance trade-off in repair and maintenance operations of LRCs in rack-based data centers. We design a configurable data placement scheme that operates along trade-off between repair and maintenance operations subject to fault tolerance constraints. Our evaluation via numerical analysis and testbed experiments demonstrates the effectiveness of our data placement scheme in balancing the performance trade-off between repair and maintenance operations.

Acknowledgements

This work was supported in part by National Natural Science Foundation of China NSFC (62202440 and 62302175) and Research Grants Council of Hong Kong (AoE/P-404/18). The corresponding author is Si Wu.

REFERENCES

[1] Erasure coding in Ceph. https://docs.ceph.com/en/latest/rados/operations/erasure-code/.

[2] Erasure coding in Hadoop 3.3.4. https://hadoop.apache.org/docs/r3.3.4/hadoop-project-dist/hadoop-hdfs/HDFSErasureCoding.html.

[3] Gurobi optimizer. http://www.gurobi.com/.

[4] Hadoop rack awareness. https://hadoop.apache.org/docs/r3.3.4/hadoop-project-dist/hadoop-common/RackAwareness.html.

[5] Indicator constraints in Gurobi optimizer. https://www.gurobi.com/documentation/current/refman/py_model_agc_indicator.html.

[6] ISA-L. https://github.com/intel/isa-l.

[7] Wondershaper. https://github.com/magnific0/wondershaper.

[8] Faraz Ahmad, Srimat T. Chakradhar, Anand Raghunathan, and T. N. Vijaykumar. ShuffleWatcher: Shuffle-aware scheduling in multi-tenant MapReduce clusters. In *Proc. of USENIX ATC*, 2014.

[9] Brian Beach. Backblaze Vaults: Zettabyte-scale cloud storage architecture. https://www.backblaze.com/blog/vault-cloud-storage-architecture/, 2019.

[10] Yu Lin Chen, Shuai Mu, Jinyang Li, Cheng Huang, Jin Li, Aaron Ogus, and Douglas Phillips. Giza: Erasure coding objects across global data centers. In *Proc. of the USENIX ATC*, 2017.

[11] Mosharaf Chowdhury, Srikanth Kandula, and Stoica Ion. Leveraging endpoint flexibility in data-intensive clusters. In *Proc. of ACM SIGCOMM*, 2013.

[12] Alexandros G Dimakis, P Brighten Godfrey, Yunnan Wu, Martin J Wainwright, and Kannan Ramchandran. Network coding for distributed storage systems. *IEEE Trans. on Information Theory*, 56(9):4539–4551, 2010.

[13] Daniel Ford, François Labelle, Florentina I. Popovici, Murray Stokely, Van-Anh Truong, Luiz Barroso, Carrie Grimes, and Sean Quinlan. Availability in globally distributed storage systems. In *Proc. of USENIX OSDI*, 2010.

[14] Parikshit Gopalan, Cheng Huang, Huseyin Simitci, and Sergey Yekhanin. On the locality of codeword symbols. *IEEE Trans. on Information theory*, 58(11):6925–6934, 2012.

[15] Shujie Han, Patrick P. C. Lee, Fan Xu, Yi Liu, Cheng He, and Jiongzhou Liu. An In-Depth study of correlated failures in production SSD-Based data centers. In *Proc. of USENIX FAST*, 2021.

[16] Yuchong Hu, Liangfeng Cheng, Qiaori Yao, Patrick P. C. Lee, Weichun Wang, and Wei Chen. Exploiting combined locality for wide-stripe erasure coding in distributed storage. In *Proc. of USENIX FAST*, 2021.

[17] Yuchong Hu, Patrick P. C. Lee, and Xiaoyang Zhang. Double regenerating codes for hierarchical data centers. In *Proc. of IEEE ISIT*, 2016.

[18] Yuchong Hu, Xiaolu Li, Mi Zhang, Patrick P. C. Lee, Xiaoyang Zhang, Pan Zhou, and Dan Feng. Optimal repair layering for erasure-coded data centers: From theory to practice. *ACM Trans. on Storage*, 13(4):33, 2017.

[19] Cheng Huang, Huseyin Simitci, Yikang Xu, Aaron Ogus, Brad Calder, Parikshit Gopalan, Jin Li, and Sergey Yekhanin. Erasure coding in Windows Azure storage. In *Proc. of USENIX ATC*, 2012.

[20] Saurabh Kadekodi, Shashwat Silas, David Clausen, and Arif Merchant. Practical design considerations for wide locally recoverable codes (LRCs). In *Proc. of USENIX FAST*, 2023.

[21] Oleg Kolosov, Gala Yadgar, Matan Liram, Itzhak Tamo, and Alexander Barg. On fault tolerance, locality, and optimality in locally repairable codes. In *Proc. of USENIX ATC*, 2018.

[22] Xiaolu Li, Runhui Li, Patrick P. C. Lee, and Yuchong Hu. OpenEC: Toward unified and configurable erasure coding management in distributed storage systems. In *Proc. of USENIX FAST*, 2019.

[23] Francisco Maturana and KV Rashmi. Locally repairable convertible codes: Erasure codes for efficient repair and conversion. In *Proc. of IEEE ISIT*, 2023.

[24] Subramanian Muralidhar, Wyatt Lloyd, Sabyasachi Roy, Cory Hill, Ernest Lin, Weiwen Liu, Satadru Pan, Shiva Shankar, Viswanath Sivakumar, Linpeng Tang, and Sanjeev Kumar. f4: Facebook's warm BLOB storage system. In *Proc. of USENIX OSDI*, 2014.

[25] Aviv Nachman, Gala Yadgar, and Sarai Sheinvald. GoSeed: Generating an optimal seeding plan for deduplicated storage. In *Proc. of the USENIX FAST*, 2020.

[26] Andreas-Joachim Peters, Michal Kamil Simon, and Elvin Alin Sindrilaru. Erasure coding for production in the EOS open storage system. In *Proc. of CHEP*, 2019.

[27] Shaya Potter and Jason Nieh. Reducing downtime due to system maintenance and upgrades. In *Proc. of USENIX FAST*, 2005.

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

[29] K. V. Rashmi, Nihar B. Shah, Dikang Gu, Hairong Kuang, Dhruba Borthakur, and Kannan 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.

[30] Ankit Singh Rawat, Dimitris S Papailiopoulos, Alexandros G Dimakis, and Sriram Vishwanath. Locality and availability in distributed storage. *IEEE Trans. on Information theory*, 62(8):4481–4493, 2012.

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

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

[33] Maheswaran Sathiamoorthy, Megasthenis Asteris, Dimitris Papailiopoulos, Alexandros G Dimakis, Ramkumar Vadali, Scott Chen, and Dhruba Borthakur. XORing elephants: novel erasure codes for big data. *Proc. of the VLDB Endowment*, 6(5):325–336, 2013.

[34] Zhirong Shen, Jiwu Shu, and Patrick P. C. Lee. Reconsidering single failure recovery in clustered file systems. In *Proc. of IEEE/IFIP DSN*, 2016.

[35] Natalia Silberstein, Ankit Singh Rawat, O Ozan Koyluoglu, and Sriram Vishwanath. Optimal locally repairable codes via rank-metric codes. In *Proc. of IEEE ISIT*, 2013.

[36] Itzhak Tamo and Alexander Barg. A family of optimal locally recoverable codes. *IEEE Trans. on Information Theory*, 60(8):4661–4676, 2014.

[37] Itzhak Tamo, Dimitris S Papailiopoulos, and Alexandros G Dimakis. Optimal locally repairable codes and connections to matroid theory. *IEEE Trans. on Information theory*, 62(12):6661–6671, 2012.

[38] Ashish Vulimiri, Carlo Curino, P. Brighten Godfrey, Thomas Jungblut, Jitu Padhye, and George Varghese. Global analytics in the face of bandwidth and regulatory constraints. In *Proc. of USENIX NSDI*, 2015.

[39] Hakim Weatherspoon and John D. Kubiatowicz. Erasure coding vs. replication: A quantitative comparison. In *Proc. of IPTPS*, 2002.

[40] Si Wu, Qingpeng Du, Patrick PC Lee, Yongkun Li, and Yinlong Xu. Optimal data placement for stripe merging in locally repairable codes. In *IEEE INFOCOM*, 2022.

[41] Si Wu, Zhirong Shen, and Patrick P. C. Lee. On the optimal repair-scaling trade-off in locally repairable codes. In *Proc. of IEEE INFOCOM*, 2020.

[42] Guofeng Yang, Huangzhen Xue, Yunfei Gu, Chentao Wu, Jie Li, Minyi Guo, Shiyi Li, Xin Xie, Yuanyuan Dong, and Yafei Zhao. XHR-Code: An efficient wide stripe erasure code to reduce cross-rack overhead in cloud storage systems. In *Proc. of IEEE SRDS*, 2022.