Optimal Repair-Scaling Trade-off in Locally Repairable Codes: Analysis and Evaluation

Si Wu, Zhirong Shen, Patrick P. C. Lee, Yinlong Xu

Abstract—How to improve the repair performance of erasure-coded storage is a critical issue for maintaining high reliability of modern large-scale storage systems. Locally repairable codes (LRC) are one popular family of repair-efficient erasure codes that mitigate the repair bandwidth and are deployed in practice. To adapt to the changing demands of access efficiency and fault tolerance, modern storage systems also conduct frequent scaling operations on erasure-coded data. In this paper, we analyze the optimal trade-off between the repair and scaling performance of LRC in clustered storage systems. Specifically, we focus on two optimal repair-scaling trade-offs, and design placement strategies that operate along the two optimal repair-scaling trade-off curves subject to the fault tolerance constraints. We prototype and evaluate our placement strategies on a LAN testbed, and show that they outperform the conventional placement schemes in repair and scaling operations.

1 Introduction

As storage systems continue to scale, failures become commonplace [7]. To provide reliability guarantees for data storage even in the presence of failures, modern storage systems increasingly adopt *erasure coding* to maintain fault tolerance at low redundancy. Compared to traditional replication, erasure coding provably achieves higher degree of reliability at the same degree of redundancy [28], and has been widely deployed in enterprise storage systems [7], [11], [14], [22]. At a high level, erasure coding takes a set of original data blocks as input, and generates additional redundant blocks called *parity blocks*, such that all original data blocks can be reconstructed from a subset of available data and parity blocks.

To maintain data availability, storage systems need to perform frequent *repair* operations to recover any lost data from failures. However, erasure coding amplifies the network traffic and disk I/Os in repair operations [20]. Specifically, the repair of a single lost block incurs network transfers and disk I/Os of multiple available blocks for reconstruction. To mitigate the expensive repair cost introduced by erasure coding, many erasure code constructions have been proposed in the literature to improve repair efficiency. *Locally repairable codes (LRC)* [11], [14], [22], [25], in particular, are a new family of erasure codes that mitigate the network traffic and disk I/Os of repair with slight addition of storage redundancy. The main idea of LRC is to partition the original data blocks into several small-size local groups and generate a *local parity block* for each local group, such that the repair of any single failed block can be achieved within a short-size group. Due to its simplicity, ease of implementation, and the repair efficiency, LRC has been widely deployed in production [11], [14], [22].

In addition to repair, storage systems also perform frequent *scaling* operations to adapt to the requirements of fault tolerance and access efficiency. By scaling, we refer to the change of erasure coding parameters to balance the trade-off between access performance and storage efficiency [34]. Scaling is important for not only the expansion in storage capacity [36], but also adapting to the storage redundancy requirements with respect to the change of reliability [13].

In this paper, we analyze the interplay of both repair and scaling operations of LRC in *clustered storage systems*, which hierarchically organize nodes in multiple *clusters* such that the cross-cluster network bandwidth appears much more scarce than the inner-cluster bandwidth [2], [5], [27]. We show that the cross-cluster repair and scaling costs cannot be simultaneously minimized, and there exists some fundamental trade-offs between the repair and scaling performance. To the best of our knowledge, this is the first work that unveils the inherent optimal repair-scaling trade-off in erasure-coded storage. To summarize, our contributions are as follows.

- We derive the feasible data placement strategies subject to the single-cluster fault tolerance constraint, under which we show that the (cross-cluster) repair and scaling costs cannot be simultaneously minimized for any possible data placements (Section 3).
- We present the formal analysis on two optimal trade-offs between the repair and scaling performance of LRC in clustered storage systems (Section 4). We explore different data placement strategies that operate along the two optimal repair-scaling trade-off curves. Specifically, for a given (cross-cluster) scaling cost, we can find a placement strategy that minimizes the (cross-cluster) repair cost.
- We implement the two extreme points of our placement strategies in each trade-off curve: one on minimizing the scaling cost, and another on minimizing the repair cost.

- *S. Wu and Y. Xu are with the School of Computer Science and Technology, University of Science and Technology of China (emails: {siwu5938,ylxu}@ustc.edu.cn).*
- *Z. Shen is with the College of Informatics, Xiamen University. (email: zhirong.shen2601@gmail.com).*
- *P. Lee is with the Department of Computer Science and Engineering, The Chinese University of Hong Kong. (email: pclee@cse.cuhk.edu.hk).*
- *An earlier conference version of the paper appeared in IEEE International Conference on Computer Communications (INFOCOM 2020) [32]. In this journal version, we add the trade-off analysis between the downcoding cost and the repair cost of the compact LRC. We also add new evaluation results in both simulation and testbed experiments.*

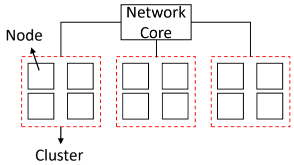


Figure 1. Example of a clustered storage system architecture.

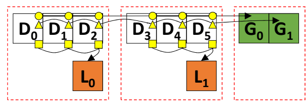


Figure 2. LRC with $(k, l, g, c) = (6, 2, 2, 3)$, where each dotted-line represents a cluster.

We conduct experiments on a LAN testbed. Experimental results show that, for the first trade-off curve, the placement with the minimum scaling cost reduces the scaling time of the baseline by up to 95.2%, while the placement with the minimum repair cost reduces the repair time of the baseline by up to 91.5%; for the second trade-off curve, the optimal-scaling placement reduces the scaling time of a second baseline by up to 97.8%, while the optimal-repair placement reduces the repair time of the second baseline by up to 73.0%. (Section 5).

2 Background

2.1 Clustered Storage System Architecture

We consider a clustered storage system modeled as a two-level hierarchical architecture [5], as shown in Figure 1. The system partitions *storage nodes* (or *nodes* in short) into multiple *clusters*, such that the nodes within a cluster are connected via the same switch, while multiple clusters are interconnected via a *network core*. A cluster can refer to a rack [11], [22] or a data center [3], [18]. The system organizes data in a collection of fixed-size *blocks*, which form the basic read/write units. Here, we assume that the cross-cluster bandwidth in clustered storage systems is much more scarce than the inner-cluster bandwidth [2], [5], [27]. Our goal is to minimize the cross-cluster network traffic in both repair and scaling operations.

2.2 Locally Repairable Codes

Basics of LRC. We present the basics of *locally repairable codes (LRC)*. We configure an LRC construction with four parameters (k, l, g, c), meaning that there are k data blocks (denoted by $D_0, D_1, \cdots, D_{k-1}$), l local parity blocks (denoted by $L_0, L_1, \cdots, L_{l-1}$), and g global parity blocks (denoted by $G_0, G_1, \cdots, G_{g-1}$), such that all $k+l+g$ data/parity blocks are stored in $k+l+g$ nodes located in c clusters. We call the set of $k+l+g$ data/parity blocks that are encoded together to be a *stripe*, and a large-scale storage system typically contains multiple stripes that are independently encoded. In the paper, our analysis focuses on a single stripe.

By storing each LRC stripe of blocks in multiple nodes and clusters, a storage system achieves both node-level and cluster-level fault tolerance. Since cluster failures are much less common than node failures in practice [18], we enforce the data placement of each LRC stripe such that the storage system provides only *single-cluster fault tolerance* (i.e., the data remains available under a single-cluster failure), while still providing multi-node fault tolerance. By storing multiple blocks in a single cluster, we can significantly reduce cross-cluster network traffic during repair [9], [10], [17], [19], [23].

There are various LRC constructions in the literature [11], [22], [25]. In this paper, we focus on the LRC construction based on Azure's Local Reconstruction Codes [11]. Specifically, LRC divides the k data blocks into l equal-size groups, assuming that k is divisible by l. It computes an XOR sum based on each group of $\frac{k}{l}$ data blocks to form a local parity block. It also computes the g global parity blocks from all k data blocks. Let b be the number of data blocks that encode into a local parity block, i.e., $b = \frac{k}{l}$. We call the collection of b data blocks and their corresponding local parity block to be a *local group*. Figure 2 gives an example of LRC for $(k, l, g, c) = (6, 2, 2, 3)$.

Repair of LRC. LRC is designed to mitigate the network traffic and disk I/Os for a single-block repair (which is much more common than a multi-block repair in practice [11], [20]) by limiting the repair within a local group. Given that transient failures account for the majority of failure events [7] and the temporarily unavailable data blocks are served by degraded reads, in this paper, we focus on the repair of a single unavailable data block in LRC.

We quantify the (single-data-block) repair performance of LRC in clustered storage systems as follows. To repair any unavailable data block, LRC retrieves other available blocks of the same local group. Let $C(D_i)$ denote the amount of cross-cluster network traffic being transferred for repairing a data block D_i. Then we define the *repair cost* of LRC (a.k.a. the average repair cost in [14]) as $\frac{1}{k}\sum_{i=0}^{k-1} C(D_i)$.

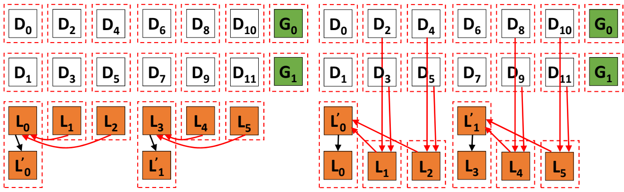
If each local group is put in the same cluster (e.g., in Figure 2), the repair cost is zero as there is no cross-cluster network transfer. If LRC adopts the flat data placement by storing each of the $k+l+g$ blocks of a stripe in a distinct cluster [11], [22], [34], then the repair cost is equal to b.

Scaling of LRC. We define *scaling* as the change of coding parameters in erasure coding. In this paper, we consider the change of two sets of LRC parameters in response to workload changes as described in [34]. Specifically, we consider two LRC constructions: (i) *fast LRC*, which stores more local parity blocks for higher repair performance (e.g., for hot data), and (ii) *compact LRC*, which stores fewer local parity blocks for higher storage efficiency (e.g., for cold data). Both fast and compact LRCs store the same numbers of data blocks and global parity blocks. We perform scaling operations to switch between the fast and compact LRCs to balance between access performance and storage efficiency.

We call the conversion *upcoding* when scaling from the fast LRC to the compact LRC, and *downcoding* when scaling from the compact LRC to the fast LRC. Note that upcoding and downcoding are also defined in [34]. Also, upcoding increases storage efficiency by reducing the number of local parity blocks, while downcoding reduces storage efficiency by increasing the number of local parity blocks. Accordingly, we define the *upcoding cost* and the *downcoding cost* as the amounts of cross-cluster network traffic transferred for upcoding and downcoding, respectively.

Note that both upcoding and downcoding happen frequently under dynamic workloads [15], [34]. Specifically, storage systems conduct upcoding when data becomes cold or the overall storage efficiency needs to be improved, while downcoding happens when data becomes hot again.

Figure 3 depicts the upcoding and downcoding for the fast LRC with $(12, 6, 2, 20)$ and the compact LRC with $(12, 2, 2, 16)$. Both codes are deployed under the flat data



(a) Upcoding (b) Downcoding

Figure 3. Upcoding and downcoding for fast LRC $(12, 6, 2, 20)$ and compact LRC $(12, 2, 2, 16)$, in which each block is stored in a distinct cluster.

placement (i.e., each block is stored in a distinct cluster). The upcoding operation (from $(12, 6, 2, 20)$ to $(12, 2, 2, 16)$) sends L_1 and L_2 across clusters to L_0 to compute a new local parity block $L_0^{'}$, and sends L_4 and L_5 across clusters to L_3 to compute a new local parity block $L_1^{'}$. Thus, the upcoding cost is 4. On the other hand, the downcoding operation (from $(12, 2, 2, 16)$ to $(12, 6, 2, 20)$) sends D_2 and D_3 to another cluster to compute L_1, and sends D_4 and D_5 to another cluster to compute L_2. It then sends L_1 and L_2 to the cluster that holds $L_0^{'}$ to compute L_0. The same is for L_3, L_4, and L_5. In total, the downcoding cost is 12.

3 Fault Tolerance And Motivation

3.1 Fault Tolerance of LRC

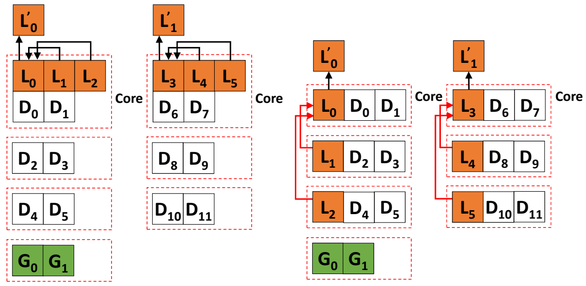
We first analyze the feasible data placement strategies subject to the single-cluster fault tolerance constraint (Section 2.2).

Lemma 1. *A (k, l, g) LRC can tolerate any $g + i$ block failures that span i local groups, where $1 \leq i \leq l$. However, the fault tolerance will fail if there exist i local groups with more than $g + i$ block failures.*

Proof. Consider a set of up to $g + i$ failed blocks that span i local groups ($1 \leq i \leq l$). Suppose that the failed blocks comprise d data blocks, x local parity blocks ($0 \leq x \leq i$), and y global parity blocks ($0 \leq y \leq g$), such that $d+x+y \leq g+i$. In other words, x local groups have failed local parity blocks, while $i - x$ local groups have failed data blocks only and their local parity blocks are available for repair. For each of the $i - x$ local groups with available local parity blocks, we can swap the available local parity block with one failed data block, and mark the data block as available and the local parity block as failed. By doing so, we can transform the set of failed blocks into $d - (i - x)$ failed data blocks, which can now be decoded by the $g - y$ surviving global parity blocks as $d - (i - x) \leq g - y$ [11]. After decoding the failed data blocks, we can recover all failed local/global parity blocks. The fault tolerance is maintained.

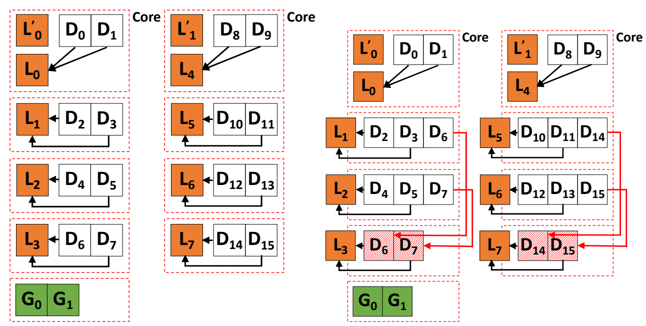
Based on the above analysis, we can also prove that a (k, l, g) LRC cannot tolerate any more than $g + i$ block failures if they span i local groups ($1 \leq i \leq l$). By swapping the available local parity blocks and the failed data blocks, we have $d-(i-x)$ failed data blocks that need to be decoded from $g - y$ available global parity blocks. If there are more than $g + i$ block failures, we have $d - (i - x) > g - y$, implying that there are more failed data blocks than the available global parity blocks. Thus, the data blocks cannot be repaired. □

By Lemma 1, we can deduce that if we provide single-cluster fault tolerance for a (k, l, g) LRC, we have to place no more than $g + i$ blocks that span i local groups in a cluster; otherwise, a cluster failure will cause data loss.



(a) Minimum upcoding cost (b) Minimum repair cost (fast)

Figure 4. Two data placements for the fast LRC $(12, 6, 2, 7)$ being upcoded to the compact LRC $(12, 2, 2, 7)$. The key is to determine the placement of the local parity blocks.



(a) Minimum downcoding cost (b) Minimum repair cost (compact)

Figure 5. Two data placements for the compact LRC $(16, 2, 2, 9)$ being downcoded to the fast LRC $(16, 8, 2, 9)$. The key is to design the placement of the data blocks.

3.2 Motivating Examples

We show via motivating examples that there is no data placement that can simultaneously minimize both the repair and scaling costs for LRC. Recall that repair is a common operation in the face of block failures, and both upcoding and downcoding are also common operations under dynamic workloads (Section 2). Thus, we compare the repair of the fast LRC and the upcoding, and the repair of the compact LRC and the downcoding.

Repair of fast LRC and upcoding. Figure 4 presents two data placements for a fast LRC with $(12, 6, 2, 7)$, which is to be upcoded to a compact LRC with $(12, 2, 2, 7)$. As l reduces from 6 to 2, we generate a local parity block of the compact LRC based on every three local parity blocks of the fast LRC.

In Figure 4(a), we place the local parity blocks in two clusters. Since there is no cross-cluster transfer in upcoding, the upcoding cost is zero, which is the minimum. On the other hand, the cost for repairing each of D_0 to D_5 is 0, 0, 1, 1, 1, 1, respectively, while the same is for D_6 to D_{11}. Thus, the repair cost is 0.67.

In Figure 4(b), we place each local group (the local parity block and its corresponding encoding data blocks) in one cluster. The repair of each data block can now be done in each cluster locally, so the repair cost is zero, which is the minimum. However, the upcoding needs to transfer four blocks across clusters. Thus, the upcoding cost is four.

Figure 4 shows that the repair and upcoding costs cannot be simultaneously minimized for any data placement.

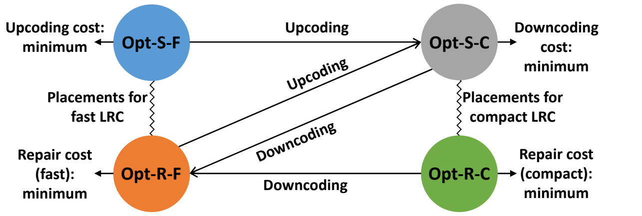


Figure 6. Roadmap for analyzing the trade-off between repair and scaling.

Repair of compact LRC and downcoding. Figure 5 gives two placement policies for a compact LRC with $(16, 2, 2, 9)$, which is to be downcoded to a fast LRC with $(16, 8, 2, 9)$. Since l increases from 2 to 8, we need to regenerate 8 local parity blocks of the fast LRC.

In Figure 5(a), each set of encoding data blocks is placed in one cluster. The downcoding can be done within each cluster locally, so the downcoding cost is zero, which is the minimum. However, as D_0-D_7 and $L_0^{'}$ span four clusters, the repair of any data block requires three cross-cluster transfers. The same holds for D_8-D_{15}. Therefore, the repair cost is 3.

In Figure 5(b), the placement of some data blocks is changed such that D_0-D_7 and $L_0^{'}$ reside in three clusters. The repair of any data block now needs only two cross-cluster transfers, while the same is for D_8-D_{15}. The repair cost is 2, which is the minimum (Section 4.5). However, the downcoding entails first relocating four data blocks across clusters and then recomputing the 8 local parity blocks in each cluster locally. Hence, the downcoding cost is 4.

Figure 5 further shows that there is no data placement for the compact LRC that simultaneously minimizes the repair and downcoding costs.

4 Trade-off Between Repair And Scaling

We now show that there exists some fundamental trade-offs between the repair and scaling (i.e., upcoding/downcoding) performance in clustered storage systems.

4.1 Roadmap

Figure 6 shows the overall roadmap of our analysis. Specifically, we first design data placements for the fast LRC that operate along the optimal trade-off curve from the minimum upcoding cost to the minimum repair cost of the fast LRC by deliberately determining the placement of the local parity blocks (Section 4.3). In particular, the placements that have the minimum upcoding cost and the minimum repair cost are called Opt-S-F and Opt-R-F, respectively. We further discuss how to downcode from the data placement for the compact LRC to Opt-S-F and Opt-R-F (Section 4.4).

We find that downcoding from the specific placement policy for the compact LRC (i.e., Opt-S-C) to Opt-R-F (i.e., a data placement for the fast LRC) has the minimum downcoding cost, but the placement of the data blocks amplifies the repair cost of the compact LRC. Thus, by modifying the placement of the data blocks, we further design placement policies for the compact LRC that operate along the optimal trade-off curve from the minimum downcoding cost to the minimum repair cost of the compact LRC (Section 4.5). The placement that gains the minimum repair cost is called Opt-R-C. We also show how to upcode from Opt-R-F to the two extreme placement strategies, i.e., Opt-S-C and Opt-R-C (Section 4.6).

4.2 Key Challenges

We emphasize that it is non-trivial to design repair-scaling trade-offs that are guaranteed to be optimal. In particular, when designing trade-off placements for the fast LRC, the key challenge is how to place the local parity blocks such that in each trade-off placement, the repair cost (fast) is minimized under the upcoding cost. In the design of the trade-off placements for the compact LRC, the difficulty lies in designing the placement of the data blocks, which is to guarantee that the repair cost (compact) is minimized given the downcoding cost in each trade-off placement.

4.3 Repair-Upcoding Trade-off Analysis

Preliminaries. We now formally analyze the trade-off between repair of the fast LRC and upcoding. We first present the definitions and notations, which we will follow throughout the paper. Since the access performance of LRC is mainly related to the number of local parity blocks, we consider the scaling operation that varies the number of local parity blocks as in [34]. To simplify our analysis, we consider the case where the number of local parity blocks for the fast LRC (denoted by l) is divisible by the number of local parity blocks for the compact LRC (denoted by l'), such that every local parity block of the compact LRC can be updated from $\frac{l}{l'}$ local parity blocks of the fast LRC. Let δ be the *scaling factor*, defined as $\delta = \frac{l}{l'}$.

For a local parity block L_i, we call the set of data blocks that generates L_i a *local data set*, denoted by $\mathcal{E}_i = \{D_{i\times b}, \cdots, D_{i\times b+b-1}\}$, where b is the number of data blocks that are encoded to L_i (defined in Section 2.2). For example, in Figure 4, the local data sets for L_0 and L_1 are $\mathcal{E}_0 = \{D_0, D_1\}$ and $\mathcal{E}_1 = \{D_2, D_3\}$, respectively. Note that $\mathcal{E}_i$ and L_i together form a local group. During upcoding, we convert every δ local groups of the fast LRC into to one local group of the compact LRC, and we call these δ local groups an *upcoding unit*. There are a total of l' upcoding units, and the i-th one is composed of the blocks $\mathcal{E}_{i\times\delta}, L_{i\times\delta}, \cdots, \mathcal{E}_{(i+1)\times\delta-1}, L_{(i+1)\times\delta-1}$. For example, in Figure 4, we have $\delta = 3$, and there are two upcoding units: (i) $\{\mathcal{E}_0, L_0, \mathcal{E}_1, L_1, \mathcal{E}_2, L_2\}$, and (ii) $\{\mathcal{E}_3, L_3, \mathcal{E}_4, L_4, \mathcal{E}_5, L_5\}$.

For each upcoding unit, we define a *core cluster* as the cluster that stores the δ local parity blocks and aggregates them into one local parity block of the compact LRC. For example, in Figure 4(a), a core cluster stores L_0, L_1, and L_2 and encodes them into $L_0^{'}$, while another core cluster stores L_3, L_4, and L_5 and encodes them into $L_1^{'}$. Suppose that $b \leq g$, such that a local data set can be entirely stored in one cluster without breaking single-cluster fault tolerance. Let θ be the maximum number of local data sets that can be collocated with their corresponding local parity blocks in one cluster. By Lemma 1, the number of data and local parity blocks (i.e., $\theta \times b + \theta$), which span θ local groups, cannot exceed $g + \theta$. Thus, we can calculate θ as $\theta = \lfloor \frac{g}{b} \rfloor$.

For example, in Figure 4, every $\theta = 1$ local data set can be collocated with its local parity block in one cluster. Our analysis focuses on $b \leq g$, while we will later show that the analysis for $b > g$ is similar.

Main idea. Our trade-off analysis between repair of the fast LRC and upcoding is organized as follows. (i) We first design the placement of the local parity blocks to achieve the globally minimum upcoding cost. (ii) Given the condition that the upcoding cost is minimized, we vary the locations of the data blocks to minimize the repair cost; note that the repair cost is not necessarily globally minimum (Section 3.2). (iii) We gradually relocate the local parity blocks, so as to trade the increased upcoding cost for the decreased repair cost and finally achieve the globally minimum repair cost. The number of clusters (i.e., c) is determined by the placement policy. Since the global parity blocks do not participate in repair and scaling operations, we place them in a dedicated cluster and omit their discussion in our analysis.

Guiding example. For better understanding of our analysis, we use a guiding example that upcodes from the fast LRC with $(12, 6, 2, 7)$ to the compact LRC with $(12, 2, 2, 7)$.

- *Minimizing upcoding cost:* First, from Figure 7(a), we place L_0, L_1, and L_2 in a core cluster, and place L_3, L_4, and L_5 in another core cluster. We perform upcoding in each core cluster by aggregating the three local parity blocks without any cross-cluster transfer. The upcoding cost is zero.
- *Minimizing repair cost under minimum upcoding cost:* Next, in Figure 7(b), we place $\mathcal{E}_0$ in one core cluster, and place $\mathcal{E}_1$ and $\mathcal{E}_2$ in two different clusters. We place $\mathcal{E}_3$, $\mathcal{E}_4$, and $\mathcal{E}_5$ in the same way. We can show that the cost for repairing each data block in $\mathcal{E}_0$ and $\mathcal{E}_3$ is zero, while the cost for repairing each data block in $\mathcal{E}_1$, $\mathcal{E}_2$, $\mathcal{E}_4$, and $\mathcal{E}_5$ is one. The repair cost of the fast LRC is $\frac{4}{6} = 0.67$. We will show that this repair cost is minimized under the minimum upcoding cost.
- *Trading increased upcoding cost for decreased repair cost:* In this example, we see that the main reason that causes non-zero repair cost is the separate placement of the local parity block and its corresponding local data set. For example, in Figure 7(b), L_1 is stored in the core cluster, while $\mathcal{E}_1$ is stored in a different cluster; the same holds for L_2, L_4, and L_5. Thus, we can gradually move one local parity block from the core cluster to the cluster where its local data set resides. In Figure 7(c), we move L_1 to the cluster that holds $\mathcal{E}_1$. By doing so, we reduce the repair cost by $\frac{1}{6} = 0.17$, while the upcoding cost increases by one (the core cluster must now retrieve the relocated L_1 across cluster for upcoding). We will also show that the repair cost in Figure 7(c) (i.e., $\frac{3}{6} = 0.5$) is minimized subject to the upcoding cost of one.
- *Minimizing repair cost:* Finally, in Figure 7(d), we move L_1, L_2, L_4, and L_5 to the clusters where their corresponding local data sets reside. As the repair can now be executed without any cross-cluster traffic, the repair cost is zero.

Algorithm design. We now present a data placement algorithm (Algorithm 1) that is guaranteed to operate on the optimal trade-off curve from the minimum upcoding cost to the minimum repair cost of the fast LRC. The input is a parameter x that decides the operation point in the trade-off curve, while the output is a placement for the fast LRC.

(i) Placing local parity blocks to minimize upcoding cost to zero. We first gather the δ local parity blocks into the selected core cluster in each upcoding unit (lines 3-4), such that we can complete upcoding within each core cluster. By doing this, the upcoding cost is zero. For example, in Figure 7(a), we store every $\delta = 3$ local parity blocks in each core cluster.

(ii) Placing data blocks to further minimize repair cost ($x = 0$). We next determine the locations of the data blocks to

Algorithm 1 Upcoding-repair trade-off placement

Input: Integer x $(0 \le x \le (\delta - \theta) \times l')$, which is a multiple of θ
Output: A placement for the fast LRC

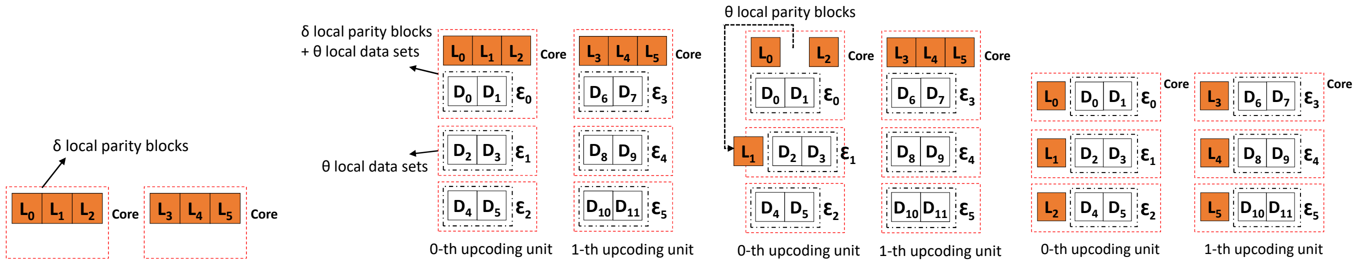
```
1:  for the i-th (0 ≤ i ≤ l' − 1) upcoding unit do
2:      // Minimizing upcoding cost
3:      Select a new core cluster
4:      Put L_{i×δ}, ···, L_{(i+1)×δ−1} in the core cluster
5:      // Further minimizing repair cost
6:      for j = 0 to θ − 1 do
7:          Put E_{i×δ+j} into the core cluster
8:      end for
9:      for j = θ to δ − 1 do
10:         if j mod θ = 0 then
11:             Select a new cluster
12:         end if
13:         Put E_{i×δ+j} into the new cluster
14:     end for
15: end for
16: // Trading upcoding cost for repair cost
17: Move x local parity blocks from the core clusters to the
    clusters where their corresponding local data sets reside
```

minimize the repair cost subject to the minimized upcoding cost. We first focus on an upcoding unit. There are δ local parity blocks that are collocated in the core cluster, and we have to decide how to place the δ local data sets. If we put more local data sets also in the core cluster, then more data blocks can be repaired in the core cluster locally and the repair cost can be minimized. Note that the blocks in the core cluster span δ local groups, so the sum of the number of local parity blocks (i.e., δ) and the number of data blocks cannot exceed $g + \delta$ to promise single-cluster fault tolerance. As a result, we can put $\theta = \lfloor \frac{g}{b} \rfloor$ local data sets in the core cluster (lines 6-8), and the cost for repairing each data block in these θ local data sets is zero. For example, in Figure 7(b), we put one local data set (i.e., $\theta = 1$) into a core cluster.

For the remaining $\delta - \theta$ local data sets, We make sure that each local data set is entirely stored in one cluster (not the core cluster), such that the cost for repairing each data block in these $\delta - \theta$ local data sets is one, as the corresponding local parity block is in the core cluster. In this manner, the repair cost is minimized. We further collocate every θ local data sets into one different cluster (lines 9-14). Each such cluster will have $\theta \times b = \lfloor \frac{g}{b} \rfloor \times b \le g$ data blocks, hence complying with single-cluster fault tolerance. For example, in Figure 7(b), for the remaining two local data sets, we put each into a different cluster. The data blocks of other upcoding units are placed in the same way. We call this placement *Opt-S-F*, where the upcoding cost is zero and the repair cost is $\frac{\delta - \theta}{\delta} = 1 - \frac{\theta}{\delta}$.

The values of δ and θ significantly influence the upcoding and repair costs, which we discuss as follows.

- If $\delta \le \theta$, then we can directly put all δ local data sets of an upcoding unit in the core cluster. Under this placement, the repair and upcoding operations can be directly performed within the core cluster, and therefore the repair and upcoding costs are zero.
- In the case where $\delta > \theta$, for the remaining $\delta - \theta$ local data sets, we collocate every θ ones into a different cluster. If we collocate more than θ local data sets in a different cluster, then when we move the corresponding local parity blocks into this cluster, it will violate single-cluster fault tolerance. If we collocate less than θ, say s $(s < \theta)$ local data sets in a



(a) Minimum upcoding cost (b) Further minimizing repair cost (c) Trading upcoding for repair (d) Minimum repair cost

Figure 7. Illustration of the different steps of our placement policy for fast LRC $(12, 6, 2, 7)$ scaled to compact LRC $(12, 2, 2, 7)$.

different cluster, then after we relocate the s corresponding local parity blocks into this cluster, the repair cost reduces by $\frac{s}{l}$ while the upcoding cost increases by one. However, in our design, after we relocate the θ corresponding local parity blocks into this cluster, the repair cost reduces by $\frac{\theta}{l}$ while the upcoding cost increases by one. That is to say, the reduction of the repair cost is the most in our design while the upcoding cost increases by one.

- We assume that $\delta - \theta$ is divisible by θ, such that every θ local data sets can be stored in the selected cluster.

(iii) Relocating local parity blocks to trade upcoding cost for repair cost ($0 \leq x \leq (\delta - \theta) \times l'$). In Opt-S-F, for each upcoding unit, there are $\delta - \theta$ local parity blocks lying in the core cluster, while their local data sets are located in different clusters. If we move such local parity blocks to where their local data sets reside, then the repair cost can be further reduced.

Since we collocate every θ local data sets into one different cluster, we can move θ corresponding local parity blocks to this cluster. By doing this, we reduce the repair cost by $\frac{\theta}{l}$ as we move θ local parity blocks to be collocated with their local data sets, such that the cost for repairing any data block in these θ local data sets reduces from one to zero. During upcoding, we first apply partial encoding of the θ local parity blocks to calculate an XOR sum in this cluster, and then send the XOR sum to the core cluster. Thus, the upcoding cost increases by one. We move (θ local parity blocks) for one different cluster in a step, and in an *upcoding unit by upcoding unit* basis, to transform Opt-S-F into a placement that trades increased upcoding cost for decreased repair cost. Since there are $\delta - \theta$ local parity blocks (per upcoding unit) that can be moved and l' upcoding units, we can then move x local parity blocks for $\frac{x}{\theta}$ different clusters, where x is a multiple of θ and $0 \leq x \leq (\delta - \theta) \times l'$ (line 17). The repair cost reduces by $\frac{x}{l}$ and the upcoding cost increases by $\frac{x}{\theta}$ compared to those costs of Opt-S-F. Thus, the cost values of the placement derived in a transformation step are shown as follows.

$$\begin{aligned} \text{upcoding cost} &= \tfrac{x}{\theta} \\ \text{repair cost (fast)} &= 1 - \tfrac{\theta}{\delta} - \tfrac{x}{l}. \end{aligned} \tag{1}$$

For example, in Figure 7(c), we move L_1 to the cluster that holds $\mathcal{E}_1$, and we reduce the repair cost by $\frac{1}{6} = 0.17$ while the upcoding cost increases by one.

Note that each transformation step will not break the fault tolerance guarantee. Specifically, suppose that a core cluster has moved out t (t is a multiple of θ and $0 \leq t \leq \delta - \theta$) local parity blocks, it then remains $\delta - t$ local parity blocks and θ local data sets. The number of blocks is $(\theta \times b + \delta - t) \leq (g + \delta - t)$, and the blocks span $\delta - t$ local groups. For a different cluster that accommodates the relocated local parity blocks, it now has θ local parity blocks collocated with their corresponding local data sets. According to Lemma 1, both clusters can guarantee single-cluster fault tolerance.

The movements (line 17) guarantee that the repair cost of the placement derived in a transformation step (i.e., $1 - \frac{\theta}{\delta} - \frac{x}{l}$) is minimized given the upcoding cost (i.e., $\frac{x}{\theta}$), which can be readily deduced by the following Theorem (see the Appendix for detailed proof).

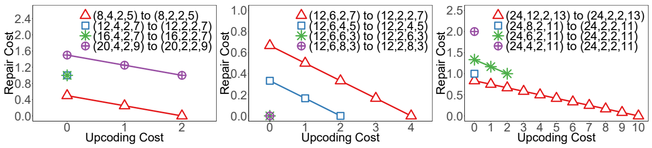
Theorem 1. *For any placement subject to single-cluster fault tolerance, if the upcoding cost is u, then the lower bound of the repair cost is $1 - \frac{\theta}{\delta} - \frac{u \times \theta}{l}$.*

(iv) Minimizing repair cost to zero ($x = (\delta - \theta) \times l'$). In the end (i.e., inputting $x = (\delta - \theta) \times l'$ in Algorithm 1), every θ local parity blocks lie together with their local data sets in one cluster. We call this placement *Opt-R-F*, where all repair operations can be done within each cluster locally, so the repair cost is zero. For example, in Figure 7(d), every local parity block lies together with its local data set in one cluster.

Extension to $b > g$. We now demonstrate that the analysis for $b > g$ is similar. If $b \geq g + 1$ and we further assume that $b \mod (g+1) \neq 0$. Since at most $g + 1$ data blocks of a local data set can be put into one cluster without breaking the fault tolerance guarantee, a local data set can be put into $\lfloor \frac{b}{g+1} \rfloor + 1$ clusters, where $\lfloor \frac{b}{g+1} \rfloor$ ones hold $g + 1$ data blocks each, and the remaining one holds $(m = b \mod (g+1)) \leq g$ data blocks. If we focus on the remaining m data blocks, then the above analysis directly applies. The only change is that the repair cost has to be added by a difference of $\lfloor \frac{b}{g+1} \rfloor$ as we have to retrieve an XOR sum of all blocks in each of the other $\lfloor \frac{b}{g+1} \rfloor$ clusters to repair any data block in a local data set.

If b is divisible by $(g + 1)$, then a local data set spans $\frac{b}{g+1}$ clusters with $g + 1$ data blocks each. To achieve the minimum upcoding cost, we gather every δ local parity blocks in each core cluster, and then we can put at most another g data blocks in a core cluster (Lemma 1). We cannot move data blocks to reduce the number of clusters (except the core cluster) a local data set spans. As a result, the cost for repairing any data block cannot be further reduced. Thus, in the placement with the minimum upcoding cost, the repair cost is also minimized.

Trade-off exemplification. We plot the upcoding cost and the repair cost of each placement and obtain a trade-off curve between the upcoding cost and the repair cost. We give some examples in Figure 8 to show the trade-off curve. Several findings are stated as follows.



(a) Varying k (b) Varying g (c) Varying l

Figure 8. Trade-off curve between the upcoding cost and the repair cost. Note that the curve covers all optimal data placements.

The most important trend is that as the upcoding cost increases, the repair cost decreases.

As shown in Figures 8(a) and 8(c), increasing k or decreasing l makes the overall repair cost increased. This is because b increases, which further results in larger $\lfloor \frac{b}{g+1} \rfloor$. For example, in Figure 8(a), in the case where $(k, l, g, c) = (8, 4, 2, 5)$, $\lfloor \frac{b}{g+1} \rfloor = 0$ (i.e., a local data set is entirely stored in one cluster), while in the case where $(k, l, g, c) = (20, 4, 2, 9)$, $\lfloor \frac{b}{g+1} \rfloor = 1$ (i.e., a local data set has to span two clusters).

Figure 8(b) tells that, as g increases, there are less trade-off points, and the overall repair cost decreases. The reason is that larger g means that more local data sets can be put into the core cluster, such that the cost for repairing any data block therein is zero. For example, in the case where $(k, l, g, c) = (12, 6, 2, 7)$, we can put one local data set into the core cluster, while in the case where $(k, l, g, c) = (12, 6, 4, 5)$, we can put two local data sets into the core cluster.

In Figure 8(a), the case where $(k, l, g, c) = (12, 4, 2, 7)$ only exhibits one point as b is divisible by $(g + 1)$, and the case where $(k, l, g, c) = (16, 4, 2, 7)$ also has one point because $\delta \leq \lfloor \frac{g}{m} \rfloor$ where $m = b \mod (g+1)$. The cases with one point in Figure 8(b) are due to that $\delta \leq (\theta = \lfloor \frac{g}{b} \rfloor)$, and those in Figure 8(c) are due to that b is divisible by $(g + 1)$.

4.4 Downcoding Procedures

We now address the downcoding procedure to restore the original layout of the fast LRC. We mainly consider the downcoding processes for Opt-S-F and Opt-R-F. Since the block layout for each upcoding unit is the same in Opt-S-F and Opt-R-F, we can readily derive that the downcoding operation for each upcoding unit is the same. Therefore, we conduct our analysis within a single upcoding unit.

Downcoding for Opt-S-F. In Opt-S-F, after upcoding, the δ local parity blocks in the core cluster are updated into a local parity block of the compact LRC. Downcoding is to rebuild the δ local parity blocks in the core cluster. Since there are θ local data sets (i.e., $\mathcal{E}_0$-$\mathcal{E}_{\theta-1}$) in the core cluster, we can first recalculate θ local parity blocks (i.e., L_0-$L_{\theta-1}$) within the core cluster. Note that there is a local parity block of the compact LRC (i.e., $L_0^{'}$) in the core cluster that can be utilized, we then have to retrieve at least $\delta - \theta - 1$ local parity blocks from other clusters. Thus, for each of $\mathcal{E}_\theta$-$\mathcal{E}_{\delta-2}$, we locate the cluster that holds it, calculate the corresponding local parity block, and send the local parity block to the core cluster. Finally, we use L_0-$L_{\delta-2}$ and $L_0^{'}$ to calculate $L_{\delta-1}$.

The downcoding cost is readily deduced as $(\delta - \theta - 1) \times l^{'}$, if $b \leq g$. For general parameters, when calculating each of the $\delta - 1$ local parity blocks, we should access partial encoded blocks from another $\lfloor \frac{b}{g+1} \rfloor$ clusters. Therefore,

$$\text{downcoding cost} = ((\delta - \lfloor \frac{g}{m} \rfloor - 1) + \lfloor \frac{b}{g+1} \rfloor \times (\delta - 1)) \times l^{'}, \quad (2)$$

where $m = b \mod (g + 1)$.

For example, in Figure 4(a), during downcoding, we can recalculate L_0 using $\mathcal{E}_0$ in the core cluster. We then calculate L_1 using $\mathcal{E}_1$ in a different cluster, and send L_1 to the core cluster. Finally, we use L_0, L_1, and $L_0^{'}$ to calculate L_2. The same is for L_3, L_4, and L_5, so the downcoding cost is two.

Downcoding for Opt-R-F. In Opt-R-F, downcoding is to restore the layout that every θ local parity blocks and their local data sets are collocated in one cluster. Thus, we can regenerate each local parity block within each cluster locally, implying that the downcoding cost is zero. For general parameters, when recalculating each local parity block, we should access partial encoded blocks from another $\lfloor \frac{b}{g+1} \rfloor$ clusters. Thus,

$$\text{downcoding cost} = \lfloor \frac{b}{g+1} \rfloor \times \delta \times l^{'} = \lfloor \frac{b}{g+1} \rfloor \times l. \quad (3)$$

For example, in Figure 4(b), we can regenerate L_0-L_5 within each cluster locally, so the downcoding cost is zero.

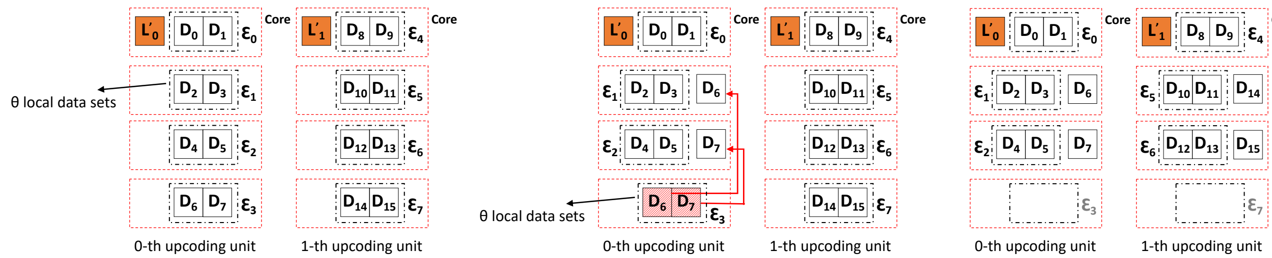
4.5 Repair-Downcoding Trade-off Analysis

Main observation. In Section 4.3, we have designed data placements for the fast LRC (e.g., Opt-S-F and Opt-R-F). After upcoding, we obtain a specific placement policy for the compact LRC. From Section 4.4, we find that downcoding from the specific placement policy for the compact LRC to Opt-R-F requires no cross-cluster transfer (under the parameter setting $b \leq g$), so the downcoding cost is minimized as zero. However, we cannot achieve the minimized repair cost of the compact LRC simultaneously, mainly because the placement of the data blocks is not so compact (Figure 5(a)). If we gradually relocate some data blocks to compactly place the data blocks, then we can trade the increased downcoding cost for the decreased repair cost, and finally achieve the minimized repair cost of the compact LRC (Figure 5(b)). This fuels us to explore the optimal trade-off between the minimum downcoding cost and the minimum repair cost of the compact LRC, which are realized by designing different placement policies for the compact LRC via deciding the placement of the data blocks.

New challenges. However, designing optimal trade-off placements for the compact LRC needs to address new challenges. First, we should find the minimum repair cost (compact) subject to single-cluster fault tolerance. Next, we should formally design placement policies to achieve the minimum downcoding cost (i.e., zero) and the minimum repair cost (compact). Finally, when we move data blocks to obtain a trade-off placement, we should guarantee that the repair cost (compact) is minimized under the downcoding cost (i.e., the trade-off placement should operate along the optimal trade-off curve).

Guiding example. We use an example that downcodes from the compact LRC with $(16, 2, 2, 9)$ to the fast LRC with $(16, 8, 2, 9)$ to guide the analysis.

- *Minimizing downcoding cost:* First, in Figure 9(a), each local data set is placed in one cluster (and $L_0^{'}$ and $L_1^{'}$ are placed in the core clusters). Downcoding can be performed in each cluster locally, so the downcoding cost is zero. We can see that the cost for repairing each data block is three, so the repair cost of the compact LRC is three.



(a) Minimum downcoding cost (b) Trading downcoding for repair (c) Minimum repair cost

Figure 9. Illustration of the different steps of our placement policy for compact LRC $(16, 2, 2, 9)$ scaled to fast LRC $(16, 8, 2, 9)$. Note that in the rightmost sub-figure, two clusters (at the bottom) will accommodate the relocated data blocks (i.e., D_6, D_7, D_{14}, and D_{15}) during downcoding.

Algorithm 2 Downcoding-repair trade-off placement

Input: Integer y $(0 \le y \le (\frac{\delta}{\theta} - \lceil\frac{\delta\times b+1}{g+1}\rceil) \times \theta \times b \times l^{'})$, which is a multiple of $\theta \times b$
Output: A placement for the compact LRC
1: **for** the i-th $(0 \le i \le l' - 1)$ upcoding unit **do**
2: `// Locating local parity block`
3: Select a new core cluster
4: Put $L_i^{'}$ in the core cluster
5: `// Minimizing downcoding cost`
6: **for** $j = 0$ to $\theta - 1$ **do**
7: Put $\mathcal{E}_{i\times\delta+j}$ into the core cluster
8: **end for**
9: **for** $j = \theta$ to $\delta - 1$ **do**
10: **if** $j \mod \theta = 0$ **then**
11: Select a new cluster
12: **end if**
13: Put $\mathcal{E}_{i\times\delta+j}$ into the new cluster
14: **end for**
15: **end for**
16: `// Trading downcoding cost for repair cost`
17: Move y data blocks from the clusters that hold the local data sets with larger ids to the clusters (except the core clusters) that hold the local data sets with smaller ids

- *Trading increased downcoding cost for decreased repair cost:* In Figure 9(a), we find that D_0-D_7 are placed in four clusters. Under the constraint of single-cluster fault tolerance, we can gradually move the data blocks of one local data set to different clusters. In Figure 9(b), we move the data blocks of $\mathcal{E}_3$ (i.e., D_6 and D_7) to different clusters. By doing so, we decrease the number of clusters the local group of the compact LRC spans. We can see that the cost for repairing each of D_0-D_7 is two while that for repairing each of D_8-D_{15} is three. Thus, the repair cost is reduced to 2.5. However, the downcoding cost increases by two (D_6 and D_7 must be relocated back across clusters for downcoding). We will prove that the repair cost in Figure 9(b) (i.e., 2.5) is minimized subject to the downcoding cost of two.
- *Minimizing repair cost:* Finally, in Figure 9(c), we move the data blocks of $\mathcal{E}_3$ (i.e., D_6 and D_7) and $\mathcal{E}_7$ (i.e., D_{14} and D_{15}) to distinct clusters. Now, the repair cost is reduced to two, which can be proved as the minimum.

Algorithm design. Algorithm 2 designs placement polices for the compact LRC that operate on the optimal trade-off curve from the minimum downcoding cost to the minimum repair cost of the compact LRC.

(i) Minimizing downcoding cost to zero (inputting $y = 0$). First, the local parity blocks of the compact LRC lie in the core clusters (lines 3-4). For example, in Figure 9(a), $L_0^{'}$ and $L_1^{'}$ reside in the two core clusters.

Next, the placement of the data blocks is determined. From Section 4.3, every θ local data sets are collocated into one cluster. Specifically, in an upcoding unit, θ local data sets reside in the core cluster (lines 6-8), while every θ local data sets in the remaining $\delta - \theta$ ones lie in a different cluster (lines 9-14). For example, in Figure 9(a), in an upcoding unit, every $\theta = 1$ local data set is put into one cluster (one local data set is in the core cluster). We call this placement *Opt-S-C*, where the downcoding cost is zero while the repair cost is $\frac{\delta}{\theta} - 1$ as the local group of the compact LRC is stored in $\frac{\delta}{\theta}$ clusters.

(ii) Relocating data blocks to trade downcoding cost for repair cost $(0 \le y \le (\frac{\delta}{\theta} - \lceil\frac{\delta\times b+1}{g+1}\rceil) \times \theta \times b \times l^{'})$. We first show that the placement of the data blocks in Opt-S-C is not compact, and we can relocate some data blocks to reduce the repair cost. In Opt-S-C, each cluster stores θ local data sets (i.e., $\theta \times b = \lfloor\frac{g}{b}\rfloor \times b \le g$ data blocks). By Lemma 1, each cluster (except the core cluster) can store up to $g+1$ data blocks with the fault tolerance guarantee. That is to say, the data blocks are not compactly stored into limited clusters. If we gradually move the data blocks of θ local data sets to different clusters, then the number of clusters the local group spans will be reduced and so the repair cost will be decreased accordingly.

We now figure out how many data blocks we can move to reduce the repair cost. By Lemma 1, we can put at most $g+1$ data/parity blocks of a local group of the compact LRC into one cluster. Since a local group has $\delta \times b$ data blocks and one local parity block of the compact LRC, the minimum number of clusters the local group spans is $\lceil\frac{\delta\times b+1}{g+1}\rceil$. In Opt-S-C, the number of clusters a local group spans is $\frac{\delta}{\theta}$. Thus, we can move the data blocks in $\frac{\delta}{\theta} - \lceil\frac{\delta\times b+1}{g+1}\rceil$ clusters to other clusters in an upcoding unit. In total, we can move $(\frac{\delta}{\theta} - \lceil\frac{\delta\times b+1}{g+1}\rceil) \times \theta \times b \times l^{'}$ data blocks for all upcoding units.

To simplify the analysis about the moving mechanism, we assume that g is divisible by b. We find that we cannot move data blocks to a core cluster as it already has $\theta \times b + 1 = \lfloor\frac{g}{b}\rfloor \times b + 1 = g+1$ data/parity blocks. For a different cluster, since it has $\theta \times b = g$ data blocks and it can accommodate up to $g+1$ data blocks, we can move one data block into it.

We now see the detailed moving procedures. We say the *id* of a local data set $\mathcal{E}_i$ is i. Based on Opt-S-C, we gradually move θ local data sets with larger ids to the clusters that store the local data sets with smaller ids (line 17). By doing this, we reduce the repair cost by $\frac{1}{l'}$ as the number of clusters one local group spans reduces by one. During downcoding, we must relocate the moved data blocks back and then locally regenerate the local parity blocks. Hence, the downcoding cost increases by $\theta \times b$. We move θ local data sets of a cluster in a step, and in an *upcoding unit by upcoding unit* manner, to transform Opt-S-C into a varied placement. Suppose that the number of moved data blocks is y $(0 \le y \le (\frac{\delta}{\theta} - \lceil \frac{\delta \times b+1}{g+1} \rceil) \times \theta \times b \times l^{'})$, which is a multiple of $\theta \times b$. The downcoding cost of a transformed placement increases by y and the repair cost reduces by $\frac{y}{\theta \times b \times l'}$ based on the costs of Opt-S-C.

$$\begin{aligned} &\text{downcoding cost} = y \\ &\text{repair cost (compact)} = \frac{\delta}{\theta} - 1 - \frac{y}{\theta \times b \times l'}. \end{aligned} \tag{4}$$

For example, in Figure 9(b), we move $\mathcal{E}_3$ (i.e., D_6 and D_7) to other clusters. The repair cost reduces by 0.5 and the downcoding cost increases by two.

The following Theorem guarantees that the repair cost in Equation (4) is minimized subject to the downcoding cost. The detailed proof is elaborated in the Appendix.

Theorem 2. *For any placement subject to single-cluster fault tolerance, if the downcoding cost is d, then the lower bound of the repair cost is* $\frac{\delta}{\theta} - 1 - \frac{d}{\theta \times b \times l'}$.

(iii) Minimizing repair cost (inputting $y = (\frac{\delta}{\theta} - \lceil \frac{\delta \times b+1}{g+1} \rceil) \times \theta \times b \times l^{'}$*).* Finally, we relocate all possible data blocks to minimize the number of clusters all local groups span to $\lceil \frac{\delta \times b+1}{g+1} \rceil$. The repair cost is then minimized to $\lceil \frac{\delta \times b+1}{g+1} \rceil - 1$. We call this placement *Opt-R-C*. For example, in Figure 9(c), we relocate $\mathcal{E}_3$ (i.e., D_6 and D_7) and $\mathcal{E}_7$ (i.e., D_{14} and D_{15}) to other clusters, and the repair cost is minimized to two.

Parameter analysis. The above analysis assumes that $b \le g$. If $b > g$, then downcoding from Opt-S-C to Opt-R-F inevitably incurs cross-cluster transfers (Section 4.4), so we cannot guarantee that the downcoding cost is minimized. Thus, the trade-off analysis is only valid for $b \le g$. Note that $b \le g$ covers a wide range of parameters.

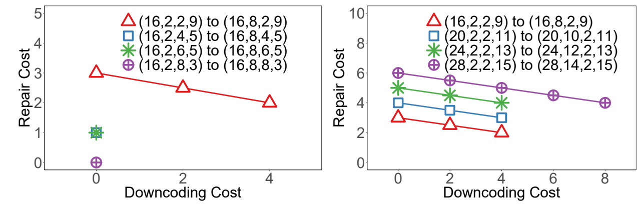
Trade-off examples. Figure 10 shows several examples of the trade-off curve from the minimum downcoding cost to the minimum repair cost of the compact LRC.

From Figure 10, we can see that as the downcoding cost increases, the repair cost of the compact LRC decreases.

As shown in Figure 10(a), as g increases, there are less trade-off points, and the overall repair cost decreases. The reason is that larger g means that more local data sets can be collocated into one cluster, such that the number of clusters the local group spans decreases. For example, for $(k, l^{'}, g, c) = (16, 2, 2, 9)$, we can place one local data set into a cluster, while for $(k, l^{'}, g, c) = (16, 2, 4, 5)$, we can place two local data sets into a cluster.

Note that increasing k or decreasing l will make $b > g$, which invalidates the downcoding-repair trade-off. Hence, we increase k as well as l, and guarantee that $b \le g$, which results in varied δ. Figure 10(b) shows that increasing δ makes the overall repair cost increased, and exhibits more trade-off points. This is because that as δ increases, the number of clusters the local group spans increases. For example, for $(k, l^{'}, g, c) = (16, 2, 2, 9)$, the number of clusters the local group spans is four, while for $(k, l^{'}, g, c) = (28, 2, 2, 15)$, the number of clusters the local group spans is seven.

In Figure 10(a), the cases with one point are due to that $\frac{\delta}{\theta} = \lceil \frac{\delta \times b+1}{g+1} \rceil$, and so in Opt-S-C, we minimize the downcoding cost and the repair cost simultaneously.



(a) Varying g (b) Varying δ

Figure 10. Trade-off curve between the downcoding and repair costs.

4.6 Upcoding Procedures

We now consider how to upcode from Opt-R-F (i.e., a placement for the fast LRC) to Opt-S-C and Opt-R-C. Our analysis also focuses on a single upcoding unit.

Upcoding to Opt-S-C. As described in Section 4.3, we send an XOR sum of the θ local parity blocks in each different cluster to the core cluster for upcoding. Since there are $\frac{\delta}{\theta} - 1$ different clusters, the upcoding cost is readily calculated as

$$\text{upcoding cost} = (\frac{\delta}{\theta} - 1) \times l^{'}. \tag{5}$$

For example, in Figure 5(a), we send L_1, L_2, and L_3 to L_0, and send L_5, L_6, and L_7 to L_4 for upcoding, so the upcoding cost is six.

Upcoding to Opt-R-C. It needs one more step, i.e., moving the data blocks in the local data sets with larger ids to other clusters. Thus, the upcoding cost is calculated as

$$\text{upcoding cost} = (\frac{\delta}{\theta} - 1) \times l^{'} + (\frac{\delta}{\theta} - \lceil \frac{\delta \times b + 1}{g+1} \rceil) \times \theta \times b \times l^{'}. \tag{6}$$

For example, in Figure 5(b), we send six local parity blocks as well as four data blocks across clusters for upcoding, so the upcoding cost is ten.

5 Evaluation

We evaluate via numerical analysis and testbed experiments the optimal placements under the fast LRC (Section 4.3): (i) Opt-S-F, the placement with the minimum upcoding cost, and (ii) Opt-R-F, the placement with the minimum repair cost of the fast LRC; and the optimal placements under the compact LRC (Section 4.5): (iii) Opt-S-C, the placement with the minimum downcoding cost, and (iv) Opt-R-C, the placement with the minimum repair cost of the compact LRC. We compare Opt-S-F and Opt-R-F with the *flat placement* (Flat) for the fast LRC, and Opt-S-C and Opt-R-C with Flat for the compact LRC. Note that Flat stores each block of a stripe in a distinct cluster [11], [22].

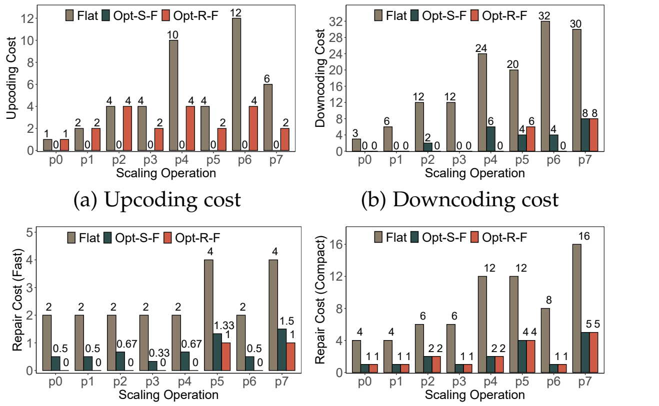
5.1 Numerical Analysis

Our numerical analysis considers four metrics: the repair cost of the fast LRC, the repair cost of the compact LRC, the upcoding cost, and the downcoding cost.

Flat. For Flat, the repair costs of the fast LRC and the compact LRC are $\frac{k}{l} = b$ and $\frac{k}{l'} = b'$, respectively (Section 2.2).

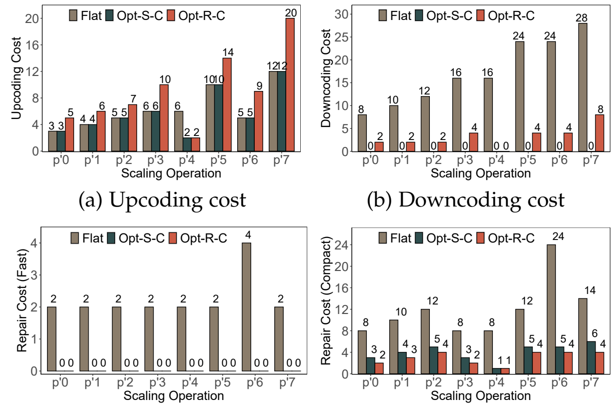
During upcoding, we send every $\delta - 1$ local parity blocks to the cluster that holds one remaining local parity block to encode into a local parity block of the compact LRC (per upcoding unit). Thus, the upcoding cost is calculated as

$$\text{upcoding cost} = (\delta - 1) \times l' = l - l'.$$



(a) Upcoding cost (b) Downcoding cost (c) Repair cost (fast) (d) Repair cost (compact)

Figure 11. Numerical results for Opt-S-F & Opt-R-F.



(a) Upcoding cost (b) Downcoding cost (c) Repair cost (fast) (d) Repair cost (compact)

Figure 12. Numerical results for Opt-S-C & Opt-R-C.

During downcoding, each of the first $\delta - 1$ local parity blocks is recalculated by sending its encoding data blocks across clusters to another cluster. The remaining one local parity block can be derived in two ways: (i) the same as the first $\delta - 1$ local parity blocks, and (ii) sending the $\delta - 1$ available local parity blocks to the cluster that holds the local parity block of the compact LRC. Hence,

$$\begin{aligned}\text{downcoding cost} &= \min\{\delta \times b \times l', ((\delta - 1) \times b + \delta - 1) \times l'\}\\ &= \min\{k,\ k + l - l' \times (b + 1)\}.\end{aligned}$$

Opt-S-F and Opt-R-F. The repair cost of the fast LRC of Opt-S-F and Opt-R-F is calculated using Equation (1), and the repair cost of the compact LRC of Opt-S-F and Opt-R-F is easily derived based on the layout after upcoding (Figure 4).

The upcoding cost of Opt-S-F and Opt-R-F is calculated using Equation (1). The downcoding cost of Opt-S-F is calculated using Equation (2), and that of Opt-R-F is calculated using Equation (3).

Opt-S-C and Opt-R-C. The repair cost of the fast LRC is zero as the layout of the fast LRC is Opt-R-F, while the repair cost of the compact LRC is calculated based on Equation (4).

The upcoding cost of Opt-S-C is calculated using Equation (5) while that of Opt-R-C is calculated using Equation (6). The downcoding cost is calculated based on Equation (4).

Scaling operations for Opt-S-F and Opt-R-F. We consider eight sets of scaling operations from (k, l, g, c) to (k, l', g, c), denoted by p_0 to p_7:

p_0: $(4, 2, 2, 3) \leftrightarrow (4, 1, 2, 3)$ p_1: $(8, 4, 2, 5) \leftrightarrow (8, 2, 2, 5)$
p_2: $(12, 6, 2, 7) \leftrightarrow (12, 2, 2, 7)$ p_3: $(12, 6, 4, 5) \leftrightarrow (12, 2, 4, 5)$
p_4: $(24, 12, 4, 7) \leftrightarrow (24, 2, 2, 4, 7)$ p_5: $(24, 6, 2, 11) \leftrightarrow (24, 2, 2, 11)$
p_6: $(32, 16, 4, 9) \leftrightarrow (32, 4, 4, 9)$ p_7: $(32, 8, 2, 13) \leftrightarrow (32, 2, 2, 13)$

Scaling operations for Opt-S-C and Opt-R-C. We consider eight sets of scaling operations from (k, l', g, c) to (k, l, g, c), denoted by p'_0 to p'_7:

p'_0: $(8, 1, 2, 5) \leftrightarrow (8, 4, 2, 5)$ p'_1: $(10, 1, 2, 6) \leftrightarrow (10, 5, 2, 6)$
p'_2: $(12, 1, 2, 7) \leftrightarrow (12, 6, 2, 7)$ p'_3: $(16, 2, 2, 9) \leftrightarrow (16, 8, 2, 9)$
p'_4: $(16, 2, 4, 5) \leftrightarrow (16, 8, 4, 5)$ p'_5: $(24, 1, 4, 7) \leftrightarrow (24, 6, 4, 7)$
p'_6: $(24, 2, 2, 13) \leftrightarrow (24, 12, 2, 13)$
p'_7: $(28, 2, 2, 15) \leftrightarrow (28, 14, 2, 15)$

Results for Opt-S-F and Opt-R-F. Figure 11 plots the costs. We summarize the following observations.

- From Figures 11(a) and 11(c), Opt-S-F always has the minimum upcoding cost (zero), while Opt-R-F is designed with the minimum repair cost.
- Opt-S-F and Opt-R-F outperform Flat in terms of the scaling and repair costs stably.
- For the cases where $b = g$ (e.g., p_0-p_2), we can only put $\theta = 1$ local parity block and its local data set in one cluster, so we cannot utilize partial encoding of the local parity blocks in each cluster for upcoding. Thus, the upcoding cost of Opt-R-F equals that of Flat. For other cases, we put more than one local parity block in one cluster, so partial encoding of the local parity blocks brings benefit to the upcoding cost. For example, for scaling operation p_4, the upcoding cost of Flat is 10 while that of Opt-R-F is four.
- In Figure 11(b), Opt-S-F has less downcoding cost than Opt-R-F for scaling operation p_5, and the reason is that Opt-S-F can exploit the local parity block of the compact LRC to calculate one remaining local parity block, therefore saving one network transfer in each upcoding unit (Section 4.4). However, for p_2, p_4, and p_6, Opt-R-F can regenerate each local parity block within each cluster locally, and thus has less downcoding cost (zero) than Opt-S-F.
- From Figure 11(d), the repair cost of the compact LRC of Opt-S-F equals that of Opt-R-F (because the layout of the compact LRC of Opt-S-F is the same as that of Opt-R-F (Figure 4)), and both are much smaller than that of Flat.
- When g increases, the scaling and repair costs of both Opt-S-F and Opt-R-F decrease, while these costs of Flat keep constant. Hence, the improvements of Opt-S-F and Opt-R-F over Flat become greater with larger g. For example, Opt-S-F reduces the repair cost of the fast LRC of Flat by 67% for scaling operation p_2 (i.e., $g = 2$), while the reduction becomes 83% for scaling operation p_3 (i.e., $g = 4$).

Results for Opt-S-C and Opt-R-C. Figure 12 shows the costs. We summarize the observations as follows.

- From Figures 12(b) and 12(d), Opt-S-C has the minimum downcoding cost (zero), while Opt-R-C has the minimum repair cost of the compact LRC.
- When $\frac{\delta}{\theta} = \lceil \frac{\delta \times b + 1}{g + 1} \rceil$ (e.g., p'_4), in Opt-S-C, we minimize the downcoding cost and the repair cost simultaneously.
- In Figure 12(a), Opt-S-C has the same upcoding cost as Flat for the cases where $b = g$ (e.g., p'_0) as we cannot exploit partial encoding during upcoding, while Opt-R-C has larger upcoding cost as it requires additional data

block movements. For the cases where $b < g$ (e.g., $p_4^{'}$), we can enable partial encoding, and so the upcoding cost of Opt-S-C/Opt-R-C is smaller than that of Flat.

- In Figure 12(c), Opt-S-C has the same repair cost of the fast LRC as Opt-R-C because the layout of the fast LRC is Opt-R-F, where the repair cost of the fast LRC is zero.
- Opt-S-C and Opt-R-C have greater improvements over Flat with larger g. For example, for scaling operation $p_3^{'}$ (i.e., g = 2), Opt-R-C reduces the repair cost of the compact LRC of Flat by 75%, while for scaling operation $p_4^{'}$ (i.e., g = 4), the reduction becomes 87.5%.

5.2 Testbed Experiments

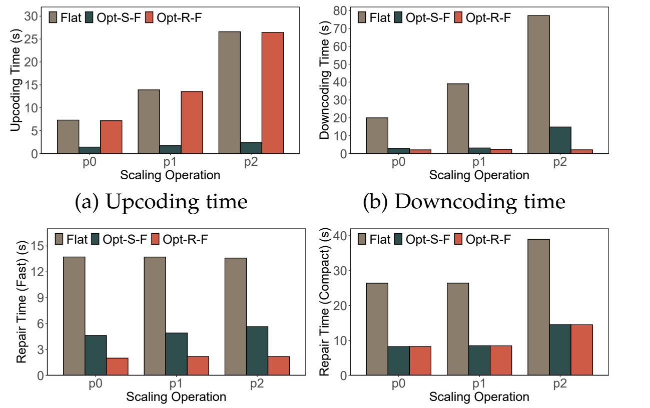
We implement Opt-S-F, Opt-R-F, Opt-S-C, Opt-R-C, and Flat (under both the fast LRC and the compact LRC) in a distributed storage system prototype, and conduct testbed experiments to understand their scaling and repair performance. In our prototype, repair and scaling are both implemented as two phase processes, where we first aggregate relevant data within a cluster, and then send the aggregated data across cluster to the destination cluster. Our prototype is written in C++ and implemented with a Coordinator (CN) and multiple Datanodes (DN). The CN sends commands, while the DNs receive commands and execute the actual data read, write, and transfer independently and in parallel, and finally reply acks to the CN.

Setup. Our testbed comprises 22 physical nodes, each of which runs Ubuntu 16.04.5 LTS with a quad-core 3.40 GHz Intel Core i5-3570, 16 GB RAM, and a Seagate ST1000DM003 7200 RPM 1 TB SATA hard disk. Each node achieves 170 MBps of disk read bandwidth and 130 MBps of write bandwidth, and 1 Gbps of network bandwidth (measured by iperf). We deploy the CN in one node, and the DNs on 20 nodes. We configure the DNs into different clusters according to the scaling operations and the placement strategy. We also configure a dedicated node to act as a network core, such that any cross-cluster traffic must traverse the network core. We use the Wonder Shaper tool [1] to control the outgoing bandwidth of the network core.

Methodology. We assume the following default configurations. We adopt the scaling operations p_2 (i.e., $(12, 6, 2, 7)$ to $(12, 2, 2, 7)$) for Opt-S-F and Opt-R-F, and $p_0^{'}$ (i.e., $(8, 1, 2, 5)$ to $(8, 4, 2, 5)$) for Opt-S-C and Opt-R-C. We configure the block size as 64 MB, the packet size as 1 MB (packet is the unit for network transfer), and the cross-cluster bandwidth as 100 Mbps (such that the ratio of inner-cluster bandwidth to cross-cluster bandwidth is 10:1). We may vary some of the settings in our experiments. We measure the repair time per block and the scaling time per stripe. The results of each experiment are averaged over five runs.

Experiment 1.1 (Performance under different scaling operations for Opt-S-F and Opt-R-F). We first evaluate the performance under different scaling operations. We consider three sets of scaling operations, i.e., p_0, p_1, and p_2. We fix the block size as 64 MB and the cross-cluster bandwidth as 100 Mbps, and then compare the scaling time and the repair time. Figure 13 shows the results.

According to our analysis, the upcoding cost of Opt-S-F is zero, while that of Flat and Opt-R-F increases as $l - l'$



(a) Upcoding time (b) Downcoding time

(c) Repair time (fast) (d) Repair time (compact)

Figure 13. Experiment 1.1: Scaling and repair time for Opt-S-F & Opt-R-F.

increases. From Figure 13(a), the experimental results comply with the theoretics. As $l - l'$ grows, the upcoding time of Opt-S-F keeps fairly stable while that of Flat and Opt-R-F increases. Thus, Opt-S-F will achieve more gain for larger $l - l'$. Overall, Opt-S-F reduces the upcoding time of Flat and Opt-R-F by 80.6% and 80.2%, 87.6% and 87.2%, and 91.0% and 91.0%, for p_0, p_1, and p_2, respectively. The upcoding time of Opt-R-F is similar to that of Flat, which confirms to the numerical results.

The repair cost of Opt-R-F is zero while that of Flat keeps constant as b keeps unchanged. The repair cost of Opt-S-F (i.e., $1 - \frac{\theta}{\delta}$) grows as $\delta = \frac{l}{l'}$ increases. From Figure 13(c), the repair time of Opt-R-F and Flat keeps stable while that of Opt-S-F grows slightly. However, Opt-R-F always has the smallest repair time. Overall, Opt-R-F reduces the repair time of Flat and Opt-S-F by 85.5% and 56.8%, 84.2% and 55.8%, and 84.0% and 61.5%, for p_0, p_1, and p_2, respectively.

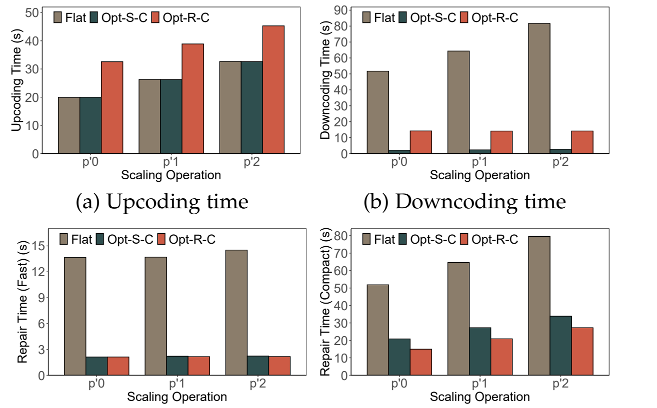
From Figure 13(b), both Opt-S-F and Opt-R-F show better downcoding time performance compared to Flat. For example, for p_2, Opt-S-F and Opt-R-F reduce the downcoding time of Flat by 80.7% and 97.3%, respectively.

Finally, from Figure 13(d), the repair time of the compact LRC of Opt-S-F is almost identical to that of Opt-R-F, and both show improvements over Flat. For example, for scaling operation p_2, Opt-S-F and Opt-R-F reduce the repair time of Flat by 62.8% and 62.8%, respectively.

Experiment 1.2 (Performance under different scaling operations for Opt-S-C and Opt-R-C). We consider $p_0^{'}$, $p_1^{'}$, and $p_2^{'}$ while other configurations are set as default. Figure 14 shows the scaling and repair time.

According to the numerical analysis, the downcoding cost of Flat increases as k increases, while that of Opt-S-C and Opt-R-C keeps stable. In Figure 14(b), the experimental results follow the theoretics. Overall, Opt-S-C has the smallest downcoding time, and it reduces the downcoding time of Flat and Opt-R-C by 95.9% and 85.1%, 96.4% and 83.6%, and 96.7% and 80.9%, for $p_0^{'}$, $p_1^{'}$, and $p_2^{'}$, respectively.

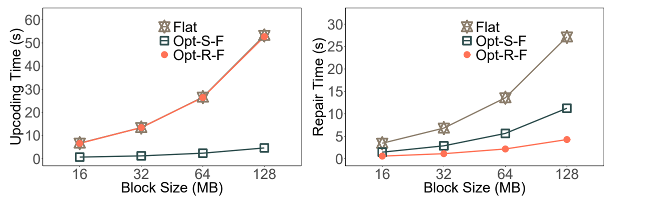
As $b^{'}$ increases, the repair cost of the compact LRC of Flat increases. Also, the repair costs of Opt-S-C and Opt-R-C increase as δ increases. From Figure 14(d), the repair time of the compact LRC increases gradually. Opt-R-C always has the smallest repair time and reduces the repair time of Flat and Opt-S-C by 71.2% and 28.3%, 67.7% and 23.3%, 65.7% and 19.5%, for $p_0^{'}$, $p_1^{'}$, and $p_2^{'}$, respectively.



(a) Upcoding time (b) Downcoding time

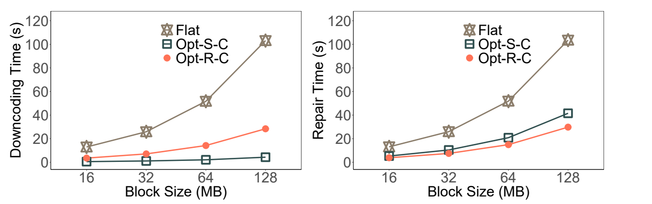
(c) Repair time (fast) (d) Repair time (compact)

Figure 14. Experiment 1.2: Scaling and repair time for Opt-S-C & Opt-R-C.



(a) Upcoding time (b) Repair time (fast)

Figure 15. Experiment 2.1: Impact of block size for Opt-S-F & Opt-R-F.



(a) Downcoding time (b) Repair time (compact)

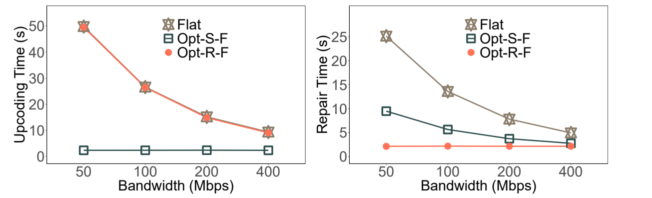
Figure 16. Experiment 2.2: Impact of block size for Opt-S-C & Opt-R-C.

From Figure 14(a), Opt-S-C has similar upcoding time to Flat while Opt-R-C has larger upcoding time, which confirms to the numerical results.

From Figure 14(c), the repair time of the fast LRC of Opt-S-C is similar to that of Opt-R-C and both are much smaller than that of Flat. For example, for scaling operation $p_0^{'}$, Opt-S-C and Opt-R-C reduce the repair time of the fast LRC of Flat by 84.5% and 84.5%, respectively.

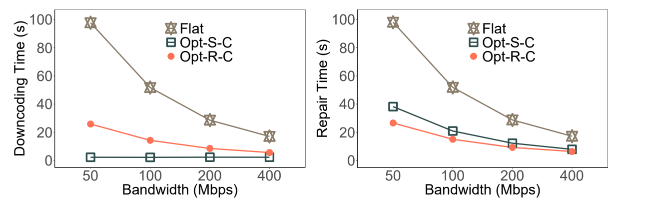
Experiment 2.1 (Impact of block size for Opt-S-F and Opt-R-F). We now evaluate the impact of the block size, varied from 16 MB to 128 MB. We test the default scaling operation (i.e., p_2) and fix the cross-cluster bandwidth as 100 Mbps. We only show the results for the upcoding time and the repair time of the fast LRC in the following experiments for Opt-S-F and Opt-R-F. Figure 15 shows the results. We can see that the upcoding time and the repair time increases with a larger block size, and Opt-S-F and Opt-R-F constantly outperform Flat. For example, Opt-S-F reduces the upcoding time of Flat from 89.6%-91.3%, and Opt-R-F reduces the repair time of Flat from 83.2%-84.3%, across all block sizes.

Experiment 2.2 (Impact of block size for Opt-S-C and Opt-R-C). We adopt the default scaling operation (i.e., $p_0^{'}$) and also test the impact of the block size for Opt-S-C and Opt-R-C. We show the results for the downcoding time and the repair time of the compact LRC in the following experiments for Opt-S-C and Opt-R-C. Figure 16 shows the results. We can see that Opt-S-C and Opt-R-C stably outperform Flat. Overall, Opt-S-C reduces the downcoding time of Flat by 95.6%-95.9%, while Opt-R-C reduces the repair time of Flat by 70.7%-71.2%, across all block sizes.



(a) Upcoding time (b) Repair time (fast)

Figure 17. Experiment 3.1: Impact of bandwidth for Opt-S-F & Opt-R-F.



(a) Downcoding time (b) Repair time (compact)

Figure 18. Experiment 3.2: Impact of bandwidth for Opt-S-C & Opt-R-C.

Experiment 3.1 (Impact of bandwidth for Opt-S-F and Opt-R-F). We now study the impact of cross-cluster bandwidth. Here, we adopt p_2 and fix the block size as 64 MB, and then vary the cross-cluster bandwidth from 50 Mbps to 400 Mbps (the ratio of inner-cluster bandwidth to cross-cluster bandwidth is 20:1-2.5:1). Figure 17 shows the results. As expected, the upcoding and repair time decreases with larger bandwidth. Besides, in Figure 17(a), Opt-S-F reduces the upcoding time of Flat by 95.2%, 91.0%, 84.1%, and 74.7% when the bandwidth is 50 Mbps, 100 Mbps, 200 Mbps, and 400 Mbps, respectively. In Figure 17(b), Opt-R-F reduces the repair time of Flat by 91.5%, 84.0%, 72.6%, and 56.6% when the bandwidth changes. These indicate that the improvements of Opt-S-F and Opt-R-F over Flat are greater with more scarce cross-cluster bandwidth.

Experiment 3.2 (Impact of bandwidth for Opt-S-C and Opt-R-C). We adopt $p_0^{'}$ and test the impact of bandwidth for Opt-S-C and Opt-R-C. Figure 18 shows the results. Overall, Opt-S-C reduces the downcoding time of Flat by 97.8%, 95.9%, 92.2%, and 87.0%, while Opt-R-C reduces the repair time of Flat by 73.0%, 71.2%, 68.1%, and 63.3%, for the four bandwidth configurations.

6 RELATED WORK

There has been extensive work on the repair performance of LRC in the literature. Theoretical studies on LRC (e.g., [8], [24]–[26]) focus on the relationship between the optimal minimum Hamming distance and the theoretical repair cost, and design explicit LRC constructions. LRC is also implemented and evaluated in Azure [11] and Facebook [22]. Kolosov *et al.* [14] study the trade-offs of different LRC constructions between storage overheads and repair costs. In contrast, our work mainly focuses on the trade-off between the repair and scaling costs in clustered storage systems.

Several studies address the scaling problem on the change of erasure coding configurations. Some studies (e.g., [4], [12],

[31], [33], [35], [36]) explore efficient data redistribution when a storage cluster adds new storage nodes, while others (e.g., [6], [16], [29], [30]) address the transition from replication to erasure coding. HeART [13] suggests the change of redundancy settings to balance between storage overhead and reliability. To adapt to workload changes, HACFS [34] proposes efficient transitions between two erasure codes to balance between storage overhead and access performance. Our work is motivated by the scenario in HACFS, and presents a formal repair-scaling trade-off analysis.

A number of studies consider the deployment of erasure coding in clustered storage that spans multiple geographic regions [3], [21]. In particular, some studies focus on minimizing the cross-cluster bandwidth during repair [9], [10], [17], [19], [23]. Our work is the first one that addresses scaling performance in clustered storage.

7 Conclusion

We investigate the optimal trade-off between the repair and scaling performance of LRC in clustered storage systems. Specifically, we analyze both the trade-off between the repair of the fast LRC and upcoding, and the trade-off between the repair of the compact LRC and downcoding. We design placement strategies that operate along the optimal repair-scaling trade-off curves subject to the single-cluster fault tolerance constraint. Both numerical studies and testbed experiments validate the efficiency of our placement strategies. The source code of our implementation is available at **http://adslab.cse.cuhk.edu.hk/software/lrctradeoff**.

Acknowledgements: This work is supported by NSFC (61832011, 62072381), CCF-Tencent Open Fund WeBank Special Fund, Zhejiang Lab (2021KF0AB01), the Fundamental Research Funds for the Central Universities (WK2150110022).

Appendix

Proof of Theorem 1. Suppose that the upcoding cost of a placement is u, and the upcoding cost induced by the i-th $(0 \leq i \leq l' - 1)$ upcoding unit is u_i ($\sum_{i=0}^{l'-1} u_i = u$). For an upcoding unit, there must be a *core cluster* that will collect and encode the δ local parity blocks. There must also be some different clusters that hold local parity blocks. We call each such cluster a *remote cluster*. Our proof is as follows.

(i) We first prove that the number of remote clusters for the i-th upcoding unit is exactly u_i. During upcoding, the core cluster will retrieve an XOR sum of the local parity blocks in each remote cluster. If the number of remote clusters is not u_i, then the upcoding cost will be different from u_i.

(ii) We next prove that a core cluster also holds local parity blocks. Otherwise, we choose one remote cluster as the new core cluster. During upcoding, the local parity blocks in the new core cluster can be retrieved locally, so the upcoding cost for the i-th upcoding unit will be $u_i - 1$ rather than u_i.

(iii) We then prove that we can collocate (at most) θ local data sets into a cluster that holds local parity blocks if the corresponding local parity blocks of these local data sets are also in this cluster. Suppose that a cluster holds r local parity blocks, and we can collocate (at most) w local data sets whose local parity blocks are included in the r ones. The number of blocks (i.e., $w \times b + r$), which span r local groups, cannot exceed $g + r$ (Lemma 1). Thus, $w = \lfloor \frac{g}{b} \rfloor = \theta$, and the cost for repairing any data block in these θ local data sets is zero.

(iv) We now show that the core cluster and remote clusters of one upcoding unit should be different from those of another upcoding unit so as to minimize the repair cost. If cluster A of one upcoding unit is the same as cluster B of another upcoding unit, then according to (iii), we can put (at most) θ local data sets into A (or B). If A is different from B, then we can put θ local data sets into A and another θ local data sets into B. Hence, if the core cluster and remote clusters for each upcoding unit are different, then we can collocate more local data sets with their local parity blocks, and so the repair cost of more data blocks is zero.

(v) We now prove that the maximum number of data blocks whose repair cost is zero is $(l' + u) \times \theta \times b$. From (iii), we can put (at most) θ local data sets into a cluster that holds local parity blocks, so as to make the cost for repairing any data block in these θ local data sets as zero. From (i), (ii), and (iv), we have (at most) l' core clusters and u remote clusters that hold local parity blocks. Thus, the maximum number of data blocks whose repair cost is zero is $(l' + u) \times \theta \times b$.

(vi) Finally, we prove that we cannot have a placement whose repair cost is less than $1 - \frac{\theta}{\delta} - \frac{u \times \theta}{l}$ subject to the upcoding cost of u. Otherwise, since the repair cost is less than one, there must be some data blocks whose repair cost is zero. We assume that the number of data blocks whose repair cost is zero is α, so the number of data blocks whose repair cost is at least one is $k - \alpha$. Therefore, $\frac{k-\alpha}{k} < 1 - \frac{\theta}{\delta} - \frac{u \times \theta}{l}$, implying $\alpha > (l' + u) \times \theta \times b$. However, this is conflicting with our proof in (v). Thus, we complete our proof.

Proof of Theorem 2. Suppose that the downcoding cost of a placement is d, and we organize our proof as follows.

(i) We first show that the number of moved local data sets (based on Opt-S-C) is at most $\frac{d}{b}$. Otherwise, since we must relocate the moved local data sets back for downcoding, the downcoding cost will be larger than $\frac{d}{b} \times b = d$.

(ii) As we move some local data sets to different clusters, the number of clusters some local groups of the compact LRC span will be reduced and so the repair cost of the data blocks in the local groups will be decreased accordingly. We assume that the number of local groups where the data blocks have reduced repair cost is x, so the number of local groups where the data blocks have unchanged repair cost is $l' - x$.

(iii) Based on (ii), we further assume that in the first to the $(x-2)$-th local group, the repair cost of the data blocks reduces by y (i.e., the number of clusters the local group spans reduces by y), and in the $(x-1)$-th local group, the repair cost of the data blocks reduces by z (i.e., the number of clusters the local group spans reduces by z).

(iv) According to (ii) and (iii), the repair cost of the placement is calculated as $((\frac{\delta}{\theta} - 1 - y) \times (x-1) + (\frac{\delta}{\theta} - 1 - z) + (\frac{\delta}{\theta} - 1) \times (l' - x))/l'$.

(v) Based on (iii), we can calculate that the number of moved local data sets is $((x-1) \times y + z) \times \theta$. Based on (i), we can show that $((x-1) \times y + z) \times \theta \leq \frac{d}{b}$.

(vi) Assuming that there is a placement with repair cost smaller than $\frac{\delta}{\theta} - 1 - \frac{d}{\theta \times b \times l'}$ subject to the downcoding cost of d, then the repair cost value in (iv) is smaller than $\frac{\delta}{\theta} - 1 - \frac{d}{\theta \times b \times l'}$, implying that $((x-1) \times y + z) \times \theta > \frac{d}{b}$, which is conflicting with our proof in (v). Thus, our proof completes.

REFERENCES

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

[2] 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.

[3] 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.

[4] L. Cheng, Y. Hu, and P. P. Lee. Coupling decentralized key-value stores with erasure coding. In *Proc. of ACM SoCC*, 2019.

[5] M. Chowdhury, S. Kandula, and I. Stoica. Leveraging endpoint flexibility in data-intensive clusters. In *Proc. of ACM SIGCOMM*, 2013.

[6] B. Fan, W. Tantisiriroj, L. Xiao, and G. Gibson. DiskReduce: RAID for data-intensive scalable computing. In *Proc. of ACM PDSW*, 2009.

[7] 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.

[8] P. Gopalan, C. Huang, H. Simitci, and S. Yekhanin. On the locality of codeword symbols. *IEEE Trans. on Information Theory*, 58(11):6925–6934, 2012.

[9] H. Hou, P. P. Lee, K. W. Shum, and Y. Hu. Rack-aware regenerating codes for data centers. *IEEE Trans. on Information Theory*, 65(8):4730–4745, Aug 2019.

[10] Y. Hu, X. Li, M. Zhang, P. P. 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.

[11] 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.

[12] J. Huang, X. Liang, X. Qin, P. Xie, and C. Xie. Scale-RS: An efficient scaling scheme for RS-coded storage clusters. *IEEE Trans. on Parallel and Distributed Systems*, 26(6):1704–1717, 2014.

[13] S. Kadekodi, K. Rashmi, and G. R. Ganger. Cluster storage systems gotta have HeART: Improving storage efficiency by exploiting disk-reliability heterogeneity. In *Proc. of USENIX FAST*, 2019.

[14] 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.

[15] J. Li, B. Li, and B. Li. Demand-Aware erasure coding for distributed storage systems. *IEEE Trans. on Cloud Computing (TCC)*, 2018.

[16] R. Li, Y. Hu, and P. P. Lee. Enabling efficient and reliable transition from replication to erasure coding for clustered file systems. *IEEE Trans. on Parallel and Distributed Systems*, 28(9):2500–2513, 2017.

[17] X. Li, R. Li, P. P. Lee, and Y. Hu. OpenEC: Toward unified and configurable erasure coding management in distributed storage systems. In *Proc. of USENIX FAST*, 2019.

[18] 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.

[19] 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.

[20] 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.

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

[22] 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.

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

[24] N. Silberstein, A. S. Rawat, O. O. Koyluoglu, and S. Vishwanath. Optimal Locally Repairable Codes via Rank-Metric Codes. In *Proc. of IEEE International Symposium on Information Theory*, 2013.

[25] I. Tamo and A. Barg. A family of optimal Locally Recoverable Codes. *IEEE Trans. on Information Theory*, 60(8):4661–4676, 2014.

[26] I. Tamo, D. S. Papailiopoulos, and A. G. Dimakis. Optimal Locally Repairable Codes and connections to Matroid theory. *IEEE Trans. on Information Theory*, 62(12):6661–6671, 2016.

[27] 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.

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

[29] S. Wei, Y. Li, Y. Xu, and S. Wu. DSC: Dynamic stripe construction for synchronous encoding in clustered file system. In *Proc. of IEEE INFOCOM*, 2017.

[30] J. Wilkes, R. Golding, C. Staelin, and T. Sullivan. The HP AutoRAID hierarchical storage system. *ACM Trans. on Computer Systems*, 14(1):108–136, 1996.

[31] C. Wu and X. He. GSR: A global stripe-based redistribution approach to accelerate RAID-5 scaling. In *Proc. of IEEE ICPP*, 2012.

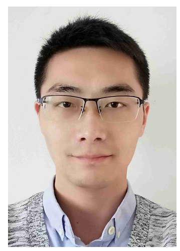
[32] S. Wu, Z. Shen, and P. P. Lee. On the optimal repair-scaling trade-off in Locally Repairable Codes. In *Proc. of IEEE INFOCOM*, 2020.

[33] S. Wu, Y. Xu, Y. Li, and Z. Yang. I/O-Efficient scaling schemes for distributed storage systems with CRS codes. *IEEE Trans. on Parallel and Distributed Systems*, 27(9):2639–2652, 2016.

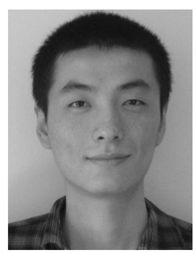
[34] M. Xia, M. Saxena, M. Blaum, and D. A. Pease. A tale of two erasure codes in HDFS. In *Proc. of USENIX FAST*, 2015.

[35] X. Zhang, Y. Hu, P. P. Lee, and P. Zhou. Toward optimal storage scaling via network coding: From theory to practice. In *Proc. of IEEE INFOCOM*, 2018.

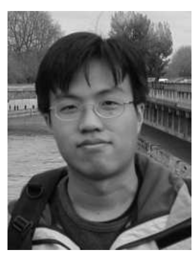
[36] W. Zheng and G. Zhang. FastScale: Accelerate RAID scaling by minimizing data migration. In *Proc. of USENIX FAST*, 2011.



Si Wu received the B.Eng. and Ph.D. degrees in Computer Science from University of Science and Technology of China in 2011 and 2016, respectively. He is now an associate researcher in School of Computer Science and Technology, University of Science and Technology of China. His research interests include storage reliability and distributed storage.



Zhirong Shen received the B.Eng. degree from University of Electronic Science and Technology of China in 2010, and the Ph.D. degree in Computer Science from Tsinghua University in 2016. He is now an associate professor at Xiamen University. His research interests include storage reliability and storage security. He is a member of the IEEE.



Patrick P. C. Lee received the B.Eng. degree (first-class honors) in Information Engineering from the Chinese University of Hong Kong in 2001, the M.Phil. degree in Computer Science and Engineering from the Chinese University of Hong Kong in 2003, and the Ph.D. degree in Computer Science from Columbia University in 2008. He is now an Associate Professor of the Department of Computer Science and Engineering at the Chinese University of Hong Kong. His research interests are in various applied/systems topics including storage systems, distributed systems and networks, operating systems, dependability, and security.

Yinlong Xu received the Bachelor's degree in mathematics from Peking University in 1983, and the master and PhD degrees in computer science from University of Science and Technology of China (USTC) in 1989 and 2004, respectively. He is currently a professor with the School of Computer Science and Technology at USTC. Currently, he is leading a group of research students in doing some networking and high performance computing research. His research interests include network coding, wireless network, combinatorial optimization, design and analysis of parallel algorithm, parallel programming tools, etc. He received the Excellent PhD Advisor Award of Chinese Academy of Sciences in 2006.