2023 IEEE International Parallel and Distributed Processing Symposium (IPDPS) | 979-8-3503-3766-2/23/$31.00 ©2023 IEEE | DOI: 10.1109/IPDPS54959.2023.00036

Boosting Multi-Block Repair in Cloud Storage Systems with Wide-Stripe Erasure Coding

Qi Yu†, Lin Wang†, Yuchong Hu†, Yumeng Xu†, Dan Feng†, Jie Fu§, Xia Zhu§, Zhen Yao§, Wenjia Wei§
†*Huazhong University of Science and Technology*
§*Huawei Technologies Co., Ltd., China*
{yuqi2021, wanglin2021, yuchonghu, daydream, dfeng}@hust.edu.cn
{fujie, zhuxia1, yaozhen9, weiwenjia}@huawei.com

***Abstract*—Cloud storage systems have commonly used erasure coding that encodes data in stripes of blocks as a low-cost redundancy method for data reliability. Relative to traditional erasure coding, wide-stripe erasure coding that increases the stripe size has been recently proposed and explored to achieve lower redundancy. We observe that wide-stripe erasure coding makes multi-block failures occur much more frequently than traditional erasure coding in cloud storage systems.**

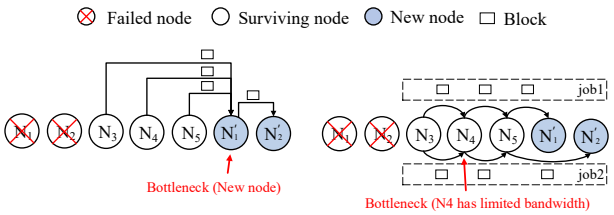
However, how to efficiently repair multiple blocks in wide-stripe erasure-coded storage systems remains unexplored. The conventional multi-block repair method sends available blocks from surviving nodes to one single new node to repair all failed blocks in a *centralized* way, which may cause the new node to be the bottleneck; recent multi-block repair methods follow pipelined single-block repair methods and the former are simply built on the latter in an *independent* way, which may cause the surviving nodes with limited bandwidth to be bottlenecks.

In this paper, we first analyze the effects of both centralized and independent ways on the multi-block repair and then propose HMBR, a hybrid multi-block repair mechanism that combines centralized and independent multi-block repairs to tradeoff the bandwidth bottlenecks caused by the new and surviving nodes, thus optimizing the multi-block repair performance. We further extend HMBR for hierarchical network topology and multi-node failures. We prototype HMBR and show via Amazon EC2 that the repair time of a multi-block failure can be reduced by up to 64.8% over state-of-the-art schemes.

I. Introduction

To maintain data reliability, erasure coding has been widely deployed as a low-cost redundancy scheme for cloud storage [1]. One class of the most popular erasure codes is Reed-Solomon (RS) codes with two configurable parameters k and m. Briefly, (k, m) RS codes encode k fixed-size *data blocks* into m *parity blocks*, where the $k+m$ blocks form a *stripe* satisfying that any k blocks of the stripe can reconstruct any block (see §II-A for details). Compared to replication that needs a redundancy of $m+1$ to tolerate m failures, (k, m) RS codes have a much less redundancy which is only $\frac{k+m}{k}$.

Although traditional erasure coding has reduced redundancy, cloud storage providers are still interested in further reducing redundancy, since even a small fraction of redundancy reduction (e.g., 14% [2]) can still save millions of dollars in clouds. Recently, the concept of *wide-stripe erasure coding* has been proposed in industry [3] and explored in academia [2], [4], [5], [6], which refers to the stripes that have a very large k (e.g., $k = 64$ [2]) such that the redundancy (i.e., $\frac{k+m}{k}$) can be extensively reduced. Thus, wide-stripe erasure coding provides an opportunity to achieve extremely low redundancy for cloud storage to maintain data reliability at ultra-low cost.



(a) Centralized repair method (b) Independent repair method

Figure 1. Illustration of two existing multi-block repair methods to repair two blocks of a $(3, 2)$ RS coded stripe.

Recent studies on wide-stripe erasure coding [2], [6] mainly aim to repair single-block failure efficiently, as any single failed block is reconstructed from k available blocks and thus wide stripe's large k leads to a high single-block repair penalty. In this paper, we observe that in cloud storage, wide-stripe erasure coding makes multi-block failures occur much more frequently than traditional erasure coding, because the former may have more failed blocks within a stripe than the latter at the same time (see II-B for details).

However, it is challenging for erasure-coded storage systems to deal with multi-block repair *efficiently* for two reasons: (1) To repair f failed blocks of a stripe, the conventional multi-block repair [7] sends k blocks from k surviving nodes to one single new node who repairs the f failed blocks in a *centralized* way. As illustrated in Figure 1(a), the new node N_1' that downloads three blocks during repair becomes bottleneck of the multi-block repair performance, as it is much more congested than other nodes each of which only transfers one block (see §II-C for details). (2) Recently, many new multi-block repair schemes [8], [9], [10] have been proposed, which are simply based on state-of-the-art pipelined single-block repair schemes [8], [9], [10] in an *independent* way. As illustrated in Figure 1(b), a multi-block repair is divided into two single-block pipelined repair jobs, and thus each surviving node has to transfer two blocks, thus causing those surviving nodes who have limited bandwidth (e.g., N_4) to be bottlenecks

1530-2075/23/$31.00 ©2023 IEEE
DOI 10.1109/IPDPS54959.2023.00036

of the multi-block repair performance (see §II-D for details).

In this paper, we propose a hybrid multi-block repair mechanism HMBR, which optimally combines centralized and independent multi-block repair methods for wide-stripe erasure-coded cloud storage systems. When repairing multiple failed blocks of each stripe, HMBR splits each available block into two parts (called *sub-blocks*), and performs centralized and independent repairs on different sub-blocks, so as to tradeoff different bottlenecks caused by the new and surviving nodes. In addition, we further design atop HMBR efficient repair schemes for rack-aware environments and multi-node failures. Our contributions include:

- We model HMBR and show how to efficiently combine centralized and independent repair methods, and prove that HMBR can achieve optimality, meaning that it can minimize the repair transfer time for multi-block repair via tuning the ratio of two repair methods (§III).
- We design HMBR which realizes the optimal hybrid of centralized and independent multi-block repair methods. We extend HMBR to address hierarchical network topology via a rack-aware repair path that reduces the cross-rack bandwidth, and design an efficient multi-node repair method atop HMBR via scheduling different multi-block repairs of different stripes on all new nodes as evenly as possible (§IV).
- We implement HMBR built on HDFS with our prototype opensourced at: https://github.com/yuchonghu/HMBR. We evaluate HMBR via Amazon EC2 experiments. The results show that HMBR can reduce the multi-block repair transfer time by up to 64.8% compared to existing schemes, as well as confirm efficiency of our enhancements atop HMBR (§V).

II. Background and Motivation

We introduce the basics of erasure coding as well as wide stripes (§II-A), specify our observation of multi-block repair issue for wide stripes in cloud storage (§II-B), describe two existing multi-block repair methods (§II-C and §II-D), and motivate the idea of HMBR via examples (§II-E).

A. Erasure Coding

A rich body of studies (see §VI) have proposed various erasure codes, among which Reed-Solomon (RS) codes [11] are the most widely deployed one in cloud storage systems (e.g., HDFS [12], Ceph [13] and Swift [14]), so we focus on RS codes in this paper.

RS codes can be constructed by two parameters k and m, and a (k, m) RS code encodes k fixed-size (e.g., 64MiB [15]) *data blocks*, denoted by D_i ($1 \leq i \leq k$), into m *parity blocks*, denoted by P_j ($1 \leq j \leq m$). The $k+m$ blocks, which form a *stripe*, are distributed across $k+m$ different nodes to tolerate any m node failures. Mathematically, each parity block is computed from a linear combination of k data blocks based on the arithmetic of Galois Field GF(2^w) in w-bit word, i.e., $P_j = \sum_{i=1}^{k} \alpha_{i,j} D_i$, where $\alpha_{i,j}$ denotes the encoding coefficient generated from the Vandermonde matrix. Thus, if D_i fails, we can decode it from k available blocks:

$$D_i = \frac{1}{\alpha_{i,j}}(P_j + \sum_{t=1}^{i-1} \alpha_{t,j} D_j + \sum_{t=i+1}^{k} \alpha_{t,j} D_j). \quad (1)$$

Based on the above basics of erasure coding, there exist three properties of RS codes as follows:

- **Property 1: Maximum Distance Separable (MDS).** (k, m) RS codes achieve optimal storage, meaning that any k out of $k+m$ blocks of a stripe suffice to decode any block.
- **Property 2: Linearity of Single-block Repair [8].** Based on Equation (1), we see that the single-block repair operation can be performed via linear additions of k available blocks, satisfying that the repair operation can be divided into k repair sub-operations operated on k nodes in a pipeline.
- **Property 3: Fine-grained Repair [16].** Each block is performed in w-bit words; i.e., content with the same offset of each block of a stripe will be encoded or decoded together, so we can divide a block into *sub-blocks* (in words), and perform a repair on sub-blocks at a fine-grained level.

Property 1 enables conventional multi-block repair to save repair bandwidth via retrieving only k available blocks collectively to repair any number of failed blocks of a stripe (see §II-C); Property 2 enables current state-of-the-art pipelined single-block repair methods via performing additions (see §II-D); Property 3 enables our proposed mechanism HMBR that combines two existing multi-block repair methods via dividing each block into two sub-blocks (see §II-E).

Wide-Stripe Erasure Coding: Recent studies on erasure coding propose a new notion, called *wide stripes* [2]. Wide stripes mean that the parameter k of RS codes is set very large (e.g., $k = 64$ for cloud storage [2]), and thus suppress the fraction of parity blocks in a stripe to achieve extreme storage savings. For example, a $(6, 3)$ RS code is commonly used in practical production [15], which has a redundancy of $1.5\times$. Alternatively, VAST company considers a $(150, 4)$ wide-stripe code [3], so the redundancy can be reduced to $1.027\times$, which achieves near-optimal redundancy, i.e., approaching one.

B. Multi-block Repair Issue for Wide Stripes in Cloud Storage

Observation: We specify the observation that wide-stripe erasure coding makes multi-block failures occur much more frequently than traditional erasure coding in cloud storage. We verify this observation via simulations as follows:

Some studies [17] show that multiple nodes in cloud storage may fail simultaneously caused by cluster power outages, and after the power has been restarted, 0.5%-1% of all nodes will never come back and the data on those nodes will be lost permanently. We simulate this scenario and vary the parameters k and m to observe how often multi-block failures occur after a cloud storage's power outage when it comes to wide stripes.

We set (k, m) from $k = 6$ [12] to $k = 64$ [2], and set the number of nodes (denoted by N) from 500 to 5000 (practical cloud storage setting [17]). We denote the ratio of the stripes with multiple failed blocks to all the stripes with failed blocks

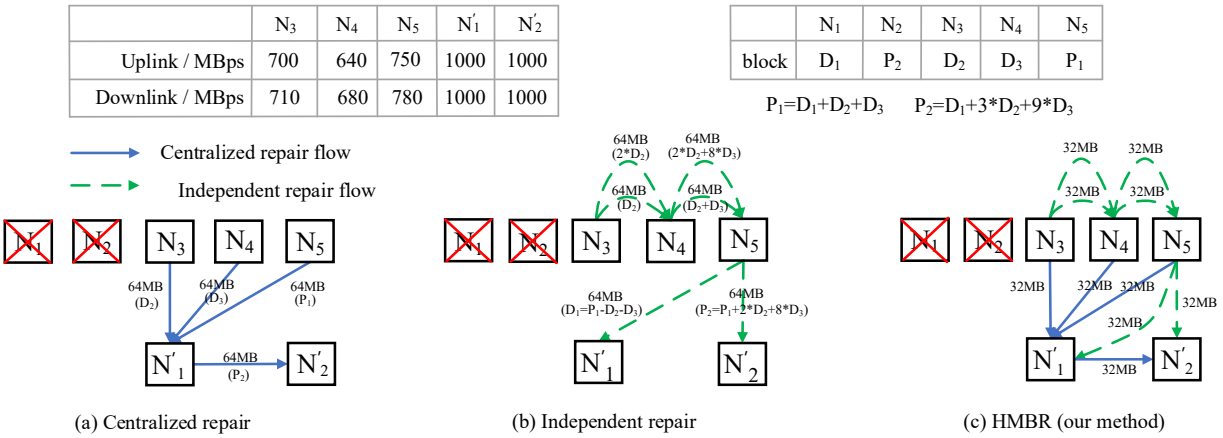


Figure 2. Motivating example for a $(3, 2)$ RS code with three data blocks D_1, D_2, D_3 and two party blocks P_1, P_2, where $P_1 = D_1 + D_2 + D_3$ and $P_2 = D_1 + 3D_2 + 9D_3$. Each block is size of 64MB. Two nodes N_1 and N_2 fail, so two blocks D_1 and P_2 are lost. Two new nodes N_1' and N_2' repair D_1 and P_2, respectively. We give each node's available bandwidth for repair, where the new nodes have sufficient bandwidth.

Table I
RESULTS OF SIMULATIONS

(k, m)	$R(N$=500)	$R(N$=1000)	$R(N$=2500)	$R(N$=5000)
(6, 3)	3.24%	3.57%	3.81%	3.92%
(9, 3)	4.46%	4.94%	5.20%	5.30%
(12, 4)	5.89%	6.80%	7.12%	7.21%
(64, 8)	28.16%	30.13%	30.80%	31.23%
(64, 16)	31.75%	32.93%	34.00%	34.36%
(64, 24)	34.15%	36.15%	36.86%	37.21%

by R and let each node store 1TiB of data. All stripes are distributed randomly across all nodes. We assume that 1% of all nodes fail at the same time and compute the R under different (k, m) and N. Table I shows the results of simulations.

Analysis: We can see that R increases with the (k, m) and becomes significant in wide-stripe settings, because stripes with a larger (k, m) will occupy more nodes and thus are more likely to incur multi-block failures. For example, when $k = 64$, R is around 30%, meaning that multi-block failures account for a non-negligible percent of all failed stripes. Therefore, it is critical to propose efficient multi-block repair methods in wide-stripe erasure-coded cloud storage systems. To this end, we will first introduce two existing multi-block repair methods in §II-C and §II-D, and propose our multi-block repair mechanism HMBR in §II-E.

C. Centralized Multi-block Repair

Based on Property 1, to repair f failed blocks of a stripe, the conventional multi-block repair method transmits k blocks from k surviving nodes to one single new node, which decodes the f failed blocks together, stores one of them, and distributes the other $f - 1$ blocks across the other $f - 1$ new nodes. As illustrated in Figure 2(a), the new node N_1' receives three blocks D_2, D_3 and P_1 to repair the lost chunks D_1 and P_2, stores D_1, and sends P_2 to the other new node N_2'.

Clearly, the single new node acts as a *center* between surviving nodes and the other new nodes, so it will be the most congested node and thus bottlenecks the total repair performance. In Figure 2(a), the new node N_1' is chosen as the center of the star, and clearly, its downlink is the most congested by downloading three blocks, thus dominating the repair transfer time, which can be computed by $t_1 = \frac{64MB \times 3}{1000MB/s} = 0.192s$, where 1000MB/s is the downlink bandwidth of N_1'.

D. Independent Multi-Block Repair

Based on Property 2, many efficient single-block repair schemes [8], [16], [9], [10] have been developed to decompose a single-block repair operation into pipelined sub-operations performed simultaneously, such that the repair bandwidth can be balanced among all nodes, so new nodes will not become the bottleneck (unlike centralized repair in §II-C).

Naturally, some new multi-block repair schemes are also proposed via performing pipelined single-block repair multiple times [8], [9], [10]. We note that these multi-block repair schemes are basically *independent* combinations of pipelined single-block repairs. The reason is that different failed blocks have different decoding coefficients (see §II-A). As illustrated in Figure 2(b), the failed block D_1 is decoded by $P_1 - D_2 - D_3$ while the failed block P_2 is decoded by $P_1 + 2D_2 + 8D_3$, which leads to two independent pipelined single-block repairs.

Unfortunately, for a multi-block repair that is divided into f independent single-block repairs, the independency makes each single-block repair uncooperative, which means the surviving nodes have to transmit $k \times f$ blocks since each of the f single-block repairs needs to transmit k blocks (Property 1). In Figure 2(b), each surviving node has to upload two blocks during pipelining, so N_4 with the slowest uplink becomes the bottleneck, dominating the repair transfer time, which can be computed by $t_2 = \frac{64MB \times 2}{640MB/s} = 0.20s$, where 640MB/s is the uplink bandwidth of N_4.

E. Motivation

As stated above, we observe that for multi-block repair, both centralized methods (bottlenecked by the central new node) and independent methods (bottlenecked by the slowest surviving node) will hinder the multi-block repair performance. This motivates us to take a hybrid of centralized and independent multi-block repairs to obtain a better tradeoff that mitigates the bottleneck among all nodes.

Main idea: Based on Property 3, we can divide each available block into two sub-blocks in a fine-grained level, such that all the upper sub-blocks use centralized repair, while all the lower sub-blocks use independent repair, which we call HMBR.

Figure 2(c) illustrates HMBR's idea. We divide each available block into two sub-blocks equally, each with the size of 32MB, and perform centralized and independent repairs on the upper and lower sub-blocks, respectively. Here, we re-consider the two bottleneck nodes in Figure 2(a) and (b): (1) the central new node N_1' needs to download four sub-blocks, including three from centralized repair and one from independent repair, so its repair transfer time $t_1 = \frac{32MB \times 4}{1000MB/s} = 0.128s$. (2) the node with the slowest uplink N_4 needs to upload three sub-blocks, including one from centralized repair and two from independent repair, so its repair transfer time $t_2 = \frac{32MB \times 3}{640MB/s} = 0.15s$. In this case, N_4 bottlenecks the multi-block repair transfer time (i.e., $0.15s$), which is faster than the centralized method (i.e., $0.192s$ in §II-C) and independent method (i.e., $0.20s$ in §II-D).

While the above example shows that HMBR can reduce the multi-block repair transfer time, how to divide each available block with the *optimal* proportion still needs analysis, which will be specified in §III.

III. Model and Analysis

We define the HMBR mechanism with its objective (§III-A), model HMBR by characterizing its repair transfer time (§III-B), and analyze how to optimize HMBR (§III-C).

A. Definition and Objective

Based on the main idea mentioned in §II-E, HMBR repairs f failed blocks by combining centralized repair (called CR for short) and independent repair (called IR for short) via dividing each available block into two sub-blocks. To specify the sub-blocks, we split a block of size B into multiple words, each of which has length of l_w (often set to 8 bytes), so a block has a total of $\frac{B}{l_w}$ words, and the sub-block is composed of multiple words.

The primary objective of HMBR is, as stated in §II-E, to find the optimal proportion of two sub-blocks within one block to minimize the repair transfer time. Note that we only consider the repair transfer time in this paper, because it often dominates the overall repair time [16], [8], which is also verified in Experiment 6 in §V.

Here, let p be the ratio of a block repaired by CR, so $1-p$ is the ratio of a block repaired by IR. In other words, $p \times \frac{B}{l_w}$ words will be repaired by CR and $(1-p) \times \frac{B}{l_w}$ will be repaired by IR. Let T_{CR} be the repair transfer time of CR and T_{IR} be the repair transfer time of IR, and the repair transfer time in HMBR is denoted by T. Clearly, HMBR can obtain the optimal value of p only when T is minimized. To minimize T, we need to first model T_{CR} and T_{IR} to obtain T (§III-B) and then analyze how to optimize T via tuning p (§III-C).

B. Characterizing Repair Transfer Time

We first obtain practical bandwidths between nodes in HMBR (§III-B1) and then use them to calculate T (§III-B2).

1) Practical Bandwidth: To analyze the practical bandwidth between nodes in HMBR, we consider the following three cases. Here, we let N_i be the i^{th} node where the i^{th} block of the stripe is located, and let U_i and D_i be the uplink bandwidth and downlink bandwidth of the i^{th} node, respectively. We let $bw_{i,j}$ indicate the practical bandwidth between N_i (sender) and N_j (receiver).

Case 1: single-to-single. We define that a single node transfers data to another node as a single-to-single case. The bandwidth between the two nodes depends on the minimum value of the uplink bandwidth of the sender and the downlink bandwidth of the receiver. For example, if N_i is the sender and N_j is the receiver. Then the bandwidth between N_i and N_j is $bw^1_{i,j} = \min(U_i, D_j)$.

Case 2: single-to-multiple. We define that a single node transfers data to multiple nodes simultaneously as a single-to-multiple case. Because the sender establishes multiple connections to other nodes at the same time, the uplink of the sender will be divided equally by the number of connections [18]. Therefore, the bandwidth between the sender and one of the receivers depends on the minimum value of the uplink bandwidth divided by the number of connections and the download uplink of the receiver. Specifically, if N_i sends data to r nodes, the practical bandwidth between N_i and the $j^{th}(1 \le j \le r)$ node of receivers is $bw^2_{i,j} = \min(\frac{U_i}{r}, D_j)$.

Case 3: multiple-to-single. We define that multiple nodes transfer data to a single node simultaneously as a multiple-to-single case. Similarly, the downlink bandwidth of the receiver will be divided equally by the number of connections. Specifically, if s nodes send data to N_j, the bandwidth between the $i^{th}(1 \le i \le s)$ node of the senders and N_j is $bw^3_{i,j} = \min(U_i, \frac{D_j}{s})$.

2) Repair Transfer Time: Based on the practical bandwidth between nodes, we first formulate the repair transfer time of CR and IR, and then obtain the repair transfer time T of HMBR.

Repair transfer time of CR: CR consists of two steps, as shown in Figure 2(a). First, k surviving nodes transfer k blocks to the central new node. Second, the central new node rebuilds f failed blocks and transfers them to other new nodes. The first step is a multiple-to-single case, and the repair transfer time is $T_1 = \frac{B}{\min_{1 \le i \le k} bw^3_{i,j}}$, where j is the index of a new node. The second step is a single-to-multiple case, and the repair transfer time is $T_2 = \frac{B}{\min_{1 \le l \le f-1} bw^2_{j,l}}$, where l is the index of the other

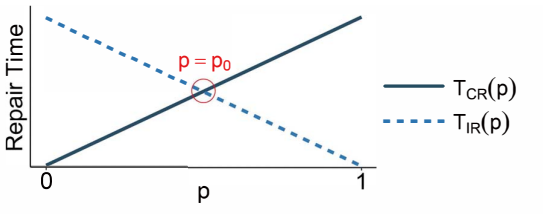


Figure 3. An example of Lemma 1. $T_{CR}(p)$ and $T_{IR}(p)$ definitely intersect when $p = p_0$.

new nodes. Then we can formulate the repair transfer time of CR as

$$T_{CR} = \frac{B}{\min_{1\le i\le k} bw_{i,j}^3} + \frac{B}{\min_{1\le l\le f-1} bw_{j,l}^2}. \quad (2)$$

Repair transfer time of IR: IR can be viewed as the combination of multiple single-to-single models. Since multiple failed blocks are repaired independently, each node has to send f blocks to other nodes (except the last node) and receive f blocks from other nodes (except the first node). Since the transfer process can be pipelined, the repair transfer time will be determined by the slowest link of the pipeline, so we can formulate the repair transfer time of IR as

$$T_{IR} = \frac{f \times B}{\min_{1\le i\le k+f-1, 1\le j\le k+f-1, i\ne j} bw_{i,j}^1} \quad (3)$$

where the i^{th} node and the j^{th} node are adjacent nodes on the pipeline.

Repair transfer time of HMBR: HMBR splits the multi-block repair job into two parts, which are performed by CR and IR respectively, via dividing each block ($\frac{B}{l_w}$ words) into two sub-blocks where the upper one ($p \times \frac{B}{l_w}$ words) and the lower one ($(1-p) \times \frac{B}{l_w}$ words). Based on Equation (2) and Equation (3), the repair time for the CR and IR in HMBR is

$$T_{CR}(p) = p \times T_{CR}, \quad T_{IR}(p) = (1-p) \times T_{IR}. \quad (4)$$

Since CR and IR are dominated by the central new node and the slowest surviving node, respectively (see §II-C and §II-D), they can be performed in parallel. Thus the repair transfer time can be formulated as

$$T(p) = \max(T_{CR}(p), T_{IR}(p)). \quad (5)$$

C. Optimality Analysis

Based on Equation (5), to minimize HMBR's repair transfer time $T(p)$, we first need to characterize two functions of $T_{CR}(p)$ and $T_{IR}(p)$, which is described in Lemma 1.

Lemma 1 *$\exists p_0 \in (0,1), T_{CR}(p_0) = T_{IR}(p_0)$. In other words, $T_{CR}(p)$ and $T_{IR}(p)$ definitely intersect when $p \in (0,1)$.*

Proof: Let $F(p) = T_{CR}(p) - T_{IR}(p)$. Then $F(0) = T_{CR}(0) - T_{IR}(0) = -T_{IR}(0) < 0$ and $F(1) = T_{CR}(1) - T_{IR}(1) = T_{CR}(1) > 0$. Obviously, $F(p)$ is continuous when $p \in (0,1)$. Therefore, based on the intermediate value theorem [19], there definitely exists one p_0 ($0 \le p_0 \le 1$) such that $F(p_0) = 0$ (i.e., $T_{CR}(p_0) = T_{IR}(p_0)$). □

Figure 3 shows an example of Lemma 1. Mathematically, $T_{CR}(p)$ and $T_{IR}(p)$ definitely intersect when $p \in (0,1)$. Therefore, we can further formulate T as

$$T(p) = \begin{cases} T_{CR}(p) & 0 \le p \le p_0 \\ T_{IR}(p) & p_0 < p \le 1 \end{cases} \quad (6)$$

Theorem 1. *$\forall p \in [0,1], T(p) \ge T(p_0)$. In other words, $T(p_0)$ is the minimum value of $T(p)$ when $p \in [0,1]$, where p_0 can be obtained via $T_{CR}(p_0) = T_{IR}(p_0)$.*

Proof: $T(p)$ increases when $p \in [0, p_0]$, and decreases when $p \in (p_0, 1]$. Therefore, $T(p_0)$ is minimum when $p = p_0$. □

Thus, Theorem 1 indicates that HMBR achieves the minimum repair transfer time when the ratio p is set to p_0, where $T_{CR}(p_0) = T_{IR}(p_0)$.

IV. Design

We design HMBR with the following design goals:

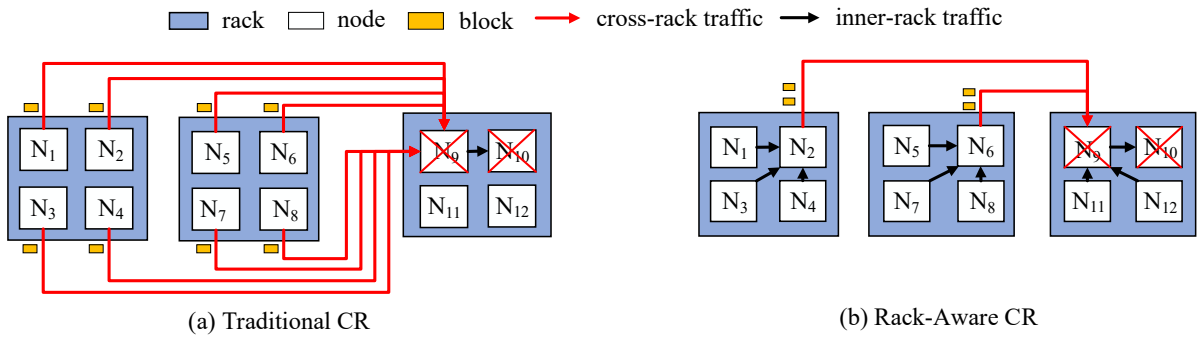
- **Minimum repair transfer time:** HMBR performs CR and IR on two sets of sub-blocks, divided from each available block with an optimal proportion based on Theorem 1, such that the repair transfer time is minimized (§IV-A).
- **Rack-aware multi-block repair:** HMBR is extended in hierarchical network topology in a way that trades inner-rack transfers for reducing cross-rack repair bandwidth, so as to improve repair performance (§IV-B).
- **Efficient multi-node repair:** HMBR is extended for efficient multi-node repair via scheduling different multi-block repairs of different stripes to reduce contention for shared link bandwidth (§IV-C).

A. HMBR

Design Idea: HMBR leverages Theorem 1 to obtain the optimal value of p_0, divides each available block into two sub-blocks based on p_0, and performs CR and IR on the corresponding sub-blocks. Theorem 1 ensures optimality.

Design Details: Consider a storage system that stores data in the fixed-size block and has parameters k, m and f. Suppose that we have obtained the uplink and downlink bandwidth of all nodes. Based on the design idea, HMBR takes the following four steps to repair f failed blocks:

- **Step 1:** HMBR obtains two functions $T_{CR}(p)$ and $T_{IR}(p)$ based on Equations (2), (3) and (4), and lets $T_{CR}(p) = T_{LR}(p)$ to solve the optimal proportion p_0.
- **Step 2:** HMBR splits a block into an upper sub-block and a lower sub-block where the upper one accounts for p_0 of the block while the lower one accounts for the rest. Specifically, a block consists of $\frac{B}{l_w}$ words (see §III-A), so for each block, its upper sub-block is its first $p \times \frac{B}{l_w}$ words while its lower sub-block is its last $(1-p) \times \frac{B}{l_w}$ words.
- **Step 3:** HMBR's CR module chooses a new node as the center that downloads all the upper sub-blocks, decodes the upper sub-blocks of f failed blocks, stores one locally, and distributes the remaining $f-1$ upper sub-blocks to the other $f-1$ new nodes. Meanwhile, HMBR's IR module constructs



(a) Traditional CR (b) Rack-Aware CR

Figure 4. Comparison of traditional and rack-aware CR with a (8,4) RS code.

(a) Chain-like IR (b) Tree-pipelined IR

Figure 5. Comparison of chain-like and tree-pipelined IR with a (8,4) RS code.

a pipeline path along the surviving nodes and the new nodes, and decodes the lower sub-blocks of f failed blocks via f pipelined paths using single-block repair methods [16].

- **Step 4:** HMBR puts the repaired upper sub-block and lower sub-block together to obtain the repaired block at each new node.

B. HMBR in Hierarchical Network

We extend HMBR to address hierarchical network topologies, which are common in rack-based cloud storage systems that organize nodes in racks. To tolerate node failures, erasure-coded storage systems [1] often distribute different blocks of a stripe across different nodes which are placed in distinct racks to tolerate both node and rack failures, which will incur substantial cross-rack bandwidth during repair, thus significantly impairing repair performance. Since available cross-rack bandwidth is only $1/5$ or even $1/20$ of inner-rack bandwidth [20], recent studies [20], [21], [22] propose *rack-aware* schemes, which split a repair operation into inner-rack repair sub-operations and cross-rack ones, and place multiple blocks of a stripe in the same rack to trade more inner-rack repairs for less cross-rack repairs.

HMBR focuses on wide-stripe erasure-coded cloud storage, and the wide stripes with a large k are inevitably stored across many racks. We propose, via rack-aware schemes, an enhanced version of HMBR (i.e., rack-aware HMBR), including two approaches to minimize the cross-rack traffic for CR and IR, respectively.

1) Rack-Aware Centralized Repair: We design rack-aware CR to minimize the cross-rack bandwidth during the centralized repair.

Design Idea: Traditional CR (see §II-C) may cause many cross-rack links to the central new node. Our main idea of rack-aware CR is to let a node serve as a *local collector* for each rack to perform inner-rack repairs and the central new node serve as a *global collector* to perform cross-rack repairs. In this way, the cross-rack repair bandwidth can be reduced. This may introduce extra inner-rack traffic, the inner-rack bandwidth is relatively more abundant than cross-rack one though. Thus, our method can still improve the repair performance in most cases.

Design Details: Rack-aware CR will repair f failed blocks in the following five steps. (Step 1) The rack-aware CR lets some node of each rack be a local collector and one of the new nodes be the global collector. (Step 2) All nodes except the local collector transfer their blocks to the local collector in their own racks. (Step 3) All local collectors compute f intermediate blocks for repairing f failed blocks. (Step 4) All local collectors send all the intermediate blocks to the global collector. (Step 5) The global collector reconstructs all the failed blocks and sends them to the other new nodes.

Figure 4 compares traditional and rack-aware CR with a $(8, 4)$ RS code. We assume that every four nodes reside in the same rack. Suppose that N_9 and N_{10} fail. As illustrated in Figure 4(a), the traditional CR requires each of $N_1, \ldots, N_8$ to transfer one block across racks, so the cross-rack repair traffic is eight blocks. Alternatively, as illustrated in Figure

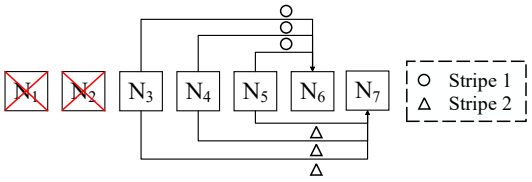


Figure 6. An example of HMBR's multi-node repair with a (3,2) RS code.

4(b), rack-aware CR will select N_2, N_6 as the local collectors, and a new node N_9 as the global collector. The nodes in N_2's rack (i.e., N_1, N_3 and N_4) transfer their blocks to N_2, which computes two intermediate blocks and sends them to the global collector N_9. N_6 does a similar job. In this way, the cross-rack repair traffic is reduced to four blocks.

2) Tree-Pipelined Independent Repair: We design tree-pipelined IR to minimize the cross-rack traffic during the independent repair.

Design Idea: Traditional IR (see §II-D) may cause many cross-rack links when pipelining each single-block repair, which has been addressed by a recent study [16] via limiting cross-rack transmissions of the chain-like pipelined repair path. Even so, however, IR may incur many congested links when pipelining multiple independent chain-like pipelined repair paths, and thus make these shared links (cross-rack or inner-rack) heavily congested. Our idea of rack-aware IR is to borrow the idea of tree-based pipelined single-block repair [8], and extend it to multi-block repair in this paper. In this way, compared to chain-like pipelined repair paths [16], it is more likely that different single-block pipelined repair paths can choose different links to decrease the number of congested shared cross-rack or inner-rack links.

Design Details: Tree-pipelined IR will repair f failed blocks in the following. For each single-block pipelined repair job, the tree-pipelined IR always selects the least frequently used link to construct the current single-block repair pipeline tree until the current job finishes. In this way, all the links, including cross-rack and inner-rack ones, can have as even load as they can, thus alleviating the congestion among the f independent single-block repair jobs.

Figure 5 compares chain-like and tree-pipelined IR with a $(8, 4)$ RS code. Similar to §IV-B1, we assume that every four nodes reside in the same rack. Suppose that N_{11} and N_{12} fail. Here, we let solid-line arrows indicate the first single-block repair job and dotted-line arrows indicate the second single-block repair job. If we use the chain-like IR, as illustrated in Figure 5(a), then both cross-rack links (e.g., $N_2 \rightarrow N_5$ and $N_6 \rightarrow N_9$) and inner-rack links are likely to be congested between two independent repair jobs. Alternatively, tree-pipelined IR has much more choices to construct the two pipelined paths such that many links can be less congested. As illustrated in Figure 5(b), each link is only occupied by one repair job and there is no congested link.

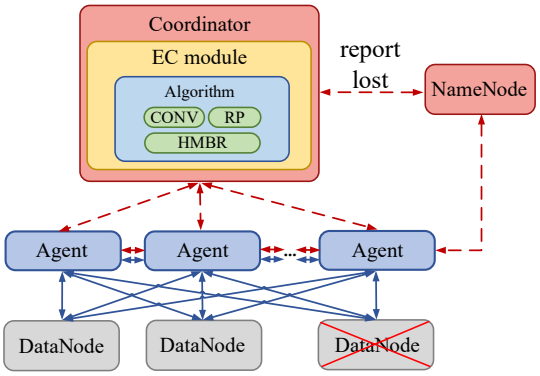


Figure 7. Architecture of HMBR.

C. HMBR for Multi-Node Repair

We now study how HMBR deals with multi-node repair which is composed of multi-block repairs of multiple stripes. We design a scheduling scheme for HMBR to efficiently repair multiple stripes.

Design Idea: Since different stripes can be repaired independently, HMBR can perform multiple single-stripe repair operations in parallel. However, for each stripe, CR will make k surviving nodes transfer one sub-block to the new central node, so for multiple stripes, their chosen central new nodes may be identical ones, which will be overloaded and thus bottleneck the repair process. Thus, our idea is to distribute the multi-stripe repair tasks for CR across all nodes as evenly as possible.

Note that IR often keeps balanced load on each node, especially $f = m$ since the case of $f = m$ means that all the surviving nodes are needed for each single-chunk repair, which often happens in wide-stripe erasure-coded storage systems [7]. Therefore, HMBR only considers how to improve multi-node repair performance of CR regardless of IR.

Design Details: HMBR combines the least-frequently-selected (LFS) way [16] and the least-recently-selected (LRS) way [2]. Specifically, HMBR first sets an array of size f that records each new node's total selected times and then sets a priority queue of size f that records each new node's least recently selected timestamp. When performing CR on a new stripe, HMBR first selects the new node candidates that have the least frequency from the array, and then determines the chosen one that is least recently selected from the priority queue.

Figure 6 shows an example of multi-node repair for HMBR. We consider a (3,2) RS code. We assume that each node has two blocks, i.e., two stripes in the system. Suppose that two nodes fail, and two stripes needed to be repaired. We let the two new nodes N_6 and N_7 have identical selected times and timestamps, so HMBR selects the new node N_6 as center of CR for Stripe 1 and N_7 for Stripe 2. In this way, both N_6 and N_7 can have even load for fair multi-node repair performance.

V. Evaluation

A. Implementation

We implement a prototype of HMBR in C++ with around 1200 lines of codes (LoC). We build HMBR atop OpenEC [23], which can work as a middleware running atop existing storage systems (e.g., HDFS) to instruct the coding operations. OpenEC realizes the erasure coding functionalities based on Intel's Intelligent Storage Acceleration Library (ISA-L) [24] and maintains an in-memory key-value store in each node and utilizes the interfaces of Redis to transfer data.

System Architecture: Figure 7 presents the architecture of HMBR. HMBR contains a centralized coordinator located on the metadata server which manages agents running on the storage nodes (with one agent per node). The coordinator keeps the metadata of erasure coding (e.g., the mapping between stripes and blocks, erasure coding policy, and the information of the block distributions). When the coordinator receives a repair request, it generates a repair solution and distributes the specific tasks to the agents on storage nodes. Then the agents can perform the repair operations cooperatively after receiving the commands from the coordinator.

EC Module: First, we expand the primary EC module in OpenEC and implement HMBR to make an extension to OpenEC's algorithm library. Second, we integrate multi-block repair combined with CR and IR to construct HMBR in OpenEC as well as implement multi-node repair.

Repair flow: When the coordinator has received a block lost report from the metadata server, it first pinpoints the information (e.g., location of the blocks, erasure coding parameters) of the to-be-repaired stripe. It then provides a repair solution for the stripe with the repair algorithm integrated into OpenEC, which includes the selected surviving blocks to participate in the repair, and the repair routing topology. The solution of repairing lost blocks in the stripe will be broken down into agent commands so that the involved agents can perform the repair operations cooperatively.

Integration with Hadoop HDFS: OpenEC can integrate with existing storage systems, like HDFS3, HDFS-RAID, etc. In this paper, we focus on HDFS3, and block/node failures can be detected by HeartBeat at HDFS3's NameNode [25].

B. Experiment Setting

Testbeds: We conduct our experiments on Amazon EC2 with an m4.xlarge instance as the coordinator and 88 m3.large instances as the data nodes. The coordinator is equipped with 4 vCPUs and 16GiB RAM and the data nodes have 2 vCPUs and 7.5GiB RAM. All of the instances are connected via a 10Gb/s network, in the US East (North Virginia) region. All the instances support the optimized coding operations on ISA-L.

Bandwidth datasets: Analysis in §III shows that HMBR has a strong connection with the network bandwidth, and many storage clusters have heterogeneous network bandwidth [26], [10]. Therefore, we generate three datasets (i.e., *WLD-2x*, *WLD-4x*, *WLD-8x*) as the different network environments of our cloud experiments [27]. We generate the datasets base on normal distribution. The difference between these datasets is the *bandwidth gap* (i.e., how many times the fastest node bandwidth differs from the slowest node bandwidth) to implicate network heterogeneity. The bandwidth gap between the fastest node and the slowest node in *WLD-2x* is 2 times and similarly in *WLD-4x* is 4 times and in *WLD-8x* is 8 times. These three datasets are provided in: https://github.com/yuchonghu/HMBR.

Some parameters: We set k ranging from 6 [12] to 64 [2] with a relatively small m. We set the block size to 64MB by default like GFS [15].

C. Experiments

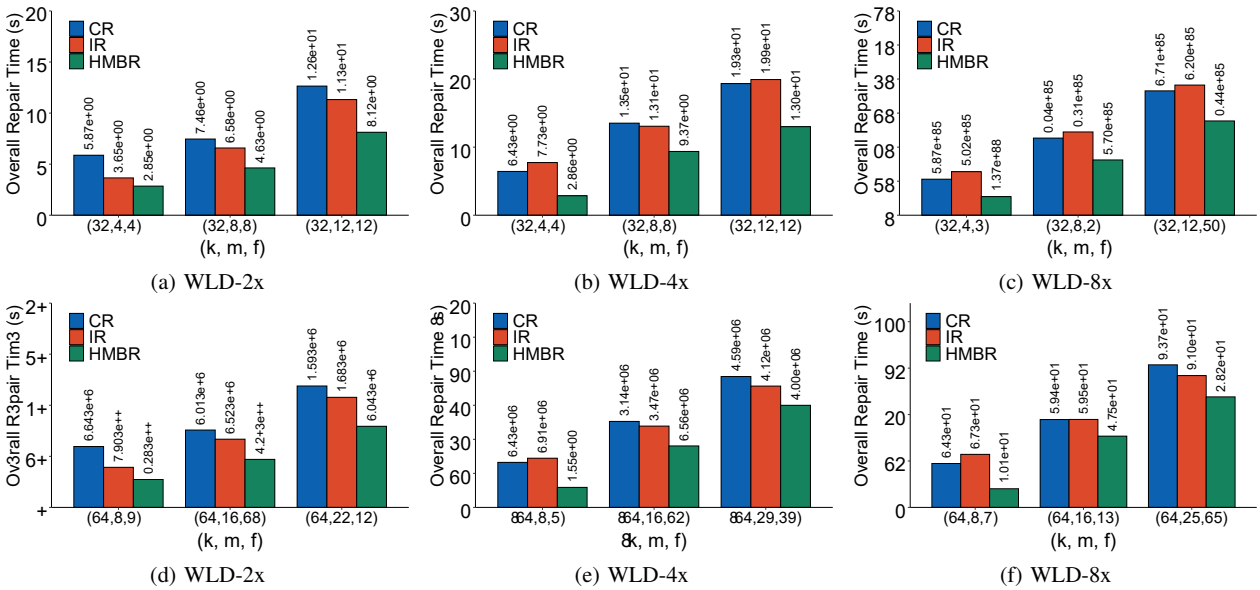
Experiment 1 (Overall multi-block repair time versus (k, m, f)): We measure the overall multi-block repair time consumed by CR, IR, and HMBR with different network workloads under different parameters (k, m, f) respectively. Figure 8(a)-(c) show the overall repair time for CR, IR, and HMBR. Overall, HMBR has less overall repair time than CR and IR. For example, when (k, m, f) = (64, 8, 8) under the WLD-8x, compared to CR and IR, HMBR can reduce the overall repair time by up to 57.5% (compared to CR) and 64.8% (compared to IR), respectively.

We can see that the overall repair time increases rapidly when f becomes larger. This is reasonable because the higher the number of failed blocks, the longer the repair time. On the other hand, we can see that the IR has a better repair performance than CR when the *bandwidth gap* is 1:2, but worse than CR when the *bandwidth gap* is 1:4 or 1:8. The reason is that IR will be bottlenecked by the slowest node of the network, but CR does not. When the *bandwidth gap* is 1:2, the bottleneck is still acceptable for IR. However, when the *bandwidth gap* is 1:4 or 1:8, IR will be significantly influenced. Since HMBR can reduce the bottleneck of both CR and IR, it always has a better repair performance than both of them.

Experiment 2 (Overall multi-block repair time versus f): We measure the overall multi-block repair time versus f for different (k, m) where $f \leqslant m$ under WLD-2x. We set (k, m) to (32, 8) and (64, 16), and let $f = 2, 4, 8$ and $4, 8, 16$, respectively. Figure 9 shows the overall repair time of CR, IR, and HMBR.

We can see that the overall repair time increases rapidly with f. This is because the higher the number of failed blocks, the longer the repair time. In addition, for a fixed (k, m), we can see when f is relatively small or large, CR has longer repair time than IR. The reason is that when f is relatively small, IR has very little bandwidth bottleneck on surviving nodes and thus outperforms CR (see II-D); when f is relatively large, CR makes the central new node very congested and thus underperforms IR. Nevertheless, HMBR always outperforms CR and IR, as explained in Experiment 1.

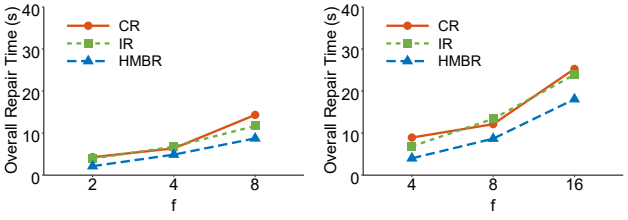
Experiment 3 (Overall multi-block repair time versus block size): We measure the overall multi-block repair time for CR, IR, and HMBR under WLD-4x, with block sizes ranging from 8MB to 64MB. Figure 10 shows the results for $(k, m, f) = (64, 8, 8)$ and $(64, 16, 16)$.



(a) WLD-2x (b) WLD-4x (c) WLD-8x

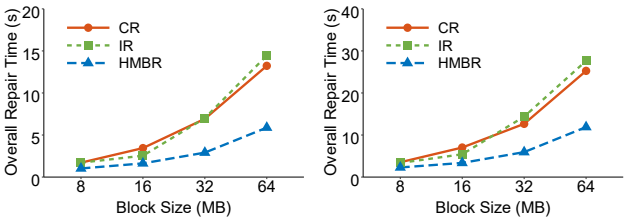
(d) WLD-2x (e) WLD-4x (f) WLD-8x

Figure 8. Experiment 1: Overall multi-block repair time of CR, IR, and HMBR for different (k, m, f) under different network workloads.



(a) (k, m) = (32, 8) (b) (k, m) = (64, 16)

Figure 9. Experiment 2: Overall multi-block repair time versus f.

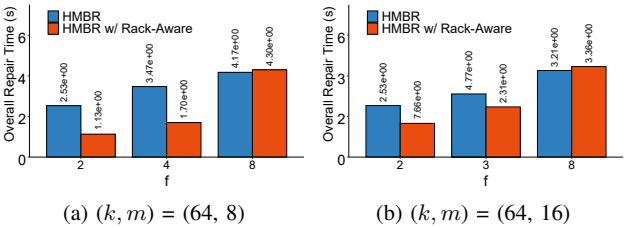


(a) (k, m) = (64, 8) (b) (k, m) = (64, 16)

Figure 10. Experiment 3: Overall multi-block repair time versus block size.

We see that the gaps between HMBR and CR/IR are fairly stable across different block sizes, and HMBR still shows performance gains over CR and IR. Additionally, we see that the overall repair time increases with the block size, simply because larger blocks need more time to repair.

Experiment 4 (Comparison of overall multi-block repair time of HMBR and rack-aware HMBR versus (k, m, f)): We measure the overall multi-block repair time of HMBR and rack-aware HMBR under different (k, m, f). To simulate the heterogeneity of inner-rack and cross-rack bandwidth, we divide all nodes into logical racks. Nodes from the same rack can transfer data to each other with an unrestricted network, while nodes want to communicate with those in other racks, the bandwidth will be limited. We use the Linux traffic tool *tc* to limit the bandwidth of each node to realize the rack topology. We set the number of nodes in each rack is 8.



(a) (k, m) = (64, 8) (b) (k, m) = (64, 16)

Figure 11. Experiment 4: Comparison of overall multi-block repair time of HMBR and rack-aware HMBR versus (k, m, f).

Figure 11 shows the repair performance of HMBR and rack-aware HMBR when (k, m) ranges from (64, 8) and (64, 16) when varying the number of failed blocks f. In general, rack-aware HMBR has a shorter overall repair time than HMBR when f is less than the number of nodes in each rack and thus has better rack-awareness. Rack-aware HMBR reduces the overall repair time by 33.9% on average and up to 55.3% (when (k, m) = (64, 8) and $f = 2$).

Note that when $f = 8$, the repair performance of rack-aware HMBR becomes a little worse than HMBR. The reason is that the number of intermediate blocks (e.g., the cross-rack traffic) that rack-aware HMBR has to transmit will increase with the number of failed blocks. And when f is equal to the number of nodes in each rack, the number of intermediate blocks of rack-aware HMBR is equal to HMBR. Thus rack-aware HMBR cannot reduce the cross-rack traffic compared to HMBR. However, rack-aware HMBR introduces extra inner-rack traffic (see §IV-B). Therefore, the repair performance of rack-aware HMBR becomes a little worse than HMBR when f is equal to the number of nodes in each rack.

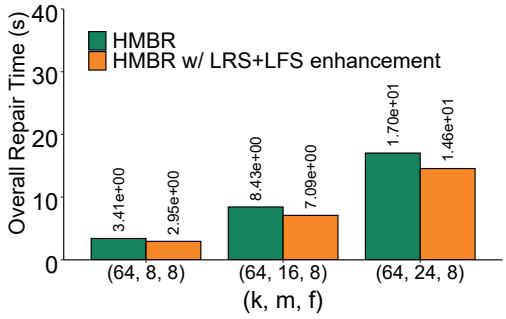


Figure 12. Experiment 5: Overall multi-node repair time of HMBR without and within the multi-node enhancement.

Table II
OVERALL REPAIR TIME BREAKDOWN

Method	(k, m)	T_t/s	T_o/s	T_t / (T_t + T_o)
CR	(32, 4)	9.52	1.08	89.81%
	(64, 8)	21.04	2.56	89.15%
IR	(32, 4)	10.8	2.0	84.38%
	(64, 8)	25.92	2.68	90.63%
HMBR	(32, 4)	4.67	0.79	85.47%
	(64, 8)	8.64	1.46	85.54%

Experiment 5 (Overall multi-node repair time of HMBR with and without the multi-node enhancement): We measure the multi-node repair performance of HMBR without and within the multi-node enhancement under different (k, m, f).

Figure 12 shows the overall multi-node repair time of HMBR with and without enhancement. We observe that our multi-node enhancement can reduce the overall repair time by 10.9% on average and 15.9% at most (when (k, m, f) = (64, 16, 8)), which confirms our method's efficiency.

Experiment 6 (Overall repair time breakdown): We measure the breakdown of the overall repair time and identify the dominant part. We decompose the overall repair time into two main parts: (i) transfer time (denoted by T_t), which refers to the network transmission of blocks, (ii) other time (denoted by T_o), which refers to the time consumed by all other repair process (e.g., CPU computation and disk I/O).

Table II shows the breakdown of the overall multi-block repair time under the bandwidth dataset of *WLD-8x*. We set the (k, m) as (32, 4) and (64, 8) and $f = m$. We can see that transfer time accounts for an average of 87.50% of the overall repair time, thus dominating the repair process. This matches HMBR's objective of minimizing the repair transfer time to improve the overall repair performance (see §III-A).

VI. RELATED WORK

Erasure coding in distributed storage. Erasure coding has been extensively studied in distributed storage systems [1], [28], [29], [30], [20], [16], [2], [23]. Most of studies mainly focus on repair performance [1], [28], [29], [16], [20], scaling performance [31], [5], and update performance [32], [4].

Existing studies on erasure coding mainly focus on small stripes with small k, while wide stripes [2], [5], [4], [6] are only investigated for high performance of single-block repair, encoding and update. In this paper, HMBR aims to improve multi-block repair performance under wide stripes.

Efficient erasure-coded repair. A rich body of studies aim to improve repair efficiency in erasure-coded storage. Regenerating codes [33], [34] minimize the repair bandwidth via enabling surviving nodes to provide computing power. Locally repairable codes [1], [35], [28] trade storage efficiency for repair performance by adding local parity blocks with different subsets of nodes. PPR [8] reduces the single-block repair time by parallelizing the repair operation as partial operations. RP [16] further reduces the single-block repair time by pipelining the repair operation in slices. PPT [9] improves RP by utilizing a pipelined tree. SMFRepair [36] utilizes idle nodes to bypass low-bandwidth links in the heterogeneous network. RepairBoost [10] focuses on improving full-node repair by careful traffic balancing and scheduling.

Existing studies on erasure-coded repair mainly focus on single-block repair, but how to perform multi-block repair efficiently remains unexplored. In this paper, HMBR aims to improve multi-block repair performance for cloud storage.

VII. DISCUSSION AND FUTURE WORK

More datasets: We only consider datasets that obey the normal distribution in §V, and we will study more datasets, such as those that obey uniform and zipf distribution. In addition, we will conduct our experiments on less-contrived workloads, and verify the efficiency of our algorithm on real-world network bandwidth workloads, so as to prove that our algorithm can be applied in practical scenarios.

Dynamic workloads: We only conduct our experiments on static workloads, and we plan to improve HMBR for dynamic workloads, such that our algorithm can cope with the dynamic change of network bandwidth. In addition, we will also optimize the rack-aware HMBR under dynamic workloads so that it will always outperform HMBR without rack awareness.

VIII. CONCLUSIONS

We propose HMBR, a new multi-block repair mechanism for wide-stripe erasure-coded cloud storage. HMBR is based on our analysis which indicates the optimality of how to combine centralized and independent repairs to minimize the repair transfer time. We realize HMBR and extend HMBR to enhance its repair performance for hierarchical network topology and multi-node failures. We prototype HMBR in HDFS and evaluate HMBR on Amazon EC2. Our evaluation demonstrates the efficiency of HMBR in multi-block and multi-node repairs.

ACKNOWLEDGMENT

This work was supported in part by the National Key Research and Development Program of China for Young Scholars (No. 2021YFB0301400), National Natural Science Foundation of China (No. 62272185), National Natural Science Foundation of China (No. 61821003) and Key Laboratory of Information Storage System Ministry of Education of China. The corresponding author is yuchonghu@hust.edu.cn.

REFERENCES

[1] 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.

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

[3] "VAST data," https://vastdata.com/providingresilience-efficiently-part-ii/.

[4] L. Cheng, Y. Hu, Z. Ke, J. Xu, Q. Yao, D. Feng, W. Wang, and W. Chen, "LogECMem: coupling erasure-coded in-memory key-value stores with parity logging," in *Proc. of SC*, 2021.

[5] Q. Yao, Y. Hu, L. Cheng, P. P. Lee, D. Feng, W. Wang, and W. Chen, "Stripemerge: Efficient wide-stripe generation for large-scale erasure-coded storage," in *Proc. of IEEE ICDCS*. IEEE, 2021.

[6] X. Yang, W. Gu, G. Li, X. Li, and Z. Dong, "Xhr-code: An efficient wide stripe erasure code to reduce cross-rack overhead in cloud storage systems," in *Proc. of IEEE SRDS*, 2022.

[7] M. Luby, R. Padovani, T. J. Richardson, L. Minder, and P. Aggarwal, "Liquid cloud storage," *ACM Trans. on Storage*, vol. 15, no. 1, pp. 1–49, 2019.

[8] S. Mitra, R. Panta, M.-R. Ra, and S. Bagchi, "Partial-parallel-repair (PPR): a distributed technique for repairing erasure coded storage," in *Proc. of ACM Eurosys*. ACM, 2016.

[9] Y. Bai, Z. Xu, H. Wang, and D. Wang, "Fast recovery techniques for erasure-coded clusters in non-uniform traffic network," in *Proc. of ICPP*, 2019.

[10] S. Lin, G. Gong, Z. Shen, P. P. Lee, and J. Shu, "Boosting full-node repair in erasure-coded storage," in *Proc. of USENIX ATC*, 2021.

[11] I. S. Reed and G. Solomon, "Polynomial codes over certain finite fields," *Journal of the society for industrial and applied mathematics*, vol. 8, no. 2, pp. 300–304, 1960.

[12] K. V. 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.

[13] S. A. Weil, S. A. Brandt, E. L. Miller, D. D. Long, and C. Maltzahn, "Ceph: A scalable, high-performance distributed file system," in *Proc. of USENIX OSDI*, 2006.

[14] "Openstack swift object storage service," http://swift.openstack.org.

[15] S. Ghemawat, H. Gobioff, and S.-T. Leung, "The google file system," in *Proc. of ACM SOSP*, 2003.

[16] X. Li, Z. Yang, J. Li, R. Li, P. P. Lee, Q. Huang, and Y. Hu, "Repair pipelining for erasure-coded storage: Algorithms and evaluation," *ACM Trans. on Storage*, vol. 17, no. 2, pp. 1–29, 2021.

[17] A. Cidon, S. Rumble, R. Stutsman, S. Katti, J. Ousterhout, and M. Rosenblum, "Copysets: Reducing the frequency of data loss in cloud storage," in *Proc. of USENIX ATC*, 2013.

[18] J. F. Kurose and K. W. Ross, "Computer networking: A top-down approach sixth edition," 2012.

[19] "Bolzano's theorem," https://mathworld.wolfram.com/BolzanosTheorem.html.

[20] 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*, vol. 13, no. 4, p. 33, 2017.

[21] Z. Shen, P. P. Lee, J. Shu, and W. Guo, "Cross-rack-aware single failure recovery for clustered file systems," *IEEE Trans. on Dependable and Secure Computing*, vol. 17, no. 2, pp. 248–261, 2017.

[22] H. Hou, P. P. Lee, K. W. Shum, and Y. Hu, "Rack-aware regenerating codes for data centers," *IEEE Trans. on Information Theory*, vol. 65, no. 8, pp. 4730–4745, 2019.

[23] 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.

[24] "Intelligent storage acceleration library," https://github.com/01org/isa-l.

[25] "Hadoop 3.0.0," https://hadoop.apache.org/docs/r3.0.0/.

[26] Q. Yao, Y. Hu, X. Tu, P. P. Lee, D. Feng, X. Zhu, X. Zhang, Z. Yao, and W. Wei, "Pivotrepair: Fast pipelined repair for erasure-coded hot storage," in *Proc. of IEEE ICDCS*, 2022.

[27] A. Varet and N. Larrieu, "How to generate realistic network traffic?" in *Proc. of IEEE COMPSAC*, 2014.

[28] 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 ACM VLDB Endowment*, 2013.

[29] H. Qiu, C. Wu, J. Li, M. Guo, T. Liu, X. He, Y. Dong, and Y. Zhao, "Ec-fusion: An efficient hybrid erasure coding framework to improve both application and recovery performance in cloud storage systems," in *Proc. of IEEE IPDPS*, 2020.

[30] J. Gu, C. Wu, X. Xie, H. Qiu, J. Li, M. Guo, X. He, Y. Dong, and Y. Zhao, "Optimizing the parity check matrix for efficient decoding of rs-based cloud storage systems," in *Proc. of IEEE IPDPS*, 2019.

[31] S. Wu, Q. Du, P. P. Lee, Y. Li, and Y. Xu, "Optimal data placement for stripe merging in locally repairable codes," in *Proc. of INFOCOM*, 2022.

[32] Z. Shen and P. P. Lee, "Cross-rack-aware updates in erasure-coded data centers," in *Proc. of ICPP*, 2018.

[33] A. G. Dimakis, P. B. Godfrey, Y. Wu, M. J. Wainwright, and K. Ramchandran, "Network coding for distributed storage systems," *IEEE Trans. on Information Theory*, vol. 56, no. 9, pp. 4539–4551, 2010.

[34] L. Pamies-Juarez, F. Blagojevic, R. Mateescu, C. Gyuot, E. E. Gad, and Z. Bandic, "Opening the chrysalis: On the real repair performance of MSR codes," in *Proc. of USENIX FAST*, 2016.

[35] O. Kolosov, G. Yadgar, M. Liram, I. Tamo, and A. Barg, "On fault tolerance, locality, and optimality in locally repairable codes," *ACM Trans. on Storage*, vol. 16, no. 2, pp. 1–32, 2020.

[36] H. Zhou, D. Feng, and Y. Hu, "Multi-level forwarding and scheduling repair technique in heterogeneous network for erasure-coded clusters," in *Proc. of ICPP*, 2021.