Distributed Storage Codes Based on Double-Layered Piggybacking Framework

RONG SUN\(^{1,2}\), (Member, IEEE), XIN LI\(^1\), LU ZHANG\(^1\), AND JINGWEI LIU\(^3\), (Member, IEEE)

\(^{1}\)State Key Laboratory of Integrated Services Networks, Xidian University, Xi’an 710071, China
\(^{2}\)Xiamen Key Laboratory of Mobile Multimedia Communications, Huaqiao University, Xiamen 361021, China
\(^{3}\)Shaanxi Key Laboratory of Blockchain and Secure Computing, Xidian University, Xi’an 710071, China

Corresponding author: Rong Sun (rsun@mail.xidian.edu.cn)

This work was supported in part by the Key Industry Innovation Chain Project of Shaanxi under Grant 2020ZDLGY05-04, Grant 2019ZDLGY13-05, and Grant 2019ZDLGY12-01, in part by the National Natural Science Foundation of China under Grant 61671340, and in part by the Collaborative Innovation Center of Information Sensing and Understanding at Xidian University.

ABSTRACT Piggybacking codes are a class of distributed storage codes adopted in distributed storage systems, which received much attention in recent years. Two novel piggybacking constructions based on the double-layered piggybacking framework are proposed. They can reduce the repair bandwidth of both the systematic nodes and the parity nodes efficiently. The average repair bandwidth ratio for the two constructions approaches to zero asymptotically as the number of systematic nodes and parity nodes tends to infinity. Compared with other piggybacking codes, the two proposed double-layered piggybacking codes, especially the second one, not only require less design constraints, but also obtain the optimal comprehensive repair efficiency, which further save the amount of data read and downloaded.

INDEX TERMS Distributed storage, piggybacking codes, MDS, repair bandwidth ratio, edge computing.

I. INTRODUCTION

With the advent of the 5G era, applications have become more and more abundant, and the number of users has also increased, which has led to a geometric growth of data [1]. The performance and scalability of storage systems are posing a serious challenge due to the large scale and the rapid growth of data. In the traditional centralized storage systems, with large scale and rapid growth of data, the burden of the storage servers has been the bottleneck of the system, which makes the reliability and security untenable and they cannot meet the requirement of big data applications [2]. At the same time, the distributed storage systems receive much attention because of their higher reliability and stronger scalability. However, various kinds of malfunctions, such as power failure, system maintenance, network interruption, etc., will occur frequently due to the existence of a large number of separated nodes [3]. In order to retain the reliability of the whole storage system, replication and erasure codes [4] are used as the two common fault-tolerant techniques. Replication is the simplest redundant fashion and is used in Hadoop Distributed File System, Google File System, Windows Azure, etc.. But it provides low storage efficiency. On the contrary, erasure codes introduce some redundancy obtaining much higher storage efficiency and reliability. They are adopted by many distributed storage systems such as Google Colossus, HDFS Raid, Microsoft Azure, OceanStore, and so on [5], [6]. To retain the fault tolerance of the whole storage system and the reliability of data downloaded, the failed nodes need to be repaired in time. A good fault-tolerant technique requires that the storage system has low redundancy overhead and low node repair bandwidth. Consequently, how to reduce the repair bandwidth of a failed node has become a research hotspot in the study of distributed storage codes. For this purpose, several distributed storage codes with low repair bandwidth are presented, such as the regenerating codes [7]–[11] and the local repairable codes (LRCs) [12]–[15]. But their construction procedures are very complex due to matrix operations during the repair process. Furthermore, MDS (maximum distance separable) array codes, such as EVENODD codes [16], [17], B-code [18], X-code [19], RDP [20], [21], STAR [22] and Zigzag codes [23], etc. with low complexity are presented. Their encoding and decoding procedures are basically based on...
the simple operation of low-order finite field. Due to the
construction procedures of MDS array codes, they have fewer
redundant nodes, which makes the fault-tolerant capability
limited and the repair efficiency low.

In 2013, K.V. Rashmi et al. proposed the idea of piggybacking framework to reduce the repair bandwidth for node repair in distributed storage systems [24]. The proposed piggybacking framework is based on an MDS code where data symbols in the instances are embedded into the piggyback blocks in other instances via the well-designed piggyback functions. The missing data symbols can be recovered by solving the piggyback functions instead of decoding the traditional MDS code, which can reduce the repair bandwidth effectively. The piggybacking codes have a distinguished feature that they can reduce the extra storage overhead without changing the node distribution structure. No additional node is added except the parity check nodes of MDS codes. Moreover, satisfying the properties of MDS codes, the piggybacking codes can achieve high-rate, have small number of substripes, and support any codeword length \( n \) or information length \( k \) of the base code (the value of parameters is only depended on the requirement of the base code). Some codes such as the Rotated-RS codes must satisfy that the number of parity nodes \( r \in \{2, 3\} \) and the number of systematic nodes \( k \leq 36 \) [25], and the EVENODD as well as RDP codes only can be achieved at \( r = 2 \). The piggybacking codes have been adopted by Facebook Warehouse Cluster, Hadoop Distributed File System (HDFS) and other systems [26] for their flexible design, low complexity and convenient implementation.

Since the invention of piggybacking codes, researchers are trying to find effective piggybacking construction method to repair the failed nodes. In [27], S. Yuan et al. introduced generalized piggybacking codes to repair the systematic nodes and proved that the average repair bandwidth ratio of systematic nodes can approach to zero with the number of parity nodes tending to infinity, which is very close to the cut-set bound of distributed storage codes. In [28], S. Kumar et al. also investigated a piggybacking framework for repairing systematic nodes, which is based on two classes of parity nodes. The primary goal of the first class of parity nodes is to provide good fault tolerance, while the second is to reduce the repair bandwidth and the repair complexity. With the number of the second class of parity nodes increasing, the repair bandwidth of the systematic nodes can be reduced, sacrificing the storage overhead of the system. C. Shangguan et al. presented a piggybacking construction for systematic nodes in [29], in which the systematic nodes are divided into irrelevant subsets with various piggyback blocks. Systematic nodes in different subsets have different repair bandwidth. Thus higher repair efficiency can be obtained and little work is done on the construction design for repairing parity nodes. B. Yang et al. considered a piggybacking framework to repair parity nodes and preserve the repairability of systematic nodes at the same time in [30].

It is worth mentioning that a large-scale distributed storage system will cost a large amount of network bandwidth if a failed node completely relies on the cloud center to be repaired. At the same time, the bottleneck of the whole system performance becomes the bottleneck of the network bandwidth, because the processing speed of the cloud center is much more than the speed of data transmission in the network. Obviously, if the computing tasks of the node repair can be completed at the edge nodes and only the result is submitted to the cloud center, not only can the unnecessary bandwidth be saved, but also the response time can be reduced, which improves the performance of system tremendously. In addition, performing edge computing tasks at the local nodes can greatly reduce the risk of data being leaked in network transmission, which can effectively protect data security and privacy.

As shown in Fig. 1, a collaborative edge frame, which connects the multiple edge nodes that are geographically distributed despite their physical location and the network structure is proposed in [31]. Edge node is a physical small data center that connects users and cloud with data processing capability. The requirement for data processing will be transmitted between the cloud and the edge nodes, and the data will be transmitted between the users and the edge nodes. The edge nodes can not only request service and content from the cloud but also perform the computations locally, such as computing offloading, data storage, caching and processing, along with IoT management etc. In view of those abilities, the edge nodes need to be well-designed to achieve the goal of data reliability, security, and privacy protection [32].

From what has been discussed above, it is safe to infer that these distributed storage edge nodes provide the opportunity to process and share data for users. Therefore, Edge computing has the significantly potential to address the concern of repairing failed nodes in distributed storage systems with piggybacking codes.

In this paper, we propose two kinds of novel double-layered piggybacking constructions that can reduce the repair bandwidth of both the systematic nodes and the parity nodes efficiently. Asymptotically, the average repair bandwidth...
The piggybacking framework operates on $m$ instances of the base code (we use the MDS code) and well-designed piggyback functions are introduced to add the information of some instances to other ones. The piggybacking framework adopts the systematic encoding form. Since the systematic nodes store all the original data symbols, the users can directly download the original data from the data symbols without using the decoding algorithm. Thus, the complexity and delay of data recovery are greatly reduced, which makes the information transmission more efficient.

For each substripe $i$, $2 \leq i \leq m$, the piggyback functions are added. The vectors $\{a_i\}_{i=1}^{m}$ are independent inputs and $\{g_{ij}, i \in \{2, \ldots, m\}, j \in \{1, \ldots, n\}\}$ are the piggyback functions. The fundamental piggybacking framework is shown in Fig. 3.

Clearly, the first column of the framework is identical to a codeword of the base code. We can get $a_1$ directly using the decoding procedure of this base code. In the second instance, subtracting the piggyback functions $\{g_{2,1}\}_{i=1}^{m}$ from the elements of the instance, the message vector $a_2$ can also be decoded via the base code decoding. Continuing this way, for any instance $i, (2 \leq i \leq m)$, the piggyback functions can be subtracted and the base code of that instance can be obtained, which can be decoded using the base code decoding. In particular, we may enable piggybacking codes to achieve our desired goals (eg. lower the repair bandwidth) by designing flexible piggyback functions.

### III. DOUBLE-LAYERED PIGGYBACKING CODES

Distributed storage systems provide data to users quickly and effectively relying on the availability provided by systematic nodes, and the systems retain this availability depending on the reliability provided by parity nodes. Neither of the two kinds of nodes can exist independently in distributed storage systems for a long time. Whereas the existing piggybacking codes are designed mostly for either repairing the systematic nodes or repairing the parity nodes. We propose a double-layered piggybacking framework that can efficiently repair both the systematic nodes and the parity nodes at the same time. The first layer of the piggybacking design is to repair systematic nodes, and the second layer is for repairing parity nodes. We introduce the construction method using a simple example firstly, then we generalize it to the general condition.

An erasure code consists of $n$ units, where $k$ of these units are systematic units, and $r$ are parity units. We call this set...
The data symbols to be embedded are arranged sequentially into a row from left to right in order, i.e., \( f_{1}^{(1)} , f_{1}^{(2)} , f_{1}^{(5)} , \ldots , f_{4}^{(2)} , f_{4}^{(5)} , f_{4}^{(6)} \). Then they are embedded into the piggyback blocks in each parity node one by one. In our example, the two reserved substripes of the first parity node do not participate in piggyback functions because they are the origination of repairing systematic nodes. The first embedded data symbol is embedded in the first piggyback block. Then the following symbol is embedded in the next piggyback block. When each piggyback block has one data symbol and there are data symbols left, we will return to the first piggyback block to embed the data symbol in order till all the data symbols are embedded.

To repair the parity nodes, the two backmost substripes are selected for embedding piggyback blocks. The parity symbols of all the parity nodes are embedded in the two substripes in a specific way. We regard each piggyback block in parity nodes and its corresponding parity symbol of substripes 3 and 4 as a “black box” \( B \), which is embedded in piggyback blocks again in the substripes 5 and 6, e.g., \( B_{k+2}^{(3)} = f_{k+2}^{(2)} + f_{1}^{(1)} \). Using the “black box”, we don’t care about which data symbols or parity symbols are in it but just treat the “black boxes” as ordinary parity symbols.

Similar to the data symbols, the parity symbols also need to be dispersed as much as possible. One difference is that the repair of parity nodes depends on the parity nodes associated with them. Therefore, it is necessary that the symbols of a parity node cannot be embedded into the piggyback blocks of the node itself. If the parity node fails, the symbols and the piggyback block associated with them are also lost, then the symbols cannot be solved by piggyback functions. Thus, we use a diagonal-order method as shown in Fig.4 to form the piggyback blocks in substripes 5 and 6, which can avoid the scenario we mentioned.

A few notes on the code design of the double-layered piggybacking framework are given as follows.

- Firstly, the reserved substripes of systematic nodes and the piggybacked substripes of parity nodes are not overlapped in this example. If the two types of substripes are overlapped, to repair the systematic nodes, what we need to read are not only the data symbols embedded in the piggyback blocks, but also the parity symbols embedded. It is clear that the non-overlapped design needs less repair bandwidth than the overlapped design for repairing the systematic nodes. To repair the parity nodes, the symbols need to be read are not only the parity symbols embedded in the piggyback blocks, but also the data symbols embedded. The number of data symbols is more than the number of “black boxes”. Obviously, the non-overlapped design needs less repair bandwidth than the overlapped design for repairing the parity nodes, too. It is because that the “black boxes” hide the symbols read unnecessarily and ensure that no extra repair bandwidth will be introduced when the two classes of nodes are repaired. Moreover, only in

A. THE DOUBLE-LAYERED PIGGYBACKED (9, 4) MDS CODE

Using a (9, 4) systematic MDS code as the base code, the double-layered piggybacked (9, 4) MDS code with 6 substripes is depicted in Fig.4. Each systematic node \( j(1 \leq j \leq 4) \) stores the data symbols \( \{ f_{ij}^{(1)} \}_{i=1}^{6} \), and each parity node \( k+r(4 \leq k \leq 9, 1 \leq r \leq 5) \) stores the parity symbols \( \{ f_{k+r,j}^{(i)} \}_{i=1}^{6} (1 \leq r \leq 5) \).

1) CODE DESIGN

To repair the systematic nodes, the substripes 3 and 4 are selected as reserved substripes for embedding the data symbols into piggyback blocks. All the symbols except the two substripes are embedded into the latter 4 parity nodes on the two reserved substripes.

To ensure that each systematic node can be recovered successfully, the data symbols in a systematic node cannot be embedded in the same piggyback block. Therefore, the data symbols need to be dispersed as much as possible [27]. According to the analysis, when the number of piggyback blocks is a constant, the number of data symbols embedded in every piggyback block directly affects the repair bandwidth of systematic nodes. So, we choose the best solution and adopt the general idea [24], [27], [29], [37]. In our example, the embedding method is explained as follows.

| substripes number | 1 | 2 | 3 | 4 | 5 | 6 |
|-------------------|---|---|---|---|---|---|
| Node 1            | \( f_{1}^{(1)} \) | \( f_{2}^{(1)} \) | \( f_{3}^{(1)} \) | \( f_{3}^{(2)} \) | \( f_{4}^{(2)} \) | \( f_{4}^{(6)} \) |
| Node 2            | \( f_{1}^{(1)} \) | \( f_{2}^{(1)} \) | \( f_{3}^{(1)} \) | \( f_{3}^{(2)} \) | \( f_{4}^{(2)} \) | \( f_{4}^{(6)} \) |
| Node 3            | \( f_{1}^{(1)} \) | \( f_{2}^{(1)} \) | \( f_{3}^{(1)} \) | \( f_{3}^{(2)} \) | \( f_{4}^{(2)} \) | \( f_{4}^{(6)} \) |
| Node 4            | \( f_{1}^{(1)} \) | \( f_{2}^{(1)} \) | \( f_{3}^{(1)} \) | \( f_{3}^{(2)} \) | \( f_{4}^{(2)} \) | \( f_{4}^{(6)} \) |
| Node \( k+1 \)     | \( f_{1}^{(1)} \) | \( f_{2}^{(1)} \) | \( f_{3}^{(1)} \) | \( f_{3}^{(2)} \) | \( f_{4}^{(2)} \) | \( f_{4}^{(6)} \) |
| Node \( k+2 \)     | \( f_{1}^{(1)} \) | \( f_{2}^{(1)} \) | \( f_{3}^{(1)} \) | \( f_{3}^{(2)} \) | \( f_{4}^{(2)} \) | \( f_{4}^{(6)} \) |
| Node \( k+3 \)     | \( f_{1}^{(1)} \) | \( f_{2}^{(1)} \) | \( f_{3}^{(1)} \) | \( f_{3}^{(2)} \) | \( f_{4}^{(2)} \) | \( f_{4}^{(6)} \) |
| Node \( k+4 \)     | \( f_{1}^{(1)} \) | \( f_{2}^{(1)} \) | \( f_{3}^{(1)} \) | \( f_{3}^{(2)} \) | \( f_{4}^{(2)} \) | \( f_{4}^{(6)} \) |
| Node \( k+5 \)     | \( f_{1}^{(1)} \) | \( f_{2}^{(1)} \) | \( f_{3}^{(1)} \) | \( f_{3}^{(2)} \) | \( f_{4}^{(2)} \) | \( f_{4}^{(6)} \) |

The table of Black boxes with corresponding internal symbols

| Black box internal symbols | Black box internal symbols |
|---------------------------|---------------------------|
| \( B_{1} \) + \( (c_{1} - c_{2}) \) | \( B_{1} \) + \( (c_{1} - c_{2}) \) |
| \( B_{2} \) + \( (c_{1} - c_{2}) \) | \( B_{2} \) + \( (c_{1} - c_{2}) \) |
| \( B_{3} \) + \( (c_{1} - c_{2}) \) | \( B_{3} \) + \( (c_{1} - c_{2}) \) |
| \( B_{4} \) + \( (c_{1} - c_{2}) \) | \( B_{4} \) + \( (c_{1} - c_{2}) \) |
| \( B_{5} \) + \( (c_{1} - c_{2}) \) | \( B_{5} \) + \( (c_{1} - c_{2}) \) |
| \( B_{6} \) + \( (c_{1} - c_{2}) \) | \( B_{6} \) + \( (c_{1} - c_{2}) \) |

FIGURE 4. The double-layered piggybacking framework of a (9, 4) piggybacking code.
non-overlapped design, can “black boxes” exist. Hence, we use a non-overlapped piggybacking design.

- Secondly, we analyze the case of whether the parity symbols on the reserved substripes of the first parity node (i.e., $f_{k+1}^{(3)}$ and $f_{k+1}^{(4)}$ in this example) are embedded in piggyback blocks or not. If the two parity symbols are not embedded, they can only be recovered by the decoding process of base codes. On the contrary, suppose that they are embedded in piggyback blocks, they can be recovered via solving the piggyback functions, which increases the number of symbols in the piggyback blocks unfortunately, resulting in the increase of the repair bandwidth for the other symbols. However, when the number of these symbols is small, the two methods have little influence on the performance of the whole framework. In order to understand the two-layered piggybacking framework we proposed, the embedding method is adopted in this paper.

- Finally, there is another method to design the piggyback blocks for parity nodes. We place all the parity symbols “black boxes” according to the order of the matrix arrangement: starting from the first row of the first column of the matrix to the last row, then to the first row of the second column until all the elements of the matrix are embedded. If the number of parity nodes is odd, the number of parity symbols is symmetrical about the middle number. In this way, parity symbols in a parity node are easily embedded in the piggyback blocks of this parity node, and we have to make additional adjustments. Consequently, we adopt the diagonal-embedding method. The diagonal method can also be used to construct piggyback blocks for systematic nodes.

2) REPAIR ANALYSES
Assume that the first systematic node fails. All the data symbols $\{f_{j,1}^{(1)}, f_{j,4}^{(1)}\}_{j=1}^{2}$ and the parity symbols $\{f_{k+1,1}^{(3)}, f_{k+1,4}^{(3)}\}$ on the reserved substripes are read to recover the missing data $\{f_{1}^{(1)}, f_{1}^{(4)}\}$ via the MDS code decoding. The amount of data read in this step is 8. Finally, the symbols $\{f_{k+3}^{(1)}, f_{k+4}^{(2)}, f_{k+5}^{(3)}, f_{k+1}^{(4)}\}$ (we have calculated $f_{k+2}^{(5)}$, $f_{k+2}^{(6)}$ in step 1) are downloaded to repair the piggyback block of the failed parity node. The amount of data read in this step is 4. The total amount of data read is $8 + 8 + 4 = 20$. The repair strategy of other parity nodes is similar.

B. THE GENERAL DOUBLE-LAYERED PIGGYBACKING FRAMEWORK
Consider $s$ substripes of a $(k + r, k)$ systematic MDS code as the base code. The number of reserved substripes for repairing systematic nodes is $a$ and the number of piggybacked substripes for repairing parity nodes is $b$. $\{f_{d_{ij}}^{(s-b-l+i+1)}\}_{j=r,i=a}^{l=2,i=b}$ are the piggybacked blocks embedding data symbols and they are also the “black boxes”. $\{f_{d_{bj}}^{(s-l+1)}\}_{j=1,i=1}^{l=1}$ are the piggyback blocks embedding parity symbols and “black boxes”. The embedding methods of the two classes of piggyback blocks are the same as the methods in the (9, 4) code given before. We obtain a general double-layered piggybacking framework, refer to Fig.5.

1) REPAIRING SYSTEMATIC NODES
In the general double-layered piggybacking framework, there are $a(r - 1)$ piggyback blocks (or black boxes) in the $a$ reserved substripes. Without loss of generality, specify the $a(r - 1)$ piggyback blocks contain $n_1, n_2, \ldots, n_{a(r-1)}$ data symbols respectively. Obviously, $\sum_{j=1}^{a(r-1)} n_j = k(s - a)$, where $k(s - a)$ is the number of data symbols being embedded. Suppose that the $j$th systematic node fails, $i \in \{1, 2, \ldots, k\}$, the repair strategy of reconstituting the missing symbols $f_{j}^{(1)}$ is shown in Algorithm 1.

Algorithm 1 Repairing the Systematic Nodes

Step 1: Read all the surviving data symbols $f_{j}^{(1)}$ for $j \neq i$ in the $a$ reserved substripes, and the parity symbols $f_{k+4}^{(3)}$ of the $a$ missing data symbols $f_{j}^{(3)}$ in the reserved substripes can be recovered via the MDS code decoding.

Step 2: Read the piggyback blocks where the remaining missing data symbols are embedded and the other data symbols in these piggyback blocks except the missing data symbols. Then we can get the remaining $s-a$ missing data symbols by solving the piggyback functions.

The remaining $s-a$ missing data symbols are embedded in $s-a$ piggyback blocks respectively. Let the number of data embedded in these blocks be $n_{ij}$ respectively. The number of extra data symbols (in the same piggyback blocks except the missing data symbols) needing to be read is $\sum_{j=1}^{a} n_{ij} = s + a$.

The amount of data read in Step 1 is $ka$, and there are $k$ systematic nodes. So in this part, the repair bandwidth is $k^2a$. Consider the $i$th piggyback block, $1 \leq i \leq a(r - 1)$,
In what follows, we obtain a lower bound and an upper bound of \( \eta \) being the number of almost evenly embedded data symbols, \( n \) is the number of systematic nodes is \( k \). By Eq. (2), we have

\[
\eta_{D1}^{sys} = k^2a + \sum_{i=1}^{a(r-1)} n_i^2.
\]

The average repair bandwidth ratio of systematic nodes \( \eta_{D1}^{sys} \) is

\[
\eta_{D1}^{sys} = \frac{1}{k^2s}(k^2a + \sum_{i=1}^{a(r-1)} n_i^2)
= \frac{1}{k^2s}(k^2a + \frac{\sum_{i=1}^{a(r-1)} n_i^2 + \sum_{i,j=1; i \neq j}^{a(r-1)} (n_i - n_j)^2}{a(r-1)})
= \frac{1}{k^2s}(k^2a + \frac{k^2(s-a)^2 + \sum_{i,j=1; i \neq j}^{a(r-1)} (n_i - n_j)^2}{a(r-1)}).
\]

Define the following three variables,

\[
\begin{align*}
    u_l &= \frac{k(s-a)}{a(r-1)}, \\
    u &= k(s-a) - u_l(r-1)a,
\end{align*}
\]

where \( u_l \) is the quotient and \( u \) is the remainder of \( \frac{k(s-a)}{a(r-1)} \). Being the number of almost evenly embedded data symbols, \( n_i - n_j \) is either 1 or 0. Thus \( \sum_{i,j=1; i \neq j}^{a(r-1)} (n_i - n_j)^2 = u(a(r-1) - u) \). By Eq. (2), we have

\[
\eta_{D1}^{sys} = \frac{1}{k^2s}(k^2a + \frac{k^2(s-a)^2}{a(r-1)} + \frac{u(a(r-1) - u)}{a(r-1)}).
\]

In what follows, we obtain a lower bound and an upper bound of \( \eta_{D1}^{sys} \). The discussions of this issue are as follows.

- If \( k(s-a) \) is a multiple of \( a(r-1) \), i.e., \( u \equiv 0 \), the data symbols are embedded in each piggyback blocks evenly. By Eq. (4), we can have

\[
\eta_{D1}^{sys} \geq \frac{1}{k^2s}(k^2a + \frac{k^2(s-a)^2}{a(r-1)})
= \frac{a}{s} + \frac{(s-a)^2}{a(r-1)s}.
\]

Define \( p_a = a/s \) being the proportion of reserved substripes. It holds that

\[
\eta_{D1}^{sys} \geq \frac{1}{k^2s}\left(k^2a + \frac{k^2(s-a)^2}{a(r-1)}\right)
= \frac{a}{s} + \frac{(s-a)^2}{a(r-1)s}.
\]

Hence, a lower bound of \( \eta_{D1}^{sys} \) for the double-layered piggybacking codes is

\[
\Gamma_{low}(\eta_{D1}^{sys}) = p_a + \frac{1}{r-1} \cdot \frac{(1 - p_a)^2}{p_a}.
\]

Obviously, \( \Gamma_{low}(\eta_{D1}^{sys}) \) is a function of \( p_a \) and the number of parity nodes \( r \).

- If \( k(s-a) \) is not a multiple of \( a(r-1) \), i.e., \( u \neq 0 \), data symbols embedded in each piggyback blocks are uneven. According to the inequality of arithmetic and geometric means, it is easy to get

\[
\begin{align*}
    u(a(r-1) - u) &\leq \left(\frac{a + (a(r-1) - u)}{2}\right)^2 \\
    &\leq \frac{(a(r-1))^2}{4}.
\end{align*}
\]

Consequently, by Eq. (4), we can have

\[
\eta_{D1}^{sys} \leq \frac{1}{k^2s}(k^2a + \frac{k^2(s-a)^2}{a(r-1)} + \frac{a(r-1)}{4})
= \frac{a}{s} + \frac{(s-a)^2}{a(r-1)s} + \frac{a(r-1)}{4k^2s}
= \frac{a}{s} + \frac{(1 + \frac{r-1}{4k^2})}{a(r-1)s} \cdot \frac{(s-a)^2}{4k^2s}.
\]

Because \( p_a = a/s \) is the proportion of reserved substripes. It holds that

\[
\eta_{D1}^{sys} \leq p_a(1 + \frac{r-1}{4k^2s}) + \frac{1}{r-1} \cdot \frac{(1 - p_a)^2}{p_a}.
\]

Hence, a upper bound of \( \eta_{D1}^{sys} \) for the double-layered piggybacking codes is

\[
\Gamma_{up}(\eta_{D1}^{sys}) = p_a(1 + \frac{r-1}{4k^2s}) + \frac{1}{r-1} \cdot \frac{(1 - p_a)^2}{p_a}.
\]

Clearly, \( \Gamma_{up}(\eta_{D1}^{sys}) \) is a function of \( p_a \) and the number of both systematic nodes \( k \) and parity nodes \( r \).

Suppose one case that the code rate of the base code is 0.5, the curves of \( \Gamma_{low}(\eta_{D1}^{sys}) \) and \( \Gamma_{up}(\eta_{D1}^{sys}) \) with different \( p_a \) for...
2) ANALYSES OF REPAIRING PARITY NODES

For the general double-layered piggybacking framework, there are $b \cdot r$ piggyback blocks in the $b$ piggybacked substripes. Without loss of generality, define the $b \cdot r$ piggyback blocks contain $m_1, m_2, \ldots, m_{br}$ symbols (including the parity symbols and the “black boxes”). $\sum_{j=1}^{br} m_j = r(s - b)$, $r(s - b)$ is the number of the parity symbols and the “black boxes” that need to be embedded. Suppose that the $r$th parity node fails, we use the following algorithm to reconstruct the missing symbols which are in the piggybacked substripes of the failed parity node.

### Algorithm 2: Repairing the Parity Nodes

1. **Read all data symbols** $\{f_{j+k}^{r(s-1+l)}\}_{j=1}^{k,l=1}$ of the $b$ piggybacked substripes and **calculate the parity symbols** $\{f_{k}^{r(s-1)+l}\}_{j=1}^{k,l=1}$ on the piggybacked substripes.

2. **Download the piggyback blocks** where the missing parity symbols and “black boxes” are embedded and the other symbols in the piggyback blocks except the missing parity symbols and “black boxes”. We can get the $s - b$ missing symbols by solving the piggyback functions.

3. **Reconstruct piggyback blocks** (which contain the parity symbols and “black boxes”) of the failed parity node, the remaining $b$ missing symbols are recovered completely.

The average repair bandwidth ratio of parity nodes $\eta_{D1}^{\text{par}}$ is

$$\eta_{D1}^{\text{par}} = \frac{1}{rks} (rkb + \sum_{j=1}^{br} m_j^2 + r(s - b))$$

$$= \frac{1}{rks} (rkb + r(s - b) + \frac{(\sum_{j=1}^{br} m_j)^2 + \sum_{j,k=1,j\neq k} (m_j - m_k)^2}{br})$$

$$= \frac{1}{rks} (rkb + r(s - b) + r^2(s - b)^2 + \sum_{j,k=1,j\neq k} (m_j - m_k)^2). \quad (14)$$

Define the following three variables,

$$v_l = \left[ \frac{r(s - b)}{br} \right], \quad v_u = \left[ \frac{r(s - b)}{br} \right], \quad v = r(s - b) - v_l br,$$

where $v_l$ and $v_u$ are the quotient and remainder of $r(s - b)/br$ respectively. By Eq. (14), we have

$$\eta_{D1}^{\text{par}} = \frac{1}{rks} (rkb + r(s - b) + \frac{r^2(s - b)^2}{br} + \frac{v(br - v)}{br}). \quad (16)$$

We obtain a lower bound and an upper bound of $\eta_{D1}^{\text{par}}$ as follows.

- If $r(s - b)$ is a multiple of $br$, i.e., $v = 0$. By Eq. (16), we have

$$\eta_{D1}^{\text{par}} \geq \frac{1}{rks} (rkb + r(s - b) + \frac{r^2(s - b)^2}{br})$$

$$= \frac{s - b - (s - b)^2}{s - b - (s - b)^2} \quad \text{for } b \cdot r \geq \frac{br}{s} + \frac{(s - b)^2}{ksb}. \quad (17)$$
Define \( p_b = b/s \) being the proportion of piggybacked substripes. It holds that

\[
\eta_{D1}^{\text{par}} \geq p_b + \frac{1 - p_b}{k} + \frac{(1 - p_b)^2}{p_b}.
\]

(18)

Hence, a lower bound of \( \eta_{D1}^{\text{par}} \) for the double-layered piggybacking codes is

\[
\Gamma_{\text{low}}(\eta_{D1}^{\text{par}}) = p_b + \frac{1 - p_b}{k} + \frac{(1 - p_b)^2}{p_b}.
\]

(19)

Obviously, \( \Gamma_{\text{low}}(\eta_{D1}^{\text{par}}) \) is a function of \( p_b \) and the number of systematic nodes \( k \).

- If \( r(s - b) \) is not a multiple of \( br \), i.e., \( v \neq 0 \), according to the inequality of arithmetic and geometric means,

\[
v(br - v) \leq \frac{(br)^2}{4}.
\]

(20)

Consequently, by Eq. (16), we can get

\[
\eta_{D1}^{\text{par}} \leq \frac{1}{rks} \left( rkb + r(s - b) + \frac{r^2(s - b)^2}{br} + \frac{br}{4} \right) = p_b(1 + \frac{1}{4k}) + \frac{1 - p_b}{k} + \frac{(1 - p_b)^2}{p_b}.
\]

(22)

As \( p_b = b/s \), it holds that

\[
\eta_{D1}^{\text{par}} \leq p_b(1 + \frac{1}{4k}) + \frac{1 - p_b}{k} + \frac{(1 - p_b)^2}{p_b}.
\]

(23)

which is a function of \( p_b \) and \( k \).

The curves of \( \Gamma_{\text{low}}(\eta_{D1}^{\text{par}}) \) and \( \Gamma_{\text{up}}(\eta_{D1}^{\text{par}}) \) with different \( p_b \) for \( k \) equals to 5, 10, 20, 100 and 1000 respectively are illustrated in Fig. 7.

The results of Fig. 7 suggest that the curves of \( \Gamma_{\text{low}}(\eta_{D1}^{\text{par}}) \) and \( \Gamma_{\text{up}}(\eta_{D1}^{\text{par}}) \) are close to each other with the same \( k \). Furthermore, the more the number of the systematic nodes \( k \), the closer the two bounds \( \Gamma_{\text{low}}(\eta_{D1}^{\text{par}}) \) and \( \Gamma_{\text{up}}(\eta_{D1}^{\text{par}}) \) are. Additionally, both of them can reach to an extreme point at appropriate \( p_b \). Being a value between \( \Gamma_{\text{low}}(\eta_{D1}^{\text{par}}) \) and \( \Gamma_{\text{up}}(\eta_{D1}^{\text{par}}) \), the minimum \( \eta_{D1}^{\text{par}} \) can be obtained via optimizing \( p_b \).

3) ANALYSES OF REPAIRING ALL KINDS OF NODES

In this subsection, the extreme values of \( \Gamma_{\text{low}}(\eta_{D1}^{\text{sys}}) \) and \( \Gamma_{\text{up}}(\eta_{D1}^{\text{sys}}) \) are obtained via optimizing \( p_a \). Subsequently, the extreme values of \( \Gamma_{\text{low}}(\eta_{D1}^{\text{par}}) \) and \( \Gamma_{\text{up}}(\eta_{D1}^{\text{par}}) \) are gotten via optimizing \( p_b \). Finally, the optimal repair efficiency of the double-layered piggybacking framework is given, and the minimum average repair bandwidth ratio of all kinds of nodes (both the systematic nodes and the parity nodes) \( \eta_{D1}^{\text{all}} \) based on this framework is obtained.

The partial derivatives of \( \Gamma_{\text{low}}(\eta_{D1}^{\text{sys}}) \) and \( \Gamma_{\text{up}}(\eta_{D1}^{\text{sys}}) \) with respect to \( p_a \) are as follows,

\[
d \Gamma_{\text{low}}(\eta_{D1}^{\text{sys}}) \left[ d p_a \right] = 1 + \frac{(p_a^2 - 1)}{p_a^2} \cdot \frac{1}{r - 1}.
\]

(24)

\[
d \Gamma_{\text{up}}(\eta_{D1}^{\text{sys}}) \left[ d p_a \right] = 1 + \frac{r - 1}{4k^2} + \frac{(p_a^2 - 1)}{p_a^2} \cdot \frac{1}{r - 1}.
\]

(25)

It can be obtained that \( p_a = \frac{\sqrt{r}}{r} \) and \( p_a = \frac{1}{\sqrt{r + (r - 1)^2/4k^2}} \) when (24) and (25) equal 0, respectively. Being convex functions about \( p_a \), the \( \min(\Gamma_{\text{low}}(\eta_{D1}^{\text{sys}})) \) and \( \min(\Gamma_{\text{up}}(\eta_{D1}^{\text{sys}})) \) are as follows,

\[
\min(\Gamma_{\text{low}}(\eta_{D1}^{\text{sys}})) = \frac{\sqrt{r}}{r} + \frac{1}{r - 1} \cdot \frac{(1 - \frac{\sqrt{r}}{r})^2}{\sqrt{r}} = \frac{2}{\sqrt{r} + 1}.
\]

(26)

\[
\min(\Gamma_{\text{up}}(\eta_{D1}^{\text{sys}})) = \frac{1}{\sqrt{r + (r - 1)^2/4k^2}} \cdot \left( 1 + \frac{r - 1}{4k^2} \right) + \frac{1}{r - 1} \cdot \left( 1 - \frac{1}{\sqrt{r + (r - 1)^2/4k^2}} \right) = 2 \frac{2\sqrt{r + (r - 1)^2/4k^2} - 2}{r - 1}.
\]

(27)

The curves of \( \min(\Gamma_{\text{low}}(\eta_{D1}^{\text{sys}})) \) and \( \min(\Gamma_{\text{up}}(\eta_{D1}^{\text{sys}})) \) with different \( r \) for code rate equals to 0.4, 0.5, 0.6 and 0.8 respectively are illustrated in Fig. 8.
The curves of \( \min(\Gamma_{low}(\eta_{D1}^{par})) \) and \( \min(\Gamma_{up}(\eta_{D1}^{par})) \) with different \( k \) are shown in Fig. 9.

- The \( \min(\Gamma_{low}(\eta_{D1}^{par})) \) and \( \min(\Gamma_{up}(\eta_{D1}^{par})) \) are dominated by the number of systematic nodes \( k \) only with no relationship to the code rate.
- The curves of \( \min(\Gamma_{low}(\eta_{D1}^{par})) \) and \( \min(\Gamma_{up}(\eta_{D1}^{par})) \) are closer to each other with increasing \( k \). There exists a double-layered piggybacking code whose repair bandwidth ratio of parity nodes is extremely close to \( \min(\Gamma_{low}(\eta_{D1}^{par})) \).

Obviously, \( \min(\Gamma_{low}(\eta_{D1}^{sys})) \) and \( \min(\Gamma_{low}(\eta_{D1}^{par})) \) are irrelevant. The minimum average repair bandwidth ratio of all nodes \( \min(\Gamma_{low}(\eta_{D1}^{all})) \) is

\[
\min(\Gamma_{low}(\eta_{D1}^{all})) = \frac{k}{k + r} \min(\Gamma_{low}(\eta_{D1}^{sys}))) + \frac{r}{k + r} \min(\Gamma_{low}(\eta_{D1}^{par})).
\]

By Eq. (26) and Eq. (30), we can get

\[
\min(\Gamma_{low}(\eta_{D1}^{all})) = \frac{k}{k + r} \left( \frac{2}{\sqrt{r + 1}} \right) + \frac{r}{k + r} \left( \frac{2 \sqrt{k} - 1}{k} \right),
\]

for \( p_a = \frac{\sqrt{k}}{r} \) and \( p_b = \frac{\sqrt{k}}{r} \).

The asymptotic analyses of \( \min(\Gamma_{low}(\eta_{D1}^{all})) \) are obtained as shown in (34), (35), and (36), as shown at the bottom of the next page. Consequently, we can get the following conclusion.

- When \( k \), the number of systematic nodes, tends to infinity, the \( \min(\Gamma_{low}(\eta_{D1}^{all})) \) approaches to a constant, which is related to \( r \).
- When \( r \), the number of parity nodes, tends to infinity, the \( \min(\Gamma_{low}(\eta_{D1}^{all})) \) approaches to a constant, which is related to \( k \).
- When both \( k \) and \( r \) tend to infinity, the \( \min(\Gamma_{low}(\eta_{D1}^{all})) \) approaches to zero.

The variation of \( \min(\Gamma_{low}(\eta_{D1}^{all})) \) vs \( k \) and \( r \) is shown in Fig. 10. Obviously, when \( k \) and \( r \) continue increasing,
\[ \min(\Gamma_{low}(\eta_{D1}^{all})) \] of the proposed double-layered piggybacking framework approaches to zero infinitely. The repair efficiency is higher with lower bandwidth ratio. In particular, our design is very suitable for large-scale distributed storage systems.

IV. THE IMPROVED DOUBLE-LAYERED PIGGYBACKING FRAMEWORK

In this section, the improved double-layered piggybacking framework is proposed. Based on the double-layered piggybacking framework presented in the former section, the construction of the piggyback functions for repairing systematic nodes is changed. Compared with the former framework, the improved method achieves a lower repair bandwidth ratio of systematic nodes when the number of parity nodes is over 3. Consequently, for the whole distributed storage system, the repair bandwidth ratio is lowered for all kinds of nodes, which is equivalent to a higher comprehensive repair efficiency. We introduce the improved method via a simple example firstly, and then we generalize it.

A. THE IMPROVED DOUBLE-LAYERED PIGGYBACKED (13, 6) MDS CODE

Using a (13, 6) systematic MDS code as the base code, the improved double-layered piggybacked (13, 6) MDS code with 7 substripes is depicted in Fig. 11. Each systematic node \( j(1 \leq j \leq 6) \) stores the data symbols \( f_j^{(0)} \), and each parity node \( k+r(k = 6, 1 \leq r \leq 7) \) stores the parity symbols \( f_{k+r}^{(1)} \), \( (1 \leq r \leq 7) \).

1) CODE DESIGN

Initially, all the systematic nodes in the piggybacking framework are divided into three subsets, i.e., \( \Phi_1 = \{1, 2\} \), \( \Phi_2 = \{3, 4\} \), and \( \Phi_3 = \{5, 6\} \). Subsequently, these 3 subsets of data symbols are embedded into the reserved substripes 5, 4, and 3, respectively. For the 5th substripe, the data symbols in the first subset \( \Phi_1 \) except the ones in the 5th substripe are embedded. For the 4th substripe, the data symbols in the second subset \( \Phi_2 \) except the ones in the 5th and 4th substripes are embedded. For the 3rd substripe, the data symbols in the third subset \( \Phi_3 \) except the ones in the 5th, 4th, and 3rd substripes are embedded. The key idea of this construction is dividing the systematic nodes into several subsets and the piggyback blocks (black boxes) in a substripe embedding part of the data symbols in the same subset. Moreover, the data symbols in a substripe of a subset are embedded into the same black box. The diagonal-order method (refer to sections III) is adopted to repair parity nodes in this improved double-layered piggybacking framework too.

2) REPAIR ANALYSES

The amount of data read and downloaded is different when repairing failed nodes in different subsets because of the different number of data symbols embedded in piggyback blocks in different subsets. Taking one node of each \( \Phi_i (1 \leq i \leq 3) \) as example, the repair process is as follows.

- Suppose that the first systematic node fails. The data symbols \( f_1^{(5)} \) and parity symbol \( f_{k+1}^{(5)} \) are read to recover the missing data symbol \( f_1^{(5)} \) via the MDS code decoding. The amount of data read in this step is 6. \( f_1^{(5)} \) being obtained, the parity symbols \( f_{k+j}^{(5)} \) \( (j=2, 3) \) can be calculated. Then, the piggyback blocks on substripe 5 and the data symbols \( f_2^{(5)} \) \( (i=1, i \neq 5) \) in the same subset are read, and we can obtain the remaining missing data symbols \( f_{1, i}^{(5)} \) \( (i=1, i \neq 5) \) by solving the piggyback functions. The amount of data read in this step is 12.

Hence, the total amount of data read and downloaded is 18. The repair strategy of systematic node with index 2 is similar.

- Suppose that the third systematic node fails. The data symbols \( f_3^{(4)} \) \( (j=1, 2) \) and parity symbols \( f_{k+1}^{(4)} \) \( (j=1, 2) \) are read to recover the missing data symbol \( f_3^{(4)} \) \( (j=1, 2) \) via the MDS code decoding. The amount of data read in this step is 12. \( f_3^{(4)} \) being obtained,

\[
\lim_{k \to \infty, r \to \infty} \min(\Gamma_{low}(\eta_{D1}^{all})) = \lim_{k \to \infty, r \to \infty} \frac{k}{k+r}(\frac{2}{\sqrt{r}+1}) + \frac{r}{k+r}(\frac{2\sqrt{K} - 1}{K}) = \frac{2}{\sqrt{r}+1}. 
\]

\[
\lim_{r \to \infty} \min(\Gamma_{low}(\eta_{D1}^{all})) = \lim_{r \to \infty} \frac{k}{k+r}(\frac{2}{\sqrt{r}+1}) + \frac{r}{k+r}(\frac{2\sqrt{K} - 1}{K}) = \frac{2\sqrt{K} - 1}{K}. 
\]

\[
\lim_{k \to \infty, r \to \infty} \min(\Gamma_{low}(\eta_{D1}^{all})) = \lim_{k \to \infty, r \to \infty} \frac{k}{k+r}(\frac{2}{\sqrt{r}+1}) + \frac{r}{k+r}(\frac{2\sqrt{K} - 1}{K}) = 0. 
\]
repairing parity nodes is

Using a (k + 1) piggybacking framework

Suppose that the fifth systematic node fails. The data strategy of systematic node with index 6 is similar.

The amount of data read and downloaded is 26. The repair

Next, the piggyback blocks on the third substripe and

The amount of data read in this step is 18.

The amount of data read in this step is 10. Hence,

f_{1}^{(3)} \{s \neq \ell \}

f_{6}^{(i)} \{s \neq \ell \}

The number of systematic nodes in subset \( \Phi_{\ell} \) is \( \psi_{\ell} \). The improved general double-layered piggybacking framework we proposed is shown in Fig.12. \( \{p_{1}^{(k-b+i+1)}\}_{i=1,j=1}^{L} \) are the piggyback blocks embedding data symbols to repair systematic nodes. \( \{p_{1}^{(k-b+i+1)}\}_{i=1,j=1}^{L} \) are the piggyback blocks embedding parity symbols and “black boxes” to repair parity nodes. The embedding method of the first-layer piggybacking framework is the same as the methods in the (13, 6) code given in this section. and the second-layer piggybacking framework adopt the diagonal-order method.

In the previous discussion, the systematic nodes in the same subset have the same repair bandwidth, and the systematic nodes in different subsets have different repair bandwidth. Suppose the systematic node \( i \) in the subset \( \Phi_{\ell} \) (1 ≤ \( \ell \) ≤ L) fails, the missing data symbols need to be recovered are \( f_{i}^{(u)} \{s \neq \ell \} \).

Initially, the \( \ell \) missing data symbols which are not embedded in piggyback blocks can be recovered via the MDS code decoding according to the structure, and the parity symbols \( f_{k+b+1}^{(s-b-\ell+1)} \{2 \leq t \leq r \} \) of the substripe with index \( s-b-\ell+1 \) can be calculated too. The amount of data read and downloaded in this step is \( k \ell \).

\[ \sum_{\ell=1}^{L} \psi_{\ell} = k \]

B. THE GENERAL IMPROVED DOUBLE-LAYERED PIGGYBACKING FRAMEWORK

Using a (k + r, k) systematic MDS code as the base code to form \( s \) substripes. The number of piggybacked substripes for repairing parity nodes is \( b \). All systematic nodes are divided into \( L \) subsets \( \{\Phi_{1}, \Phi_{2}, \cdots, \Phi_{L}\} \), the number of systematic nodes in subset \( \Phi_{\ell} \) (1 ≤ \( \ell \) ≤ L) is \( \psi_{\ell} \).
For simple analysis, let the number of systematic nodes in node \( \eta \) be \( k \), and the number of piggyback blocks \( \gamma \) be \( \gamma \). Consequently, the total amount of data read and downloaded in this step is \( (s - t) \), where \( s, t \in \Phi_i, t \neq i \). The amount of data read and downloaded to repair node \( i \) is \( k \ell + (s - \ell) \psi \), where \( \ell \) is the subset with node \( i \), where \( i, t \in \Phi_i, t \neq i \). The amount of data read and downloaded to repair node \( i \) is \( k \ell + (s - \ell) \psi \).

The total repair bandwidth \( \gamma D_2 \) for all the \( k \) systematic nodes is

\[
\gamma D_2 = \sum_{\ell=1}^{L} (k \ell + (s - \ell) \psi) \psi \ell.
\]

(37)

Thus the average repair bandwidth ratio of systematic node, \( \eta D_2 \), is

\[
\eta D_2 = \frac{1}{k^2} \sum_{\ell=1}^{L} (k \ell + (s - \ell) \psi) \psi \ell.
\]

For simple analysis, let the number of systematic nodes in each subset be as equal as possible. Define the following three variables,

\[
w_\ell = \left\lfloor \frac{k}{L} \right\rfloor, \quad w_u = \left\lceil \frac{k}{L} \right\rceil, \quad w = k - w_u L,
\]

where \( w_\ell \) and \( w_u \) are the quotient and remainder of \( k/L \) respectively. By Eq. (38), we have

\[
\eta D_2 = \frac{1}{k^2} \left( \sum_{\ell=1}^{L} w_\ell \ell \psi \ell \right) + \frac{1}{k^2} \sum_{\ell=1}^{L} \ell \psi \ell (k - \psi \ell).
\]

(40)

\[
\sum_{\ell=1}^{L} w_\ell \ell \psi \ell = \frac{k^2}{L} \left( \sum_{\ell=1}^{L} (w_\ell - \ell) \psi \ell \right) + \frac{1}{k^2} \sum_{\ell=1}^{L} \ell \psi \ell (k - \psi \ell).
\]

\[
= \frac{1}{k^2} \left( k^2 + w(L - w) \right).
\]

(41)

\[
\sum_{\ell=1}^{L} \ell \psi \ell (k - \psi \ell) = \frac{1}{k^2} \sum_{\ell=1}^{L} \ell w_u (k - w_u)
\]

+ \frac{1}{k^2} \sum_{\ell=1}^{L} \ell w_u (k - w_u)

Hence, the average repair bandwidth ratio of systematic nodes, \( \eta D_2 \), is given in (43), as shown at the bottom of the next page. A lower bound and an upper bound of \( \eta D_2 \) are presented as following.

- From Eq. (43), it can be shown that \( \eta D_2 \) has the smallest value when \( w = 0 \), which means that \( k \) is a multiple of \( L \) and \( w_\ell = w_u = \frac{k}{L} \).

\[
\eta D_2 \geq \frac{L}{2s} + \frac{1}{L} \left( 1 - \frac{1}{2} \right).
\]

(44)

The right side of the inequation (44) can be a lower bound of \( \eta D_2 \) for the improved double-layered piggybacking codes.

- If \( k \) is not a multiple of \( L \), i.e., \( w \neq 0 \). According to the inequality of arithmetic and geometric means, we have

\[
w(L - w) \leq \frac{(w + L - w)^2}{4} = \frac{L^2}{4}.
\]

(45)

\[
w_u (k - w_u) \leq \frac{k^2}{4}.
\]

(46)

\[
w_\ell (k - w_\ell) \leq \frac{k^2}{4}.
\]

(47)

\[
w(1 + w) \leq \frac{(2w + 1)^2}{4}.
\]

(48)

\[
(w + L + 1) - (L - w) \leq \frac{(2L + 1)^2}{4}.
\]

(49)

Consequently, by Eq. (43), we obtain

\[
\eta D_2 \leq \frac{1}{L} + \frac{L}{4k^2} + \frac{(2w + 1)^2 + (2L + 1)^2}{32s}.
\]

(50)

Hence, a upper bound of \( \eta D_2 \) for the improved double-layered piggybacking codes is

\[
\Gamma_{up}(\eta D_2) = \frac{1}{L} + \frac{L}{4k^2} + \frac{(2w + 1)^2 + (2L + 1)^2}{32s}.
\]

(51)
Suppose that the code rate of the base code is 0.5, \( r = s \) (In the new framework, it is required that \( r \geq s \)). The curves of \( \Gamma_{low}(\eta_{D2}^{sys}) \) and \( \Gamma_{up}(\eta_{D2}^{sys}) \) vs \( L \) for different \( s \) (50, 100, 200, 500 and 1000) are illustrated in Fig. 13.

The results from Fig. 13 suggest that the curves of \( \Gamma_{low}(\eta_{D2}^{sys}) \) are smooth overall and they can reach to an extreme point when \( (52) = 0 \), \( L = \sqrt{2s} - 1 \). Then, \( \min(\Gamma_{low}(\eta_{D2}^{sys})) \) is a decreasing function with \( s \) and the improved framework requires \( s \leq r \). In order to obtain the minimum of \( \Gamma_{low}(\eta_{D2}^{sys}) \), we set \( s = r \). Hence, Eq. (54) can be described as Eq. (55).

\[
\min(\Gamma_{low}(\eta_{D2}^{sys})) = \frac{\sqrt{2r - 1}}{r}. \tag{55}
\]

The curve of \( \min(\Gamma_{low}(\eta_{D2}^{sys})) \) with different \( r \) is shown in Fig. 14.

The following are a few notes from Fig. 14,

- The \( \min(\Gamma_{low}(\eta_{D2}^{sys})) \) is dominated by the number of parity nodes \( r \) only, with no relationship to the code rate.
- The \( \min(\Gamma_{low}(\eta_{D2}^{sys})) \) is decreasing as \( r \) increases.

Because we do not change the construction of piggyback blocks designed for parity nodes, so \( \eta_{D2}^{par} = \eta_{D1}^{par} \). According to the analyses above, it is easy to know that \( \min(\Gamma_{low}(\eta_{D2}^{sys})) \) and \( \min(\Gamma_{low}(\eta_{D2}^{par})) \) are irrelevant. Thus, the minimum average repair bandwidth ratio of all nodes \( \min(\Gamma_{low}(\eta_{D2}^{all})) \) is

\[
\min(\Gamma_{low}(\eta_{D2}^{all})) = \frac{k}{r(k + r)} \min(\Gamma_{low}(\eta_{D2}^{sys})) + \frac{r}{k + r} \min(\Gamma_{low}(\eta_{D2}^{par})). \tag{56}
\]

By Eq. (55) and Eq. (30), we can get

\[
\min(\Gamma_{low}(\eta_{D2}^{all})) = \frac{k}{k + r} \left( \frac{2r - 1}{r} \right) + \frac{r}{k + r} \left( \frac{2\sqrt{k} - 1}{k} \right), \tag{57}
\]

for \( L = \sqrt{2r - 1} \) and \( p_b = \frac{\sqrt{k}}{r} \).

At the end of this subsection, the asymptotic analyses of \( \min(\Gamma_{low}(\eta_{D2}^{all})) \) are obtained as shown in (58), (59), and (60), as shown at the bottom of the next page. Consequently, we can get the following conclusion.

\[
\eta_{D2}^{sys} = \frac{1}{k^2} \left( \frac{k^2 + w(L - w)}{L} \right) + \frac{1}{k^2} \left( \frac{w(w(k - w_u)(1 + w) + w(L + 1)(k - wu)(L - w))}{2} \right). \tag{43}
\]
When the $k$ tends to infinity, the $\min(\Gamma_{low}(t_{all}^{(r1)}))$ approaches to a constant, which is related to $r$.

When the $r$ tends to infinity, the $\min(\Gamma_{low}(t_{all}^{(r2)}))$ approaches to a constant, which is related to $k$.

When both the $k$ and the $r$ tend to infinity, the $\min(\Gamma_{low}(t_{all}^{(r2)}))$ approaches to zero.

V. DECODING COMPLEXITY ANALYSES

In this section, we give some analyses on the decoding complexity of the proposed two double-layered piggybacking codes.

A. DECODING COMPLEXITY FOR THE FIRST DOUBLE-LAYERED PIGGYBACKING CODE

1) COMPLEXITY FOR REPAIRING A SYSTEMATIC NODE

As the statement in Section III, piggybacking codes, adopt two kinds of calculations to repair a failed systematic node, i.e., the MDS code decoding and solving piggyback functions. Recall Fig.5 in Section III. Assume that the systematic node with index $i$ fails. We have Eq.(61), as shown at the bottom of this page.

Firstly, the data symbol $f_i^{(s-b)}$ of the systematic vector $f_{s-b}$ on the substripe $s - b$ is repaired via the MDS code decoding. The symbols $f_1^{(s-b)}, \ldots, f_{i-1}^{(s-b)}, f_{i+1}^{(s-b)}, \ldots, f_k^{(s-b)}$ are required. Distributed storage systems usually take the form of systematic codes, thus the vector representation of $a_{s-b}$ can be described as $(f_1^{(s-b)}, \ldots, f_{i-1}^{(s-b)}, b_{i+1}^{(s-b)}, \ldots, f_k^{(s-b)})$. Denote the vector representation of $a_{s-b}$ as $(a_{s-b,1}, \ldots, a_{s-b,k})$ and the vector representation of $f_{i+1}^{(s-b)}$ as $(b_{i+1,1}, \ldots, b_{i+1,k})$. Then, $f_i^{(s-b)}$ can be worked out by Eq.(61).

Obviously, the MDS code decoding for the recovery of $f_i^{(s-b)}$ costs $k$ multiplications and $k - 1$ additions. In this way, there are $a$ missing symbols similar to the repair operation in Fig.5. Hence, $ak$ multiplications and $a(k - 1)$ additions are cost to repair them.

Additionally, the data symbol $f_i^{(1)}$ of the systematic vector $a_1$ on the substripe 1 is repaired via solving piggyback functions. $f_i^{(1)}$ can be figured out as follows,

- Compute the parity symbol $f_{k+1}^{(s-b-a+1)}$ in the “black box” where the piggyback block embedded the data symbols. This step costs $k$ multiplications and $k - 1$ additions;

- Subtract the parity symbol from the “black box” to get the piggyback block. Thus, one addition is needed;

- Subtract other surviving data symbols from the piggyback block. Thus, $x - 1$ additions are required. Actually, $x$ is the average number of data symbols embedded in a piggyback block, i.e., $x = \frac{s - b}{(r - 1)a}$.

In this way, there are $s - a$ symbols similar to the repair operation in Fig.(5). Hence, $(s - a)k$ multiplications and $(s - a)(x + k - 1)$ additions are cost.

In summary, $sk$ multiplications and $\frac{s - a - 2k}{2} + s(k - 1)$ additions are cost for repairing a systematic node.

2) COMPLEXITY FOR REPAIRING A PARITY NODE

For simplicity, recall Fig. 4 in Section III. Assume that the parity node with index $k + 2$ fails. First of all, $f_1^{(1)}$ can be worked out as follows,

- Compute the parity symbol $f_{k+1}^{(5)}$ in the “black box” where the piggyback block embedded the missing symbols. This step costs $k$ multiplications and $k - 1$ additions;

- Subtract the parity symbol from the “black box” to get the piggyback block. Thus, one addition is needed;

- Subtract other surviving parity symbols from the piggyback block. Thus, $y - 1$ additions are required. Actually, $y$ is the average number of parity symbols embedded in a piggyback block, i.e., $y = \frac{s - b}{r}$.

In this way, there are $s - b$ missing symbols similar to the repair operation in Fig. 5. Hence, $(s - b)k$ multiplications and $(s - b)(y + k + 1)$ additions are cost.

Secondly, the missing symbol $f_{k+2}^{(5)} + f_{k+3}^{(1)} + f_{k+4}^{(2)}$ can be figured out as follows,

- Compute the parity symbol $f_{k+2}^{(5)}$. This step costs $k$ multiplications and $k - 1$ additions;

- Sum the parity symbols $f_{k+2}^{(5)}, f_{k+3}^{(1)}$ and $f_{k+4}^{(2)}$. In this step, $y$ additions are needed;

In this way, there are $b$ missing symbols similar to the repair operation in Fig. 5. Hence, $bk$ multiplications and $b(y + k + 1)$ additions are cost.

\[
\lim_{k \to \infty, r \to \infty} \min(\Gamma_{low}(t_{all}^{(all)})) = \lim_{k \to \infty} \frac{2 \sqrt{r - 1} - \sqrt{r}}{k + r} \left( \frac{2 \sqrt{1 - k^2}}{k} + \frac{2 \sqrt{1 - k^2}}{r} \right) = \frac{2 \sqrt{r - 1} - \sqrt{r}}{k + r}. \tag{58}
\]

\[
\lim_{r \to \infty} \min(\Gamma_{low}(t_{all}^{(r2)})) = \frac{2 \sqrt{r - 1}}{k + r} \left( \frac{2 \sqrt{r - 1} - \sqrt{r}}{k} + \frac{2 \sqrt{r - 1} - \sqrt{r}}{r} \right) = \frac{2 \sqrt{r - 1}}{k + r}. \tag{59}
\]

\[
\lim_{k \to \infty, r \to \infty} \min(\Gamma_{low}(t_{all}^{(r2)})) = \lim_{k \to \infty} \frac{2 \sqrt{r - 1}}{k + r} \left( \frac{2 \sqrt{r - 1} - \sqrt{r}}{k} + \frac{2 \sqrt{r - 1} - \sqrt{r}}{r} \right) = 0. \tag{60}
\]

\[
f_{s-b}^{(s-b)} = a_{s-b,i} \begin{pmatrix}
f_{k+1}^{(s-b,1)} - (a_{s-b,1} f_{k+1,1} + \ldots + a_{s-b,i} f_{k+1,i} + a_{s-b,k} f_{k+1,k}) 
\end{pmatrix} \tag{61}
\]
In summary, \( sk \) multiplications and \( s(\frac{s-k}{b} + k - 1) \) additions are cost to repair a parity node.

### B. Decoding Complexity for the Improved Double-Layered Piggybacking Codes

1) Complexity for Repairing a Systematic Node

For simplicity, recall Fig. 11 in Section IV. Suppose that the systematic node with index 3 fails and it is in the subset \( \Phi_l (1 \leq l \leq L) \).

Firstly, the missing symbol \( f_{3}^{(4)} \) is repaired via the MDS code decoding. As previously analyzed, the MDS code decoding for the recovery of \( f_{3}^{(4)} \) costs \( k \) multiplications and \( k - 1 \) additions. In this way, there are \( l \) missing symbols similar to the repair operation in Fig. 12. Hence, \( kl \) multiplications and \( (k - 1)l \) additions are cost to repair them.

Secondly, \( f_{3}^{(1)} \) can be figured out as follows,

- Compute the parity symbol \( f_{k}^{(4)} \). This step costs \( k \) multiplications and \( k - 2 \) additions;
- Subtract the parity symbol from the “black box” to get the piggyback block. Thus, one addition is needed;
- Subtract other surviving data symbols from the piggyback block. Thus, \( \Phi_i - 1 \) additions are required. Actually, \( \Phi_i \) is the average number of systematic nodes in a subset, i.e., \( \Phi_i = \frac{k}{l} \).

In this way, there are \( s - l \) missing symbols similar to the repair operation in Fig. 12. Hence, \( k(s - l) \) multiplications and \( (\Phi_i + k - l)(s - l) \) additions are cost.

In summary, \( sk \) multiplications and \( k(s - l) + s(k - 1) \) additions are cost to repair a systematic node. When Eq.(52) equals 0, \( L = \sqrt{2s - 1} \), then we can get \( \min (\Gamma_{low}(\eta_{sys}^{low})) \). So, the number of additions can be modified to \( \frac{k(s - l)}{\sqrt{2s - 1}} + s(k - 1) \).

2) Complexity for Repairing a Systematic Node

Because the construction of piggyback blocks designed for parity nodes remains unchanged, the complexity for repairing a parity node remains the same. \( sk \) multiplications and \( s(\frac{s-k}{b} + k - 1) \) additions are cost to repair a parity node.

### VI. Comparison with Some Existing Piggybacking Codes

In this section, the proposed piggybacking codes are compared with some existing piggybacking codes, including the three piggybacking schemes proposed by Rashmi et al. [24], [37], abbreviated as R-PB-1, R-PB-2, and R-PB-3 respectively, as well as the generalized piggybacking codes presented by Yuan et al. [27], called G-PB. Moreover, the first double-layered piggybacking code we proposed in the comparison is abbreviated as D-PB-1, and the second (the improved) is D-PB-2.

In Table 1, the requirements for the numbers of parity nodes and substripes for different classes of piggybacking codes are given. In this table, \( m \) is a positive integer. R-PB-2 needs more parity nodes and more substripes. For R-PB-1 and R-PB-3, the number of substripes must be even. For G-PB, the choose of parameters are more flexible while it can only repair systematic nodes. In D-PB-1, we can just add one more substripe compared with G-PB so as to repair both the systematic nodes and parity nodes. The number of substripes in D-PB-2 is no larger than \( r \), it obtains the lowest repair bandwidth ratio.

In Fig.15, the repair bandwidth ratio for different piggybacking codes with \((n,k)\) is shown. The piggybacking codes have the property of repair-by-transfer [38], i.e., the system only read those symbols that need to be downloaded. Thus the amount of data read is equal to the amount of data downloaded. The \( \eta_{sys} \), \( \eta_{par} \), and \( \eta_{all} \) of different piggybacking codes with \((n,k)\) are given in Fig.15(a), Fig.15(b), and Fig.15(c) respectively.

As shown in Table 2, for R-PB-1 and R-PB-3, the number of substripes is 8. For R-PB-2, it is \( 4(2r - 3) \). It is the values
of $k$ and $r$ that influence the optimal repair bandwidth ratio for G-PB, D-PB-1, so the number of substripes can be set randomly. We set the number of substripes 10 for G-PB and D-PB-1 simply. For D-PB-2 we set the number of substripes being the same as $r$ because of the construction. In addition, we also list the optimal substripes design parameters for the minimum repair bandwidth ratio in Table 2.

For the systematic nodes, the repair bandwidth ratio $\eta_{sys}$ of D-PB-1 we proposed is equal to that of G-PB. In particular, when the code parameters $n$ and $k$ increase enough, the repair bandwidth ratio $\eta_{sys}$ of D-PB-2 is the smallest. R-PB-3 and G-PB cannot repair the parity nodes. D-PB-1 and D-PB-2 we proposed have the smallest parity node repair bandwidth ratio. And when the code parameters $n$ and $k$ increase, the $\eta_{all}$ for D-PB-2 decreases largely. Thus compared with other piggybacking codes, the D-PB-2 we proposed has the optimal comprehensive repair performance, see Fig.15(c).

### VII. PIGGYBACKING CODES WITH MULTIPLE INSTANCES

Piggybacking codes are used in distributed storage systems, so the case that their constructions impact on the performance of storage systems cannot be ignored when the frameworks are designed. The requirement of a large number of substripes presents multiple challenges on the systems. The primary issue is that a large number of substripes results in a large number of fragmented data reading which is detrimental to the performance of disks [39]. The second issue is that numerous substripes also restrict the minimum size of files that can be handled by the code (and hence by the storage system) [37].

To the best of our knowledge, the Rotated-RS [25] codes, the (repair-optimized) EVENODD [16], [17], and the RDP codes [20], [21] are MDS, high-rate and have either a constant or linear (in $k$) number of substripes. However, they have fewer redundant nodes, which makes the fault-tolerant capability limited and the repair efficiency low. MSR codes are of particular interest to this paper since they are MDS. The Product-Matrix MSR codes [8] are explicit, practical MSR code constructions which have linear (in $k$) number of substripes. But the code rate of these codes is low. Cadambe et al. [40] show the existence of high-rate MSR codes as the number of substripes approaches infinity. The MSR constructions presented in [41] are high rate, but necessarily require the number of substripes to be exponential in $k$. Actually, it has been shown in [42], that an exponential number of substripes is a fundamental limitation of any high-rate MSR code optimizing both the amount of data read and downloaded [37].

The two kinds of piggybacking codes under the double-layered piggybacking framework we presented are both multiple substripes of an instance. we can achieve fewer substripes in each instance via embedding multiple instances one by one when the number of substripes is constant. Clearly, the parity symbols in the first two substripes do not participate in the repair of the systematic nodes as in Fig.4. These “useless” parity symbols are used to connect multiple instances. Firstly, sum up these parity symbols of $b$ piggybacked substripes corresponding to parity nodes respectively, then add these summing symbols to the first substripe (corresponding to the parity symbols of the parity nodes respectively) of the next instance.

As shown in Fig.16, piggyback blocks for repairing parity nodes in the instance 1 are embedded into the first substripe of instance 2, and these parity messages are transmitted to the last $b$ piggybacked substripes via encoding. Then, these parity messages will be transmitted into the instance 3 by embedding again. One can embed any number of instances in this way. Since the parity messages of the former instance are embedded into the “useless” substripe of the latter instance,

### TABLE 2. Comparison of the code parameters in piggybacking design framework for different explicit codes.

| Code Parameters: $(n, k)$ | R-PB-1 substripes | R-PB-2 substripes | R-PB-3 substripes | G-PB substripes | D-PB-1 substripes | D-PB-2 substripes | $L, b$ |
|--------------------------|-----------------|------------------|-----------------|----------------|----------------|----------------|------|
| (12, 10)                 | 8               | 4                | 8               | 10             | 3, 7           | 10             | 7, 3 |
| (15, 12)                 | 8               | 12               | 8               | 10             | 5, 5           | 10             | 6, 3 |
| (18, 14)                 | 8               | 20               | 8               | 10             | 5, 5           | 10             | 5, 3 |
| (20, 15)                 | 8               | 28               | 8               | 10             | 6, 4           | 10             | 5, 3 |
| (34, 28)                 | 8               | 36               | 8               | 10             | 6, 4           | 10             | 5, 2 |
| (45, 38)                 | 8               | 44               | 8               | 10             | 7, 3           | 10             | 4, 2 |
| (53, 45)                 | 8               | 52               | 8               | 10             | 7, 3           | 10             | 4, 2 |
| (59, 50)                 | 8               | 60               | 8               | 10             | 7, 3           | 10             | 4, 2 |
| (92, 80)                 | 8               | 84               | 8               | 10             | 8, 2           | 10             | 3, 2 |
| (105, 85)                | 8               | 148              | 8               | 10             | 8, 2           | 10             | 3, 2 |
| (120, 90)                | 8               | 228              | 8               | 10             | 9, 1           | 10             | 2, 2 |
the repair method of the systematic nodes is unchanged. And the repair bandwidth ratio are the same, regarding the new piggyback blocks after embedding as new symbols. Since the repair method of the parity nodes are unchanged, the repair bandwidth ratio remains the same. In addition, the way of the improved double-layered piggybacking codes with multiple instances is similar.

For the requirement of node repair, each instance is independent each other before being embedded and node repair methods of every instances remain unchanged after being embedded. So the double-layered piggybacking codes with multiple instances not only have the same repair performance compared to the ones with single instance, but also have a small number of substripes, which makes its application have more practical significance in distributed storage system. Suppose that we need to get specific symbols in one piggyback block, what we need to do is that encoding, constructing and embedding these instances from left to right (former to latter) in turn until the piggyback block is created.

VIII. CONCLUSION

In this paper, initially, we propose a double-layered piggybacking framework (called D-PB-1) to repair both systematic nodes and parity nodes effectively at the same time via optimizing the proportion of the reserved substripes $p_s$ and piggybacked substripes $p_b$. The repair bandwidth ratio for this scheme is reduced greatly. When the number of systematic nodes $k$ and parity nodes $r$ tends to infinity, the average repair bandwidth ratio approaches to zero. Subsequently, the improvement of the double-layered piggybacking framework (called D-PB-2) is proposed, which results in a smaller repair bandwidth ratio by changing the construction of the piggyback functions for systematic nodes. Compared with other piggybacking codes, the two classes of double-layered piggybacking codes we proposed not only have flexible requirements but also obtain the optimal comprehensive repair efficiency, especially for D-PB-2. These results not only have great practical significance to piggybacking codes but also have theoretical significance to repairing one type of the nodes and can be adopted in more other distributed storage systems.

In our future work, we will try to maintain the performance of the piggybacking scheme while making the complexity lower and promote the applications in actual systems.

ACKNOWLEDGMENT

The authors would like to thank the anonymous reviewers for their reasonable suggestions.

REFERENCES

[1] V. W. S. Wong, Key Technologies for 5G Wireless Systems. Cambridge, U.K.: Cambridge Univ. Press, 2017.
[2] D. Ford, F. I. Popovici, M. Stokely, V. A. Truong, L. Barroso, C. Grimes, and S. Quinlan, “Availability in globally distributed storage systems,” in Proc. Usenix Conf. Oper. Syst. Design Implement., 2010, pp. 61–74.
[3] B. Schroeder and G. A. Gibson, “Understanding disk failure rates: What does an MTTF of 1,000,000 hours mean to you?” ACM Trans. Storage, vol. 3, no. 3, p. 8, Oct. 2007.
[4] J. Li and B. Li, “Erasure coding for cloud storage systems: A survey,” Tsinghua Sci. Technol., vol. 18, no. 3, pp. 259–272, Jun. 2013.
[5] B. Calder, J. Wang, A. Ogos, N. Nilakantan, A. Skjolsvold, S. Mickelvie, Y. Xu, S. Srivastav, J. Wu, and H. Simitci, “Windows azure storage: A highly available cloud storage service with strong consistency,” in Proc. ACM Symp. Oper. Syst. Prac., 2011, pp. 143–157.
[6] J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherpoon, W. Weimer, C. Wells, and B. Zhao, “OceanStore: An architecture for global-scale persistent storage,” ACM SIGARCH Comput. Archit. News, vol. 28, no. 5, pp. 190–201, Dec. 2000.
[7] A. G. Dimakis, P. B. Godfrey, Y. Wu, M. J. Wainwright, and K. Ramchandran, “Network coding for distributed storage systems,” IEEE Trans. Inf. Theory, vol. 56, no. 9, pp. 4539–4551, Sep. 2010.
[8] K. V. Rashmi, N. B. Shah, and P. V. Kumar, “Optimal exact-reconstructing codes for distributed storage at the MSR and MBR points via a product-matrix construction,” IEEE Trans. Inf. Theory, vol. 57, no. 8, pp. 5227–5239, Aug. 2011.
[9] N. B. Shah, K. V. Rashmi, P. V. Kumar, and K. Ramchandran, “Interference alignment in regenerating codes for distributed storage: Necessity and code constructions,” IEEE Trans. Inf. Theory, vol. 58, no. 4, pp. 2134–2158, Apr. 2012.
[10] K. V. Rashmi, N. B. Shah, P. V. Kumar, and K. Ramchandran, “Explicit construction of optimal exact reconstructing codes for distributed storage,” in Proc. 47th Annu. Allerton Conf. Commun., Control, Comput. (Allerton), Sep. 2009, pp. 1243–1249.
[11] B. Zhu, K. W. Shum, and H. Li, “Fractional repetition codes with optimal reconstruction degree,” IEEE Trans. Inf. Theory, vol. 66, no. 2, pp. 983–994, Feb. 2020.
[12] D. S. Papailiopoulos and A. G. Dimakis, “Locally repairable codes,” in Proc. IEEE Int. Symp. Inf. Theory Proc., Jul. 2012, pp. 2771–2775.
[13] M. Shahabinejad, M. Khabbazian, and M. Ardakani, “An efficient binary locally repairable code for Hadoop distributed file system,” IEEE Commun. Lett., vol. 18, no. 8, pp. 1287–1290, Aug. 2014.
[14] D. S. Papailiopoulos, J. Luo, A. G. Dimakis, C. Huang, and J. Li, “Simple regenerating codes: Network coding for cloud storage,” in Proc. IEEE INFOCOM, Mar. 2012, pp. 2801–2805.
[15] Y.-S. Su, “On constructions of a class of binary locally repairable codes with multiple repair groups,” IEEE Access, vol. 5, pp. 3524–3528, 2017.
[16] M. Blaum, J. Brady, J. Bruck, and J. Menon, “EVENODD: An efficient scheme for tolerating double disk failures in raid architectures,” ACM SIGARCH Comput. Archit. News, vol. 22, no. 2, pp. 245–254, 1994.
[17] Z. Wang, A. G. Dimakis, and J. Bruck, “Rebuilding for array codes in distributed storage systems,” in Proc. IEEE Globecom Workshops, Dec. 2010, pp. 1905–1909.
[18] L. Xu, V. Bohossian, J. Bruck, and D. G. Wagner, “Low-density MDS codes and factors of complete graphs,” IEEE Trans. Inf. Theory, vol. 45, no. 6, pp. 1817–1826, Sep. 1999.
[19] L. Xu and J. Bruck, “X-code: MDS array codes with optimal encoding,” IEEE Trans. Inf. Theory, vol. 45, no. 1, pp. 272–276, Jan. 1999.
[20] P. Corbett, B. English, A. Goel, T. Grcanac, S. Kleiman, J. Leong, and S. Sankar, “Row-diagonal parity for double disk failure correction,” in Proc. 3rd USENIX Conf. File Storage Technol. Berkeley, CA, USA: USENIX Association, 2004, pp. 1–14.
[21] L. Xiang, Y. Xu, J. C. S. Lui, and Q. Chang, “Optimal recovery of single disk failure in RDP code storage systems,” ACM SIGMETRICS Perform. Eval. Rev., vol. 38, no. 1, pp. 119–130, Jun. 2010.
[22] C. Huang and L. Xu, “STAR : An efficient coding scheme for correcting triple storage node failures,” IEEE Trans. Comput., vol. 57, no. 7, pp. 889–901, Jul. 2008.

[23] I. Tamo, Z. Wang, and J. Bruck, “Zigzag codes: MDS array codes with optimal rebuilding,” IEEE Trans. Inf. Theory, vol. 59, no. 3, pp. 1597–1616, Mar. 2013.

[24] K. V. Rashmi, N. B. Shah, and K. Ramchandran, “A piggybacking design framework for read-and download-efficient distributed storage codes,” in Proc. IEEE Int. Symp. Inf. Theory, Jul. 2013, pp. 331–335.

[25] O. Khan, R. C. Burns, J. S. Plank, W. Pierce, and C. Huang, “Rethinking erasure codes for cloud file systems: Minimizing I/O for recovery and degraded reads,” in Proc. Usenix Conf. File Storage Technol., 2012, p. 20.

[26] 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. Usenix Hotstorage, 2013, pp. 1–5.

[27] S. Yuan and Q. Huang, “Generalized piggybacking codes for distributed storage systems,” in Proc. IEEE Global Commun. Conf. (GLOBECOM), Dec. 2016, pp. 1–6.

[28] S. Kumar, A. G. I Amat, I. Andriyanova, and F. Brannstrom, “A family of erasure correcting codes with low repair bandwidth and low repair complexity,” in Proc. IEEE Global Commun. Conf. (GLOBECOM), Dec. 2015, pp. 1–6.

[29] C. Shangguan and G. Ge, “A new piggybacking design for systematic MDS storage codes,” 2016, arXiv:1610.08223. [Online]. Available: http://arxiv.org/abs/1610.08223

[30] B. Yang, X. Tang, and J. Li, “A systematic piggybacking design for minimum storage regenerating codes,” IEEE Trans. Inf. Theory, vol. 61, no. 11, pp. 5779–5786, Nov. 2015.

[31] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, “Fog computing and its role in the Internet of Things,” in Proc. 1st Ed. MCC workshop Mobile cloud Comput. (MCC), 2012, pp. 13–16.

[32] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, “Edge computing: Vision and challenges,” IEEE Internet Things J., vol. 3, no. 5, pp. 637–646, Oct. 2016.

[33] S. Yuan, Q. Huang, and Z. Wang, “A repair-efficient coding for distributed storage systems under piggybacking framework,” IEEE Trans. Commun., vol. 66, no. 8, pp. 3245–3254, Aug. 2018.

[34] C. Huang, H. Simicici, Y. Xu, A. Ogos, B. Calder, P. Gopalan, J. Li, and S. Yekhanin, “Erasure coding in windows azure storage,” in Proc. Usenix Conf. Tech. Conf., vol. 37, no. 5, 2012, pp. 82–96.

[35] M. Sathiamoorthy, M. Asteris, D. Papailiopoulos, A. G. Dimakis, R. Vadali, S. Chen, and D. Borthakur, “XORing elephants: Novel erasure codes for big data,” Proc. VLDB Endowment, vol. 6, no. 5, pp. 325–336, Mar. 2013.

[36] C. Suh and K. Ramchandran, “On the existence of optimal exact-repair MDS codes for distributed storage,” Univ. California Berkeley, Berkeley, CA, USA, Tech. Rep., 2010, arXiv:1004.4663. [Online]. Available: https://arxiv.org/abs/1004.4663

[37] K. V. Rashmi, N. B. Shah, and K. Ramchandran, “A piggybacking design framework for read-and download-efficient distributed storage codes,” IEEE Trans. Inf. Theory, vol. 63, no. 9, pp. 5802–5820, Sep. 2017.

[38] N. B. Shah, K. V. Rashmi, P. V. Kumar, and K. Ramchandran, “Distributed storage codes with repair-by-transfer and nonachievability of interior points on the storage-bandwidth tradeoff,” IEEE Trans. Inf. Theory, vol. 58, no. 3, pp. 1837–1852, Mar. 2012.

[39] K. V. Rashmi, N. B. Shah, D. Gu, H. Kuang, D. Borthakur, and K. Ramchandran, “A ‘hitchhiker’s’ guide to fast and efficient data reconstruction in erasure-coded data centers,” ACM SIGCOMM Comput. Commun. Rev., vol. 44, no. 4, pp. 331–342, 2015.

[40] V. R. Cadambe, S. A. Jafar, H. Maleki, K. Ramchandran, and C. Suh, “Asymptotic interference alignment for optimal repair of MDS codes in distributed storage,” IEEE Trans. Inf. Theory, vol. 59, no. 5, pp. 2974–2987, May 2013.

[41] D. Papailiopoulos, A. G. Dimakis, and V. R. Cadambe, “Repair optimal erasure codes through Hadamard designs,” IEEE Trans. Inf. Theory, vol. 59, no. 5, pp. 3021–3037, May 2013.

[42] I. Tamo, Z. Wang, and J. Bruck, “Access versus bandwidth in codes for storage,” IEEE Trans. Inf. Theory, vol. 60, no. 4, pp. 2028–2037, Apr. 2014.

RONG SUN (Member, IEEE) received the B.E. degree in telecommunications engineering and the M.E. and Ph.D. degrees in communications and information systems from Xidian University, Xian, China, in 1998, 2001, and 2008, respectively. She is currently with the State Key Laboratory of Integrated Services Networks, School of Telecommunications, Xidian University. Her research interests include wireless communications, channel coding design, and information theory.

XIN LI received the M.S. degree in electronics and communication engineering from Xidian University, in 2019. He is currently working with ZTE Cooperation. His research interests include channel coding and distributed storage.

LU ZHANG is currently pursuing the M.S. degree in electronics and communication engineering with Xidian University. Her research interests include channel coding and distributed storage.

JINGWEI LIU (Member, IEEE) received the B.S. degree in applied mathematics and the M.S. and Ph.D. degrees in communication and information systems from Xidian University, Xian, China, in 2001, 2004, and 2007, respectively. He is currently with the School of Telecommunications Engineering, Xidian University. He has published more than 40 articles in journals and conference proceedings. His research interests include information security, network security, and cryptography. He is a member of the Chinese Association for Cryptologic Research.