Generalized Piggybacking Codes for Distributed Storage Systems

Shuai Yuan[1], Qin Huang[2,1,*], *Senior Member, IEEE*, Zulin Wang[2], *Member, IEEE*

[1]Qian Xuesen Laboratory of Space Technology

China Academy of Space Technology, Beijing, China, 100094

[2]School of Electronic and Information Engineering

Beihang University, Beijing, China, 100191

Email: yuanshuai@qxslab.cn; qhuang.smash@gmail.com; wzulin_201@163.com

Abstract

This paper generalizes the piggybacking constructions for distributed storage systems by considering various protected instances and piggybacked instances. Analysis demonstrates that the proportion of protected instances determines the average repair bandwidth for a systematic node. By optimizing the proportion of protected instances, the repair ratio of generalized piggybacking codes approaches zero instead of 50% as the number of parity check nodes tends to infinity. Furthermore, the computational complexity for repairing a single systematic node cost by generalized piggybacking codes is less than that of the existing piggybacking designs.

Index Terms

piggybacking, distributed storage systems, MDS, node repair

I. INTRODUCTION

Nowadays, distributed storage systems (DSSs) are being increasingly employed by network applications. Data in DSSs is deployed over multiple storage devices. However, these discrete devices are prone to failure because of malfunctions or maintenance. In order to ensure the reliability of the stored data even in the occurrence of node unavailability, DSSs are supposed to introduce redundancy to resist storage node failures. Replication is the simplest redundant fashion, and has been adopted to improve the reliability by many DSSs, such as the Google File System [1] and the Hadoop Distributed File System (HDFS) [2]. With the rapid growth of amount of storage data, erasure coding has become a better choice for DSSs. Compared with replication, it is able to provide orders of magnitude reliability increasing for same storage resource consumption [3]. As a result, several large-scale systems, such as OceanStore [4], Total Recall [5], Windows Azure Storage [6], and Google Colossus(GFS2) [7], have employed erasure coding techniques to improve their storage efficiency.

Maximum distance separable (MDS) codes as one kind of erasure codes have been introduced into many DSSs for their optimal storage efficiency. MDS property can be used to recover missing data in a DSS. Consider an n-node DSS deployed with an (n, k) MDS code. If one node of this storage system is failed, data stored in k nodes is required to reconstruct the missing data in this failure node. k times amount of stored data is needed to recover the missing data. Thus, the usage of network and disk is significantly high, i.e., the repair efficiency is very low. To address this repair issue, many codes have been constructed to reduce the transmission data for repairing failure node.

As the statement in [8], there are three types of node repair: exact repair, functional repair and exact repair of the systematic part. However, exact repair is the most considered from in practical

DSSs. In [9], Dimakis et al. defined the amount of transmission data during repairing one single failed node as *repair bandwidth*. The authors derived an optimal tradeoff between storage and repair bandwidth (theoretic cut-set bound), and proposed *regenerating codes* which lie on the tradeoff curve. In [10], [11], [12], [13], [14], the existence and the construction of regenerating codes have been studied. However, the optimal tradeoff provided by regenerating codes was only derived for functional repair. Almost all the interior points on the storage-bandwidth tradeoff are not achievable under exact repair [15].

MDS array codes are another important class of erasure codes used in DDSs. They have the advantage of simple encoding and decoding procedures, so that they can be easily implemented in hardware devices. Many designs of MDS array codes, such as EVENODD [16], B-code [17], X-code [18], RDP [19], STAR [20] and Zigzag codes [21], have been presented for storage and communication applications. However, the repair bandwidth of MDS array codes can not achieve the theoretic cut-set bound.

In 2011, Rashmi et al. proposed a new kind of distributed storage codes called *piggybacking codes* to reduce the data amount read and downloaded for node repair [22]. The key idea of piggybacking codes is taking several instances of an existing base code, and attaching linear combinations of symbols in some protected instances to other non-protected instances. Hence, the missing symbols in protected instances are able to be recovered by solving these linear equations instead of MDS decoding. Piggybacking is a simple and useful construction to improve the repair efficiency of missing nodes. Several designs of piggybacking codes were presented in [22] and [23]. These designs are able to save 25% to 50% repair bandwidth for one failed node on average. Facebook Warehouse Cluster and the new Hadoop Distributed File System (HDFS) have employed piggybacking codes to improve their repair efficiency [24].

Although piggybacking codes are practical and easy to implementation, the reduction of repair bandwidth of the proposed piggybacking designs still has a gap to the theoretic cut-set bound of regenerating codes. In [23], Rashmi, Shah, and Ramchandran gave three specific piggybacking

constructions. The second one we represent with RSR-II is the most efficient construction in terms of repair bandwidth. The description in [23] shows that RSR-II codes are able to save up to 50% of repair bandwidth. This paper investigates the mechanism in reduction of repair bandwidth by using piggybacking codes. From the recovery methods of the systematic symbols, we distinguish instances of piggybacking codes with protected stripes and non-protected stripes. An analysis of a lower bound on the repair bandwidth of RSR-II codes implies that the proportion of protected instances determines the repair efficiency of piggybacking constructions.

This paper firstly presents a generalized piggybacking design with various protected and non-protected stripes in order to obtain various proportion of protected stripes. Second, a lower bound and an upper bound on the repair bandwidth of generalized piggybacking codes are introduced. The analysis of the two bounds indicates that by optimizing the proportion of protected stripes, the *repair ratio* (defined as average repair bandwidth as a fraction of the amount of original messages) of a generalized piggybacking code approaches zero instead of 50% as the number of parity check nodes tends to infinity. It is closer to that of minimum storage regenerating (MSR) codes which has the theoretical lower bound. At last, the computational complexity for the repair of a single failed systematic node is analyzed. The results show that the generalized piggybacking codes are able to provide more efficient repair with little complexity overhead.

The remainder of this paper is organized as follows. Section II briefly introduces the piggybacking framework and RSR-II codes. Section III performs an analysis of the repair efficiency of RSR-II codes. Our generalized piggybacking codes are presented in Section IV. Finally, the conclusion is given in Section V.

II. BACKGROUND

A. *Maximum distance separable codes*

Consider an (n, k, d) linear block code $\mathcal{C}$, where n is its code length, k is its dimension, and d represents the minimum Hamming distance. Code $\mathcal{C}$ is called an MDS code, if its minimum

Hamming distance d meets the Singleton bound, i.e.,

$$d = n - k + 1. \tag{1}$$

MDS codes are an important class of linear block codes. For given parameters n and k, the minimum distance d reaches the maximum possible value. Thus, MDS codes are able to correct as many as $(n - k)$ erasures for given n and k.

MDS codes have been extensively applied in many DSSs. In an n-node storage system, initially the original message is divided into k information packets. Subsequently, the k packets are encoded into n packets and stored in the n nodes respectively. With the MDS property, messages from any k out of n nodes could reconstruct the original message. Thus, the system is able to tolerate the failures of any $(n - k)$ storage nodes.

B. Piggybacking framework

In this subsection, we introduce the piggybacking framework which is the basis of constructing piggybacking codes. Piggybacking framework guarantees that DSSs are able to employ piggybacking codes without extra cost of storage. Moreover, the decoding properties of the error-correction codes adopted by original DSSs, such as the minimum distance or the MDS property, are not ruined by piggybacking reconstruction.

In general, the piggybacking framework operates on multiple instances of an existing base code and adds several designed functions of the data in some instances onto other instances. The base code of piggybacking framework can be arbitrary. In fact, it is a very attractive feature in practice. Under the piggybacking framework, the DSSs enjoy a repair bandwidth reduction with only small modification based on their existing error-correction codes.

Consider a linear block code $\mathcal{C}_1$ represented by n encoding functions $\{f_i\}_{i=1}^{n}$. Suppose $\mathbf{u}$ is the original message of $\mathcal{C}_1$. The n encoded symbols are $\{f_i(\mathbf{u})\}_{i=1}^{n}$. For an n-node system, using $\mathcal{C}_1$ as the base code, the piggybacking framework, which has α instances of $\mathcal{C}_1$, is illustrated in Fig.1.

	stripe 1	stripe 2	stripe 3	$\cdots$	stripe α
node 1	$f_1(\mathbf{u}_1)$	$f_1(\mathbf{u}_2) + g_{2,1}(\mathbf{u}_1)$	$f_1(\mathbf{u}_3) + g_{3,1}(\mathbf{u}_1, \mathbf{u}_2)$	$\cdots$	$f_1(\mathbf{u}_\alpha) + g_{\alpha,1}(\mathbf{u}_1, \cdots, \mathbf{u}_{\alpha-1})$
node 2	$f_2(\mathbf{u}_1)$	$f_2(\mathbf{u}_2) + g_{2,2}(\mathbf{u}_1)$	$f_2(\mathbf{u}_3) + g_{3,2}(\mathbf{u}_1, \mathbf{u}_2)$	$\cdots$	$f_2(\mathbf{u}_\alpha) + g_{\alpha,2}(\mathbf{u}_1, \cdots, \mathbf{u}_{\alpha-1})$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
node n	$f_n(\mathbf{u}_1)$	$f_n(\mathbf{u}_2) + g_{2,n}(\mathbf{u}_1)$	$f_n(\mathbf{u}_3) + g_{3,n}(\mathbf{u}_1, \mathbf{u}_2)$	$\cdots$	$f_n(\mathbf{u}_\alpha) + g_{\alpha,n}(\mathbf{u}_1, \cdots, \mathbf{u}_{\alpha-1})$

Fig. 1. Piggybacking framework

As shown in Fig.1, the n rows correspond to the n storage nodes, the α columns are called α *stripes*, $\{\mathbf{u}_i\}_{i=1}^{\alpha}$ are α independent original messages and $\{g_{i,j}\}_{i=2,j=1}^{\alpha,n}$ are *piggyback functions*.

It is a very important consideration that the piggyback functions added on the i-th stripe ($i \in \{2, 3, \cdots, \alpha\}$) can only be linear combinations of original messages of stripes $\{1, 2, \cdots, (i-1)\}$. This principle guarantees that all the stripes of this piggybacking framework are decodable through a recursion process: In stripe 1, no piggyback functions are added, so the original message $\mathbf{u}_1$ can be directly recovered by using the decoding procedure of $\mathcal{C}_1$. For stripe 2, with the decoded $\mathbf{u}_1$, it is easy to compute the added piggyback functions $\{g_{2,j}(\mathbf{u}_1)\}_{j=1}^{n}$ and subtract them from the stored symbols. Then, $\mathbf{u}_2$ is decodable. In a similar way, after the decoding procedures of stripes $\{1, 2, \cdots, (i-1)\}$ are finished, $\mathbf{u}_1, \mathbf{u}_2, \cdots, \mathbf{u}_{i-1}$ are available to the piggyback functions $\{g_{i,j}(\mathbf{u}_1, \cdots, \mathbf{u}_{i-1})\}_{j=1}^{n}$. The base code of this stripe is obtained after subtracting these piggybacking functions, so that $\mathbf{u}_i$ can be recovered.

As the statement above, the α symbols stored in one node are independent. Sometimes, an invertible linear transformation is performed to simplify the computation. Such a transformation still retains the decoding properties of the piggybacking framework.

C. RSR-II codes

Under the piggybacking framework described in Section.II-B, Rashmi et al. have presented three designs of piggybacking codes for different considerations. The second design RSR-II is

constructed for the purpose of pursuing high efficiency of repair. As the statement in [23], RSR-II codes can save up to 50% repair bandwidth of a systematic node.

For the sake of simple description, an (n, k) MDS code in systematic form is chosen as the base code. Denote $r = n - k$ as the number of parity check nodes. RSR-II codes consist of $(2r - 3)$ instances of the base code. Represent the $(2r - 3)$ associated original messages as $\mathbf{a}_1, \mathbf{a}_2, \cdots, \mathbf{a}_{2r-3}$, where $\mathbf{a}_i$ $(i \in \{1, 2, \cdots, 2r - 3\})$ is a vector of length k, and $\mathbf{a}_i = [a_{i,1}, a_{i,2}, \cdots, a_{i,k}]$. Then, the $(2r - 3)$ stripes are shown in the following form:

node 1	$a_{1,1}$	$a_{2,1}$	$\cdots$	$a_{2r-3,1}$
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
node k	$a_{1,k}$	$a_{2,k}$	$\cdots$	$a_{2r-3,k}$
node $k+1$	$\mathbf{p}_1^T \mathbf{a}_1$	$\mathbf{p}_1^T \mathbf{a}_2$	$\cdots$	$\mathbf{p}_1^T \mathbf{a}_{2r-3}$
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
node $k+r$	$\mathbf{p}_r^T \mathbf{a}_1$	$\mathbf{p}_r^T \mathbf{a}_2$	$\cdots$	$\mathbf{p}_r^T \mathbf{a}_{2r-3}$

where $\mathbf{p}_1, \mathbf{p}_2, \cdots, \mathbf{p}_r$ are r encoding vectors corresponding to the r parity check symbols of the base code.

The piggyback functions of RSR-II codes are $(r - 1)^2$ linear combinations of the systematic symbols of the first $(r - 1)$ stripes, and they are added on the last $(r - 1)$ parity check symbols of the last $(r - 1)$ stripes. The construction of these piggyback functions is taken in three steps.

First, the k systematic nodes are split into $(r - 1)$ node sets $\{S_i\}_{i=1}^{r-1}$ as evenly as possible. Without loss of generality, we suppose k is not a multiple of $(r - 1)$, and define three variables as follows,

$$t_l = \left\lfloor \frac{k}{r-1} \right\rfloor, \ t_h = \left\lceil \frac{k}{r-1} \right\rceil, \ t = k - (r-1)t_l. \tag{2}$$

Hence, the first t node sets $\{S_i\}_{i=1}^{t}$ are of size t_h, and the remaining $\{S_i\}_{i=t+1}^{r-1}$ are of size t_l.

$\mathbf{p}_i^T\mathbf{a}_1$	$\cdots$	$\mathbf{p}_i^T\mathbf{a}_{r-2}$	$\mathbf{p}_i^T\mathbf{a}_{r-1}+$ $\sum_{j=1,j\neq i-1}^{r-1}\mathbf{q}_{i,j}^T\hat{\mathbf{v}}_i$	$\mathbf{p}_i^T\mathbf{a}_r+$ $\mathbf{q}_{i,1}^T\mathbf{v}_i$	$\cdots$	$\mathbf{p}_i^T\mathbf{a}_{r+i-3}+$ $\mathbf{q}_{i,i-2}^T\mathbf{v}_i$	$\mathbf{p}_i^T\mathbf{a}_{r+i-2}+$ $\mathbf{q}_{i,i}^T\mathbf{v}_i$	$\cdots$	$\mathbf{p}_i^T\mathbf{a}_{2r-3}+$ $\mathbf{q}_{i,r-1}^T\mathbf{v}_i$

(a) node $(k+i)$ with piggyback functions

$\mathbf{p}_i^T\mathbf{a}_1$	$\cdots$	$\mathbf{p}_i^T\mathbf{a}_{r-2}$	$\mathbf{q}_{i,i-1}^T\mathbf{a}_{r-1}-$ $\sum_{j=r}^{2r-3}\mathbf{p}_i^T\mathbf{a}_j$	$\mathbf{p}_i^T\mathbf{a}_r+$ $\mathbf{q}_{i,1}^T\mathbf{v}_i$	$\cdots$	$\mathbf{p}_i^T\mathbf{a}_{r+i-3}+$ $\mathbf{q}_{i,i-2}^T\mathbf{v}_i$	$\mathbf{p}_i^T\mathbf{a}_{r+i-2}+$ $\mathbf{q}_{i,i}^T\mathbf{v}_i$	$\cdots$	$\mathbf{p}_i^T\mathbf{a}_{2r-3}+$ $\mathbf{q}_{i,r-1}^T\mathbf{v}_i$

(b) node $(k+i)$ with an invertible linear transform

Fig. 2. Stored symbols in piggybacked node $(k+i)$

Second, define two sets of vectors of length k $\{\mathbf{v}_i\}_{i=2}^r$ and $\{\hat{\mathbf{v}}_i\}_{i=2}^r$ with

$$\mathbf{v}_i = \mathbf{a}_{r-1} + i\mathbf{a}_{r-2} + i^2\mathbf{a}_{r-3} + \cdots + i^{r-2}\mathbf{a}_1, \tag{3}$$

$$\hat{\mathbf{v}}_i = \mathbf{v}_i - \mathbf{a}_{r-1} = i\mathbf{a}_{r-2} + i^2\mathbf{a}_{r-3} + \cdots + i^{r-2}\mathbf{a}_1. \tag{4}$$

Then, introduce $(r-1)^2$ selection vectors $\{\mathbf{q}_{i,j}\}_{i=2,j=1}^{r,r-1}$ to separate the k tuples in each vector of $\{\mathbf{v}_i\}_{i=2}^r, \{\hat{\mathbf{v}}_i\}_{i=2}^r$ into $(r-1)$ segments. And the selection vectors are defined as follows

$$\mathbf{q}_{i,j} = \mathbf{M}_j\mathbf{p}_i, \tag{5}$$

where $\{\mathbf{M}_j\}_{j=1}^{r-1}$'s are diagonal matrices of size $(k\times k)$. On the diagonal of $\mathbf{M}_j$, only the positions corresponding to the systematic nodes in S_j are "1". Therefore,

$$\sum_{j=1}^{r-1}\mathbf{q}_{i,j} = \mathbf{p}_i, \ \forall i \in \{2,\cdots,r\}. \tag{6}$$

Finally, add the piggyback functions of $\{\mathbf{v}_i\}_{i=2}^r, \{\hat{\mathbf{v}}_i\}_{i=2}^r$ and $\{\mathbf{q}_{i,j}\}_{i=2,j=1}^{r,r-1}$ into the parity check symbols in the last $(r-1)$ nodes. Hence, node $(k+i)$, $i \in \{2,3,\cdots,r\}$, has the following form as shown in Fig.2(a). An invertible linear transformation is introduced to reduce the complexity for node repair. Finally, symbols in node $(k+i)$ are illustrated in Fig.2(b).

D. Repair bandwidth of RSR-II codes

We use *repair ratio* γ to represent the measure of repair efficiency of a distributed storage code. Repair ratio is defined as the average amount of transfer data needed for repairing one

failure node as a fraction of original messages. In this subsection, we recall the repair procedure of one systematic node by RSR-II codes. Then, the repair ratio of RSR-II γ_1^{sys} is computed.

Consider an n-node DSS deployed with an (n, k) RSR-II code. For the sake of simple description, we represent the first $(r-1)$ stripes as *protected stripes*, whose systematic symbols are involved in the piggyback functions and defined as *protected symbols*. Meanwhile, the last $(r-2)$ stripes are represented as *non-protected stripes*, whose systematic symbols are named with *non-protected symbols*. If the l-th systematic node fails, repair procedure of this node is to recover the missing protected symbols $\{a_{i,l}\}_{i=1}^{r-1}$ and the missing non-protected symbols $\{a_{i,l}\}_{i=r}^{2r-3}$. Assume node l belongs to S_j which is one of the $(r-1)$ node sets described in Section.II-C. The repair procedure is described in Algorithm 1.

Algorithm 1 The repair algorithm of RSR-II codes

---

1 Recovering the missing non-protected symbols $\{a_{i,l}\}_{i=r}^{2r-3}$;

   The base code of this RSR-II code is in systematic MDS form. According to MDS property, $a_{i,l}$ can be directly recovered with $a_{i,1}, \cdots, a_{i,l-1}, a_{i,l+1}, \cdots, a_{i,k}, \mathbf{p}_1^T \mathbf{a}_i$.

2 Getting the piggyback functions involved with the missing protected systems $\{a_{i,l}\}_{i=1}^{r-1}$;

   As statement in II-C, there are $(r-1)$ piggyback functions containing $\mathbf{q}_{i,j}$'s $(i = [2, \cdots, r])$. These piggyback functions are linear combinations of the protected symbols in S_j. Download the $(r-1)$ parity check symbols containing the $(r-1)$ piggyback functions, and subtract the items about $\{\mathbf{a}_j\}_{j=r}^{2r-3}$. Then, the $(r-1)$ piggyback functions involved with $\{a_{i,l}\}_{i=1}^{r-1}$ are left.

3 Recovering the missing protected symbols $\{a_{i,l}\}_{i=1}^{r-1}$;

   Including $\{a_{i,l}\}_{i=1}^{r-1}$, the other surviving protected symbols in $S_j \backslash l$ are also involved with the $(r-1)$ piggyback functions obtained in step 2. Download these surviving symbols, and subtract them out from the $(r-1)$ piggyback functions. Then, $\{a_{i,l}\}_{i=1}^{r-1}$ can be reconstructed by solving the left $(r-1)$ linear combinations.

From Algorithm 1, $(r-2)k$ symbols are needed to be downloaded in step 1, and $(r-1)$ symbols are needed in step 2. In step 3, if the size of S_j is t_h, the number of downloaded symbols is $(r-1)(t_h-1)$. Otherwise, if the size is t_l, $(r-1)(t_l-1)$ symbols are downloaded. We denote the average repair bandwidth of one systematic node as B_1^{sys}. The number of systematic nodes in the node sets of size t_h is $t \cdot t_h$, and the number of those systematic nodes in the node set of size t_l is $(r-1-t)t_l$. Thus

$$
\begin{aligned}
B_1^{sys} &= \frac{1}{k}[tt_h((r-2)k + (r-1)t_h) \\
&\quad + (r-1-t)t_l((r-2)k + (r-1)t_l)].
\end{aligned}
\tag{7}
$$

Thus, the repair ratio γ_1^{sys} is

$$
\begin{aligned}
\gamma_1^{sys} &= \frac{B_1^{sys}}{k(2r-3)} \\
&= \frac{1}{k^2(2r-3)}[tt_h((r-2)k + (r-1)t_h) + \\
&\quad (r-1-t)t_l((r-2)k + (r-1)t_l)] \\
&= \frac{1}{k^2(2r-3)}[k^2(r-2) + \\
&\quad (tt_h^2 + (r-1-t)t_l^2)(r-1)].
\end{aligned}
\tag{8}
$$

III. EFFICIENCY ANALYSIS FOR RSR-II CODES

In this section, a further analysis on the repair efficiency of RSR-II is performed.

Here, we introduce a notation *stripe-repair ratio* η to measure the repair efficiency of one stripe

$$
\eta \triangleq \frac{\text{repair bandwidth for a systematic symbol}}{\text{the amount of original message of this stripe}}.
$$

Consider a piggybacking code with β stripes. Assume the stripe-repair ratios of these stripes are $\{\eta_i\}_{i=1}^{\beta}$. Denote the proportions of these stripes as $\{p_i\}_{i=1}^{\beta}$. Thus, the repair ratio for systematic nodes of this piggybacking code γ^{sys} has the following form,

$$
\gamma^{sys} = \sum_{i=1}^{\beta} p_i \eta_i.
\tag{9}
$$

Recall the RSR-II codes described in Section II-D. The repair procedure deals with the missing protected and non-protected symbols in two different measures: MDS decoding is adopted for the recovery of non-protected symbols, and the amount of downloading for repairing one missing non-protected symbol is k symbols. As regard to the missing protected symbols, solving linear combinations is employed, and the average bandwidth is t_h or t_l, which depends on the size of node set containing the failure node. Denote η_p and η_{np} as the stripe-repair ratios of protected and non-protected stripes, respectively. The amount of original message of one stripe equals to the k symbols stored in the systematic nodes. Hence,

$$\eta_p \approx \frac{t_h \text{ or } t_l}{k} \approx \frac{1}{r-1} \tag{10}$$

$$\eta_{np} = 1. \tag{11}$$

Although only an approximate value of η_p is given by Equation (10), it is obvious that $\eta_p < \eta_{np}$, i.e., repair procedure for protected stripes requires less downloaded symbols compared with non-protected stripes. This is the mechanism in reduction of repair bandwidth by using piggybacking codes.

In the remainder of this section, we explore the critical factors influencing the repair efficiency through an analysis of γ_1^{sys}. Represent the proportion of protected stripes with p_p. Thus, the proportion of non-protected stripes is $(1 - p_p)$. Rewrite γ_1^{sys} as the form of Equation (9). Then,

$$\gamma_1^{sys} = \frac{r-2}{2r-3} \cdot \frac{k^2}{k^2} + \frac{r-1}{2r-3} \cdot \frac{tt_h^2 + (r-1-t)t_l^2}{k^2}$$

$$= (1 - p_p) \cdot \eta_{np} + p_p \cdot \eta_p, \tag{12}$$

where $p_p = \frac{r-1}{2r-3}$, $\eta_{np} = 1$ and $\eta_p = \frac{tt_h^2 + (r-1-t)t_l^2}{k^2}$. The inequality of quadratic and arithmetic means tells that for x nonnegative integers $n_1, n_2, \cdots, n_x$, they satisfy the following inequality.

$$\sum_{i=1}^{x} n_i^2 \geq \frac{\left(\sum_{i=1}^{x} n_i\right)^2}{x}. \tag{13}$$

Thus,

$$
\begin{aligned}
\eta_p &= \frac{tt_h^2 + (r-1-t)t_l^2}{k^2} \\
&\geq \frac{(tt_h + (r-1-t)t_l)^2}{k^2(r-1)} \\
&= \frac{1}{r-1},
\end{aligned}
\tag{14}
$$

with equality if and only if $t_l = t_h$, i.e., k is a multiple of $(r-1)$. In this case, γ_1^{sys} is able to reach a lower bound $\min(\gamma_1^{sys})$, and

$$
\begin{aligned}
\min(\gamma_1^{sys}) &= \frac{r-2}{2r-3} + \frac{r-1}{2r-3} \cdot \frac{1}{r-1} \\
&= \frac{r-1}{2r-3}.
\end{aligned}
\tag{15}
$$

According to Equation (15), γ_1^{sys} approaches 0.5 as the number of parity check nodes tends to infinite, i.e., RSR-II codes are able to save at most 50% repair bandwidth. For a DSS whose parameters (n, k, r) are given, in order to further improve the repair efficiency, the structure of piggybacking design is supposed to be modified. As the analysis above, the protected stripe-repair ratio η_p is smaller than η_{np}. It implies that the repair efficiency of piggybacking codes may be improved by increasing p_p according to Equation (12). Actually, larger p_p means more protected symbols involved in one piggyback function that leads to the reduction of η_p. Therefore, it is possible to improve the repair efficiency of piggybacking codes by optimizing the proportion of protected stripes p_p.

IV. GENERALIZED PIGGYBACKING CODES

In this section, we present a generalized construction which contains various protected and non-protected stripes. An analysis is performed to clarify the relationship between repair ratio γ and the proportion of protected stripes p_p. The results show that our proposed generalized piggybacking codes are able to provide more efficient node repair by optimizing p_p. The repair ratio γ_2^{sys} of the generalized piggybacking codes approaches zero when the number of the parity check nodes tends to infinity.

Fig. 3. $(s + p)$ instances of the base code.

A. Code design

Similarly, choose an (n, k) systematic MDS code $\mathcal{C}_2$ as the base code of a generalized piggybacking code. $r = n - k$ is the parity check number. Two parameters s and p are introduced to represent the numbers of protected and piggybacked stripes, respectively. Figure 3 depicts the $(s + p)$ instances of $\mathcal{C}_2$.

According to the construction principle of piggybacking framework, piggyback functions added on the i-th stripe should only involve the original messages of the stripes $[1, \cdots, i-1]$. For the sake of simple analysis, we add the piggyback functions only on the parity check symbols in non-protected stripes. Redefine the non-protected stripes as piggybacked stripes. As illustrated in Fig.3, all symbols stored in the $(s + p)$ stripes are divided into 4 regions.

- Region A contains all the systematic symbols of the protected stripes.
- Region B contains all the systematic symbols and the first parity check symbol of the piggybacked stripes.
- Region C contains all the parity check symbols of the protected stripes.
- Region D contains the last $(r - 1)$ parity check symbols of the piggybacked stripes.

Once a systematic node failure happens, the repair procedure is supposed to regenerate the

$(s+p)$ missing symbols in Region A and B. Similar to RSR-II codes, the systematic symbols in Region B are self-sustaining: According to the MDS property, missing symbols in one row of Region B could be recovered by the surviving symbols in the other k rows. As for the systematic symbols in Region A, piggybacking functions are constructed to protected them. These piggyback functions are supposed to be embedded in Region D. The size of Region D is $(r-1)p$, i.e., at most $(r-1)p$ piggyback functions can be designed. It is a noteworthy fact that the s failed protected symbols in one row of Region A should be simultaneously recovered by solving a set of linear combinations. In order to guarantee that there are enough piggyback functions to simultaneously recover those s missing symbols in Region A, the following inequality must be satisfied when we choose the parameters s and p.

$$(r-1)p \geq s. \tag{16}$$

In the remainder this subsection, an method of the construction of $(r-1)p$ piggyback functions is illustrated as follows.

1 Construct a $\lceil \frac{ks}{(r-1)p} \rceil \times (r-1)p$ empty piggybacking array.

   Each column of this piggybacking array corresponds to one piggyback function.

2 Fill the protected symbols in Region A into the piggybacking array.

   The protected symbols in Region A form a $k \times s$ array as shown in Fig.3. Step 2 takes these symbols in rowwise from the $k \times s$ array and fills them into the piggybacking array. Obviously, if ks is not divisible by $(r-1)p$, the last row of this piggyback array would not be full.

3 Obtain the $(r-1)p$ piggybacking functions, and add them in Region D. After all protected symbols are allocated into the piggyback array, sum the symbols in each column up. Thus, $(r-1)p$ piggybacking functions are obtained, and they can be added into Region D in an arbitrary order.

It is remarkable that the piggyback functions are only summations of some protected symbols.

As a result, the recovery of missing protected symbols could be very simple. An example is presented to illustrate the partition method and the repair procedure.

Example 1. *Consider an $(8,4)$ systematic MDS code as the base code. Set $s = 3$, and $p = 2$. Denote $\mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{d}, \mathbf{e}$ of length 4 as the 5 input message vectors. Thus, the original storage array is*

$$
\begin{array}{ccc|cc}
a_1 & b_1 & c_1 & d_1 & e_1 \\
a_2 & b_2 & c_2 & d_2 & e_2 \\
a_3 & b_3 & c_3 & d_3 & e_3 \\
a_4 & b_4 & c_4 & d_4 & e_4 \\
\hline
\mathbf{p}_1^T\mathbf{a} & \mathbf{p}_1^T\mathbf{b} & \mathbf{p}_1^T\mathbf{c} & \mathbf{p}_1^T\mathbf{d} & \mathbf{p}_1^T\mathbf{e} \\
\mathbf{p}_2^T\mathbf{a} & \mathbf{p}_2^T\mathbf{b} & \mathbf{p}_2^T\mathbf{c} & \mathbf{p}_2^T\mathbf{d} & \mathbf{p}_2^T\mathbf{e} \\
\mathbf{p}_3^T\mathbf{a} & \mathbf{p}_3^T\mathbf{b} & \mathbf{p}_3^T\mathbf{c} & \mathbf{p}_3^T\mathbf{d} & \mathbf{p}_3^T\mathbf{e} \\
\mathbf{p}_4^T\mathbf{a} & \mathbf{p}_4^T\mathbf{b} & \mathbf{p}_4^T\mathbf{c} & \mathbf{p}_4^T\mathbf{d} & \mathbf{p}_4^T\mathbf{e}
\end{array}
$$

The protected symbols in Region A are $\{a_1, a_2, a_3, a_4\}$, $\{b_1, b_2, b_3, b_4\}$, $\{c_1, c_2, c_3, c_4\}$ and $\{d_1, d_2, d_3, d_4\}$. Fill them into a 2×6 piggyback array. We have

$$
\begin{array}{cccccc}
a_1 & b_1 & c_1 & a_2 & b_2 & c_2 \\
a_3 & b_3 & c_3 & a_4 & b_4 & c_4
\end{array}
$$

Sum the symbols in each column up, and then we achieve the six piggyback functions $(a_1 + a_3), (b_1 + b_3), (c_1 + c_3), (a_2 + a + 4), (b_2 + b_4), (c_2 + c_4)$. Finally, the generalized piggybacking code can be constructed as follows

a_1	b_1	c_1	d_1	e_1
a_2	b_2	c_2	d_2	e_2
a_3	b_3	c_3	d_3	e_3
a_4	b_4	c_4	d_4	e_4
$\mathbf{p}_1^T\mathbf{a}$	$\mathbf{p}_1^T\mathbf{b}$	$\mathbf{p}_1^T\mathbf{c}$	$\mathbf{p}_1^T\mathbf{d}$	$\mathbf{p}_1^T\mathbf{e}$
$\mathbf{p}_2^T\mathbf{a}$	$\mathbf{p}_2^T\mathbf{b}$	$\mathbf{p}_2^T\mathbf{c}$	$\mathbf{p}_2^T\mathbf{d}+(a_1+a_3)$	$\mathbf{p}_2^T\mathbf{e}+(b_1+b_3)$
$\mathbf{p}_3^T\mathbf{a}$	$\mathbf{p}_3^T\mathbf{b}$	$\mathbf{p}_3^T\mathbf{c}$	$\mathbf{p}_3^T\mathbf{d}+(c_1+c_3)$	$\mathbf{p}_3^T\mathbf{e}+(a_2+a_4)$
$\mathbf{p}_4^T\mathbf{a}$	$\mathbf{p}_4^T\mathbf{b}$	$\mathbf{p}_4^T\mathbf{c}$	$\mathbf{p}_4^T\mathbf{d}+(b_2+b_4)$	$\mathbf{p}_4^T\mathbf{e}+(c_2+c_4)$

B. Analysis on repair bandwidth

Recall the construction of piggyback functions in Section.IV-A. If ks is not dividable by $(r-1)p$, the systematic symbols partitioned into the $(r-1)p$ piggyback functions are uneven. Here, we define the $(r-1)p$ sizes of these piggyback functions as the numbers of contained systematic symbols in Region A. Without loss of generality, assume the $(r-1)p$ sizes are not all the same, and denote them as $n_1, n_2, \cdots, n_{(r-1)p}$. Obviously, they satisfy that

$$\sum_{i=1}^{(r-1)p} n_i = ks. \tag{17}$$

Suppose that the l-th systematic node fails, $l \in \{1, \cdots, k\}$. All remaining symbols stored in Region B except node l are needed to reconstruct $\{a_{s+1,l}, \cdots, a_{s+p,l}\}$ with the MDS property. The amount transmitted in this step is kp symbols. In Region D, the s parity check symbols containing the piggyback functions of $\{a_{1,l}, \cdots, a_{s,l}\}$ are required to recover the s missing protected symbols. Moreover, the components along $\{\mathbf{a}_{s+1}, \cdots, \mathbf{a}_{s+p}\}$ should be subtracted out from the s downloaded parity check symbols. However, the left piggybacking functions are still involved with some other protected symbols besides $\{a_{1,l}, \cdots, a_{s,l}\}$. Hence, more symbols in Region A are needed. Assume the sizes of these s piggybacking functions are $n_{i_1}, n_{i_2}, \cdots, n_{i_s}$. The download amount of systematic symbols from Region A in this step is $(n_{i_1} + n_{i_2} + \cdots + n_{i_s} - s)$.

Now we derive the total bandwidth of repairing all the k systematic nodes. Symbols in Region

B need to be downloaded k^2p times. Consider a parity check symbol stored in Region D. Suppose the size of the piggybacking function embedded in this parity check symbol is n_i ($i \in \{1, \cdots, (r-1)p\}$). During the repair procedures, the parity check symbol needs to be downloaded n_i times. Meanwhile, each of the n_i involved systematic symbols in Region A needs to be downloaded $(n_i - 1)$ times. Therefore, the total repair bandwidth of all the k systematic nodes is $k^2p + \sum_{i=1}^{(r-1)p} n_i^2$.

From the above, the average repair ratio γ_2^{sys} is

$$\gamma_2^{sys} = \frac{1}{k^2(s+p)}(k^2p + \sum_{i=1}^{(r-1)p} n_i^2). \tag{18}$$

Rewrite Equation (18) as

$$
\begin{aligned}
\gamma_2^{sys} &= \frac{1}{k^2(s+p)}(k^2p + \sum_{i=1}^{(r-1)p} n_i^2) \\
&= \frac{1}{k^2(s+p)}\left[k^2p + \frac{(\sum_{i=1}^{(r-1)p} n_i)^2 + \sum_{i\neq j}(n_i - n_j)^2}{(r-1)p}\right] \\
&= \frac{1}{k^2(s+p)}\left[k^2p + \frac{k^2s^2 + \sum_{i\neq j}(n_i - n_j)^2}{(r-1)p}\right].
\end{aligned} \tag{19}
$$

Without loss of generality, assume ks is not dividable by $(r-1)p$, and

$$t_l' = \left\lfloor \frac{ks}{(r-1)p} \right\rfloor, \ t_h' = \left\lceil \frac{ks}{(r-1)p} \right\rceil, \ t' = ks - t_l'(r-1)p. \tag{20}$$

Thus, t' out of $(r-1)p$ piggyback functions have the size of t_h', and the rest $(r-1)p - t'$ ones have the size of t_l'. Then, γ_2^{sys} goes to

$$\gamma_2^{sys} = \frac{1}{k^2(s+p)}\left[k^2p + \frac{k^2s^2}{(r-1)p} + \frac{t'((r-1)p - t')}{(r-1)p}\right]. \tag{21}$$

In a DSS, the parameters of base code (k, r) are given. Thus, γ_2^{sys} is varied with different values of (s, p). In order to explore the relationship between γ_2^{sys} and the proportion of protected
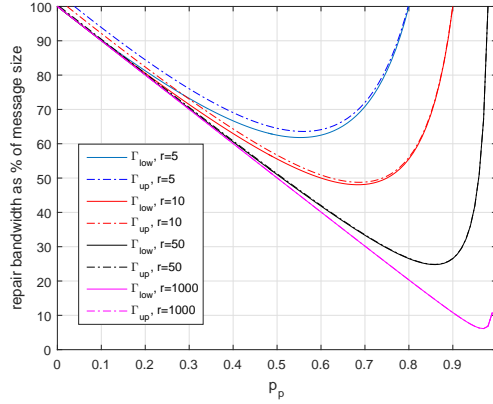
Fig. 4. The lower and upper bounds with various p_p.

instances $p_p = \frac{s}{s+p}$, the lower and upper bounds of γ_2^{sys} are derived as follows,

$$\gamma_2^{sys} \geq \frac{1}{k^2(s+p)}\left(k^2 p + \frac{k^2 s^2}{(r-1)p}\right)$$
$$= \frac{p}{s+p} + \frac{s}{s+p} \cdot \frac{s}{(r-1)p} \tag{22}$$

$$\gamma_2^{sys} \leq \frac{1}{k^2(s+p)}\left[k^2 p + \frac{k^2 s^2}{(r-1)p} + \frac{(r-1)p}{4}\right]$$
$$= \frac{p}{s+p}\left(1 + \frac{r-1}{4k^2}\right) + \frac{s}{s+p} \cdot \frac{s}{(r-1)p} \tag{23}$$

Rewrite the lower and upper bounds as functions $\Gamma_{low}(p_p)$ and $\Gamma_{up}(p_p)$ of p_p. Then,

$$\Gamma_{low}(p_p) = (1 - p_p) + \frac{p_p{}^2}{1 - p_p} \cdot \frac{1}{r-1} \tag{24}$$

$$\Gamma_{up}(p_p) = (1 - p_p)\left(1 + \frac{r-1}{4k^2}\right) + \frac{p_p{}^2}{1 - p_p} \cdot \frac{1}{r-1} \tag{25}$$

Example 2. *Assume the code rate of the base code is* 0.5*, i.e.,* $k = r$*. For various* r*'s, Figure 4 shows the curves of* $\Gamma_{low}(p_p)$ *and* $\Gamma_{up}(p_p)$ *with* p_p*.*

It illustrates that the lower bound $\Gamma_{low}(p_p)$ *and upper bound* $\Gamma_{up}(p_p)$ *are close to each other. Moreover, both of them can reach their extreme points by optimizing* p_p *which implies that the generalized piggybacking code can obtain optimum* γ_2^{sys} *with appropriate parameters* (s, p)*.*

Further analyze the optimum condition for γ_2^{sys} with the derivatives of $\Gamma_{low}(p_p)$ and $\Gamma_{up}(p_p)$, which are with respect to p_p and listed as follows

$$\frac{\partial \Gamma_{low}(p_p)}{\partial p_p} = \frac{-rp_p^2 + 2rp_p - (r-1)}{(r-1)(1-p_p)^2} \tag{26}$$

$$\frac{\partial \Gamma_{up}(p_p)}{\partial p_p} = \frac{-rp_p^2 + 2rp_p - (r-1)}{(r-1)(1-p_p)^2} - \frac{r-1}{4k^2}. \tag{27}$$

Let $\frac{\partial \Gamma_{low}(p_p)}{\partial p_p}$ and $\frac{\partial \Gamma_{up}(p_p)}{\partial p_p}$ equal to zero. Then, we work out the minimum values of $\Gamma_{low}(p_p)$ and $\Gamma_{up}(p_p)$ as follows,

1) $\min(\Gamma_{low}(p_p)) = \frac{2}{\sqrt{r}+1}$, when $p_p = 1 - \frac{1}{\sqrt{r}}$;

2) $\min(\Gamma_{up}(p_p)) = \frac{-2+2\sqrt{r+\frac{(r-1)^2}{4k^2}}}{r-1}$, when $p_p = 1 - \frac{1}{\sqrt{r+\frac{(r-1)^2}{4k^2}}}$.

The results indicate that

1) $\min(\Gamma_{low}(p_p))$ is only determined by the number of parity check nodes r;

2) $\min(\Gamma_{up}(p_p))$ is determined by both k and r. However, for high code rate, $\min(\Gamma_{up}(p_p))$ is dominantly determined by r;

3) $\min(\Gamma_{up}(p_p))$ corresponds closely to $\min(\Gamma_{low}(p_p))$. In other words, there exists a generalized piggybacking code whose repair ratio is very close to the lower bound.

Figure 5 shows the curves of $\min(\Gamma_{low}(p_p))$ and $\min(\Gamma_{up}(p_p))$ with r. It implies that

$$\min(\gamma_2^{sys}) \approx \min(\Gamma_{low}(p_p)) = \frac{2}{\sqrt{r}+1}. \tag{28}$$

At the end of this subsection, we perform asymptotic analyses of $\min(\gamma_1^{sys})$ and $\min(\gamma_2^{sys})$, and compare them with the repair ratio of *minimum storage regenerating (MSR)* codes γ_{MSR}. The limits of $\min(\gamma_1^{sys})$ and $\min(\gamma_2^{sys})$ as r approaches infinity are

$$\lim_{r\to+\infty} \min(\gamma_1^{sys}) = \lim_{r\to+\infty} \frac{r-1}{2r-3} = 0.5 \tag{29}$$

$$\lim_{r\to+\infty} \min(\gamma_2^{sys}) = \lim_{r\to+\infty} \frac{2}{\sqrt{r}+1} = 0. \tag{30}$$

As described in [9], [8], [25], MSR codes which correspond to the best storage efficiency are one of two most important classes of regenerating codes. The repair bandwidth for one failure
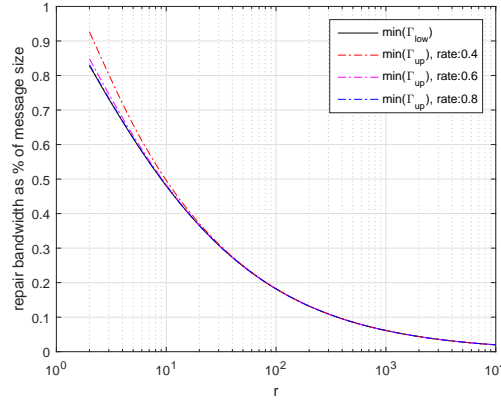
Fig. 5. Minimum values of γ_1^{sys} and γ_2^{sys}.

node is

$$B_{MSR} = \frac{\mathcal{M}d}{k(d-k+1)}, \tag{31}$$

where $\mathcal{M}$ represents the size of original messages, d denotes the number of accessed surviving nodes, and k is the dimension of the MSR code. For the sake of simple comparison, we set the code rate to 0.5, and $d = n - 1$ such that the MSR code provides the highest repair efficiency. Thus,

$$\gamma_{MSR} = \frac{B_{MSR}}{\mathcal{M}} = \frac{2}{r} - \frac{1}{r^2}. \tag{32}$$

The curves of $\min\left(\gamma_1^{sys}\right)$, $\min\left(\gamma_2^{sys}\right)$ and γ_{MSR} are shown in Fig.6. It shows that $\min\left(\gamma_2^{sys}\right)$ approaches zero instead of 50% as the number of parity check nodes tends to infinity. As a result, compared with RSR-II codes, generalized piggybacking codes are able to provide more efficient node repair with less bandwidth. Moreover, $\min\left(\gamma_2^{sys}\right)$ is closer to γ_{MSR} - the theoretical lower bound of repair ratio.

Table I compares the repair efficiency of RSR-II codes and generalized piggybacking codes with various code parameters n and k. It is illustrated that with the increasing of the number of parity check nodes, generalized piggybacking codes can reach smaller repair bandwidth.
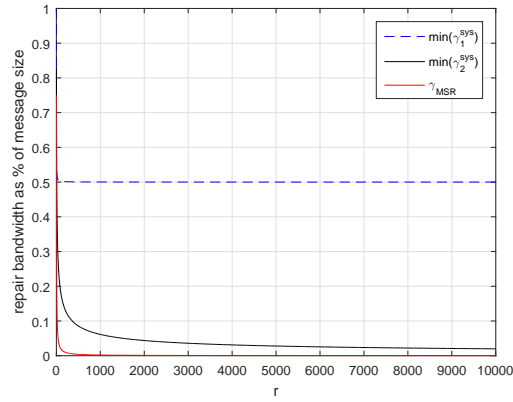
Fig. 6.   Lower bounds on the average repair bandwidths.

TABLE I

EFFICIENCY COMPARISON FOR DIFFERENT EXPLICIT CODES

n, k	RSR-II codes		generalized piggybacking codes		
	stripes	γ_1^{sys}	s, p	stripes	γ_2^{sys}
$10, 5$	7	0.5886	1, 1	2	0.6400
$20, 10$	17	0.5341	2, 1	3	0.4867
$30, 15$	27	0.5207	3, 1	4	0.4133
$40, 20$	37	0.5147	4, 1	5	0.3700
$50, 25$	47	0.5114	4, 1	5	0.3344
$80, 40$	77	0.5068	5, 1	6	0.2740
$200, 100$	197	0.5026	9, 1	10	0.1819

C. Analysis on decoding complexity

In this subsection, the complexity of node repair procedure of generalized piggybacking codes is analyzed first. Then the comparison with RSR-II codes is performed. It is shown that the computational complexity for repairing a single systematic node cost by generalized piggybacking codes is much less than that of RSR-II codes.

As the statement in Section.III and IV-B, piggybacking codes adopt two kinds of calculations

to repair a failed node. MDS decoding is used for the recovery of the missing symbols in non-protected or piggybacked stripes, while solving linear combinations is employed to reconstruct the missing symbols in protected stripes. Recall the generalized piggybacking code, in Section.IV-A, which has s protected stripes and p piggybacked stripes. The repair procedure of the l-th systematic node is described in Section.IV-B.

In order to recover $a_{s+i,l}$ - the missing symbol of the i-th piggybacked stripe, the symbols $\{a_{s+i,1}, \cdots, a_{s+i,l-1}, a_{s+i,l+1}, \cdots, a_{s+i,k}, \mathbf{p}_1^T \mathbf{a}_{s+i}\}$ are required. Denote the vector representation of $\mathbf{p}_j$ ($j \in \{1, \cdots, r\}$) as $[p_{j,1}, \cdots, p_{j,k}]$. Then, $a_{s+i,l}$ can be worked out by the below equation.

$$
\begin{aligned}
a_{s+i,l} = \ & p_{j,l}^{-1} [\mathbf{p}_1^T \mathbf{a}_{s+i} - (a_{s+i,1} p_{j,1} + \cdots + \\
& a_{s+i,l-1} p_{j,l-1} + a_{s+i,l+1} p_{j,l+1} + \cdots + \\
& a_{s+i,k} p_{j,k})].
\end{aligned}
\tag{33}
$$

Hence, the MDS decoding for the recovery of one missing symbol in a piggybacked stripe costs k multiplications and $(k-1)$ additions.

Consider the recovery of $a_{i,l}$ - the missing symbol in the i-th protected stripe. According to the description of Section.IV-A, we denote the piggyback function which involves $a_{i,l}$ together with other $(n_x - 1)$ protected symbols as F_x. In order to reconstruct $a_{i,l}$, from Region D, the stored symbol ξ containing F_x is needed, and the $(n_x - 1)$ surviving protected symbols are also required. Hence, $a_{i,l}$ can be figured out as follows.

- Compute the parity check symbol in ξ. This step costs k multiplications and $(k-1)$ additions.
- Subtract the parity check symbol from ξ. Thus, 1 addition is needed.
- Subtract the $(n_x - 1)$ surviving protected symbols form the left F_x. Thus, $(n_x - 1)$ additions are required.

Actually, n_x represents the size of the piggyback function F_x, i.e., n_x equals to t'_l or t'_h. Therefore, solving linear combinations for one missing protected symbol costs k multiplications and $\frac{ks}{(r-1)p} + k - 1$ additions, on average.

TABLE II

COMPUTATIONAL COMPLEXITY FOR NODE REPAIR

	Multiplications	Additions
MDS decoding	k	$(k-1)$
Solving linear combinations	k	$\frac{ks}{(r-1)p} + k - 1$

The computational complexity of MDS decoding and solving linear combinations is listed in Table II.

According the analysis in Section.III, solving linear combinations is introduced by piggybacking codes to reduce the repair bandwidth of partial missing symbols. For RSR-II codes, $(r-1)$ missing protected symbols need to be simultaneously recovered by solving a group of $(r-1)$ linear functions. As a result, we have to perform Gaussian elimination. However, for generalized piggybacking codes, piggyback functions are simple summations of some protected symbols. Compared with the calculations for MDS decoding, those for solving linear combinations cost only $\frac{ks}{(r-1)p}$ more additions. Thus, the generalized piggybacking framework is able to provide high repair efficiency because it can significantly reduce the repair bandwidth for a single failed systematic node with low computational complexity.

V. CONCLUSION AND DISCUSSION

This paper presents a generalized piggybacking construction with various protected instances and piggybacked instances. Compared with the previous design, our proposed generalized piggybacking codes can save more repair bandwidth by optimizing the proportion of protected instances. When the number of parity check nodes tends to infinity, the average repair bandwidth as a fraction of total messages approaches zero. Moreover, complexity analysis demonstrates that generalized piggybacking codes are able to efficiently repair the failed node with reasonable

complexity overhead.

In fact, if we look at piggybacking functions from the view of error-correction codes, piggy-backing codes are perfect encounter between codes with small minimum Hamming distance and codes with large minimum Hamming distance. The repair of systematic symbols in piggybacked stripes is relied on the base codes of these stripes. These base codes have strong erasure-correction capability due to their large minimum distance. However, it results in strong correlation among all the symbols. Thus, decoding of these good codes requests large amount of data access. For the repair of protected stripes, piggybacking functions are linear combinations of the protected systematic symbols. In other words, these symbols together with piggyback functions can be considered as linear codes with small minimum distance. Since these bad codes have weak correlation among symbols, their decoding requests small amount of data access.

VI. ACKNOWLEDGMENT

REFERENCES

[1] S. Ghemawat, H. Gobioff and S.-T. Leung, "The Google file system", in *Proc. ACM SIGOPS operating systems review*, vol. 37, no. 5, 2003, pp. 2943.

[2] D. Borthakur, "Hdfs architecture guide," 2008. [Online]. Available: http://hadoop.apache.org/common/docs/current/hdfs design.pdf

[3] H. Weatherspoon and J. D. Kubiatowicz, "Erasure coding vs. replication: A quantitative comparison," in *Proc. Peer-to-Peer Systems(IPTPS)*, 2002, pp. 328337.

[4] S. C. Rhea, P. R. Eaton, D. Geels, H. Weatherspoon, B. Y. Zhao, and J. Kubiatowicz, "Pond: The oceanstore prototype," in *Proc. 2nd USENIX Conf. File and Storage Technologies(FAST)*, 2003, pp. 114.

[5] R. Bhagwan, K. Tati, Y. Cheng, S. Savage, and G. M. Voelker, "Total recall: System support for automated availability management," in *Proc. 1st Conf. Networked Systems Design and Implementation(NSDI)*, 2004, pp. 2525.

[6] B. Calder, J. Wang, A. Ogus, N. Nilakantan, A. Skjolsvold, S. McKelvie, Y. Xu, S. Srivastav, J. Wu, H. Simitci et al., "Windows azure storage: a highly available cloud storage service with strong consistency," in *Proc. 23rd ACM Symposium on Operating Systems Principles*, 2011, pp. 143157.

[7] "Google-gfs2 colossus," 2012. [Online]. Available: http://www.quora.com/Colossus-Google-GFS2.

[8] A. G. Dimakis, K. Ramchandran, Y. Wu, and C. Suh, "A survey on network codes for distributed storage," *Proceedings of the IEEE*, vol. 99, no. 3, pp. 476489, 2011.

[9] A. G. Dimakis, P. Godfrey, Y. Wu, M. J. Wainwright, and K. Ramchandran, "Network coding for distributed storage systems," *IEEE Trans. Inf. Theory*, vol. 56, no. 9, pp. 45394551, 2010.

[10] D. Cullina, A. G. Dimakis, and T. Ho, "Searching for minimum storage regenerating codes," in *Proc. 47th Annu. Allerton Conf. Commun., Control, Comput.*, 2009.

[11] K. V. Rashmi, N. B. Shah, P. V. Kumar, and K. Ramchandran, "Explicit construction of optimal exact regenerating codes for distributed storage," in Proc. 47th Annu. Allerton Conf. Commun., Control, Comput., 2009, pp. 12431249.

[12] C. Suh and K. Ramchandran, "Exact-repair MDS codes for distributed storage using interference alignment," in *Proc. IEEE Int. Symp. Inf. Theory*, 2010, pp. 161165.

[13] K. V. Rashmi, N. B. Shah, and P. V. Kumar, "Optimal exact-regenerating codes for distributed storage at the MSR and MBR points via a product-matrix construction," *IEEE Trans. Inf. Theory*, vol. 57, no. 8, pp. 52275239, 2011.

[14] J. Li, X. Tang, and U. Parampalli, "A framework of constructions of minimal storage regenerating codes with the optimal access/update property," *IEEE Trans. Inf. Theory*, vol. 61, no. 4, pp. 19201932, 2015.

[15] N. B. Shah, K. V. Rashmi, P. V. Kumar, and R. Kannan, "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. 18371852, 2012.

[16] M. Blaum, J. Brady, J. Bruck, and J. Menon, "EVENODD: An efficient scheme for tolerating double disk failures in RAID architectures," *IEEE Trans. Computers*, vol. 44, no. 2, pp. 192202, 1995.

[17] 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. 18171826, 1999.

[18] L. Xu and J. Bruck, "X-code: MDS array codes with optimal encoding," *IEEE Trans. Inf. Theory*, vol. 45, no. 1, pp. 272276, 1999.

[19] 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 Conference on File and Storage Technologies(FAST)*, 2004.

[20] C. Huang and L. Xu, "STAR: An efficient coding scheme for correcting triple storage node failures," *IEEE Trans. Computers*, vol. 57, no. 7, pp. 889901, 2008.

[21] I. Tamo, Z. Wang, and J. Bruck, "Zigzag codes: MDS array codes with optimal rebuilding," *IEEE Trans. Inf. Theory*, vol. 59, no. 3, pp. 15971616, 2013.

[22] K. 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*, 2013, pp. 331335.

[23] K. Rashmi, N. B. Shah, and K. Ramchandran, "A piggybacking design framework for read-and download-efficient distributed storage codes," 2013. [Online]. Available: http://arxiv.org/pdf/1302.5872.pdf

[24] K. V. Rashmi, N. B. Shah, G. Dikang, K. Hairong, B. Dhruba, and R. Kannan, "A solution to the network challenges of data recovery in erasure-coded distributed storage systems: A study on the facebookwarehouse cluster," in *Presented as part of the 5th USENIX Workshop on Hot Topics in Storage and File System*, 2013.

[25] B. Yang and X. Tang, "A systematic piggybacking design for minimum storage regenerating codes," *IEEE Trans. Inf. Theory*, vol. 61, no. 11, pp. 57795786, 2015.