SCIENCE CHINA
Information Sciences

• RESEARCH PAPER •

December 2022, Vol. 65 222303:1–222303:17
https://doi.org/10.1007/s11432-021-3421-6

Scalable local reconstruction code design for hot data reads in cloud storage systems

Zhikai ZHANG[1], Shushi GU[1,2*] & Qinyu ZHANG[1,2*]

[1]*School of Electronic and Information Engineering, Harbin Institute of Technology* (*Shenzhen*), *Shenzhen* 518055, *China;*
[2]*Peng Cheng Laboratory, Shenzhen* 518055, *China*

Received 19 August 2021/Revised 26 November 2021/Accepted 26 January 2022/Published online 22 November 2022

Abstract Since demand for data is significantly heterogeneous in cloud storage systems (CSSs), there is traffic congestion in nodes storing hot data. In erasure-coded CSSs, traffic congestion can be alleviated by degraded reads sacrificing the bandwidth of surviving nodes. Local reconstruction codes (LRCs) reduce the bandwidth consumption of degraded reads, but cannot provide skewed throughput gain for the hot data. In this paper, we propose a scalable local reconstruction code (SLRC) that relies on LRCs but is more flexible in improving the throughput of a specific data block. First, we develop the local maximum throughput (LMT) to measure the maximum throughput of the hot data blocks by analyzing the actual read arrival rate of LRCs. Further, we elaborate on the structure of SLRC and analyze their performance metrics, which include storage overhead, reconstruction cost, and LMT. To select the appropriate code, we present the minimum reconstruction cost, minimum storage overhead, and minimum penalty algorithms. Finally, we implement extensive experiments on several typical SLRCs on the Hadoop distributed file system. Higher LMT and lower bandwidth consumption can be provided by SLRCs for hot data block degraded reads in CSSs compared with RS codes and LRCs.

Keywords cloud storage systems, erasure code, scalable local reconstruction code, degraded reads, local maximum throughput

Citation Zhang Z K, Gu S S, Zhang Q Y. Scalable local reconstruction code design for hot data reads in cloud storage systems. Sci China Inf Sci, 2022, 65(12): 222303, https://doi.org/10.1007/s11432-021-3421-6

1 Introduction

Data popularity is a variant in the cloud storage system (CSS). An investigation of traces from Yahoo's Druid cluster reflects that the top 1% of data is requested more frequently than the bottom 40% [1, 2]. The most popular data in a certain social network is referred to as hot data [3] such as system overload and traffic congestion [4, 5], which can cause critical problems in CSSs. In contrast, some data stored in CSSs is rarely requested; they are referred to as cold data [6], which consumes few system resources. When the demands exceed the service capacity of the node storing the hot data, temporary failure will occur, resulting in system status degeneration [7]. In the practical CSSs, the proportion of temporary failures generated from hot data may achieve approximately 90% [8].

CSSs use replications to tackle data failures [9]. Conventional n-replication requires expensive storage resources for facing exponential data growth [10]. Here, erasure codes have been adopted by several enterprise-level storage systems for data fault-tolerance because of their low storage overhead, e.g., Google GFSII [11], Windows Azure [12], and Facebook [13]. In an erasure-coded storage system, degraded reads are employed to alleviate the effects of temporary failures. When the data cannot be read directly from a failed node, the required data will be reconstructed by reading the encoded data blocks from a certain number of other helper nodes [14]; this is referred to as degraded reads. Unfortunately, the degraded reads for hot data sacrifice a substantial amount of bandwidth of surviving nodes storing cold data, which leads to extra system burden [15]. Reed-Solomon (RS) (k, r) codes, for instance, have become a common choice because of their maximum distance separable (MDS) property [16]. However, RS codes must read

* Corresponding author (email: gushushi@hit.edu.cn, zqy@hit.edu.cn)

Table 1 Parameters summary

Parameter	Description	Parameter	Description
k	The number of data blocks	l	The number of local parity blocks
r	The number of global parity blocks	l_0	The scalable index of the SLRC
λ_i^{d}	The DRAR of the i-th data block	$\lambda_i^{\mathrm{d}\prime}$	The ARAR of the i-th data block
$\lambda_i^{\mathrm{lp}\prime}$	The ARAR of the i-th local parity block	$\lambda_i^{\mathrm{gp}\prime}$	The ARAR of the i-th global parity block
λ^{d}	The DRAR of one data block	$\lambda^{\mathrm{d}\prime}$	The ARAR of one data block
$\lambda^{\mathrm{lp}\prime}$	The ARAR of one local parity block	$\lambda^{\mathrm{gp}\prime}$	The ARAR of one global parity block
λ^{th}	The threshold value of determining the hot data	p_i	The proportion of the i-th block direct read
p_i^{l}	The proportion of the i-th block degraded read through a local repairable group	p_i^{g}	The proportion of the i-th block degrade read through a global repairable group
$q_{i,j}^{\mathrm{l}}$	The probability that the i-th data block is part of the j-th data block's degraded read through a local repairable group	$q_{i,j}^{\mathrm{g}}$	The probability that the i-th data block is part of the j-th data block's degraded read through a global repairable group
$q_{i,j}^{\mathrm{lp}}$	The probability that the i-th local parity block is part of the j-th data block's degraded read through a local repairable group	$q_{i,j}^{\mathrm{gp}}$	The probability thet the i-th global parity block is part of the j-th data block's degraded read through a global repairable group
ρ^{th}	The ratio of λ^{th} to the same DRAR of all cold data blocks	μ	The rate of the exponential distribution that service times at the servers follow
λ_i	The rate of read requests for each data block each unit time follows	ρ	The ratio of the rate of the hot data block and cold data block

and download all k blocks of a file from n $(n = k + r)$ surviving nodes (disks) for data recovery, although only one block failure occurs [17]. Thus, reducing the bandwidth consumption becomes critical in the issue of designing erasure codes for CSSs.

Local reconstruction code (LRC) (k, l, r), which was proposed by Huang et al. [18], can divide k data blocks into l repairable groups, reducing the number of data blocks needed for reconstruction. Theoretically, LRCs can reduce their repair bandwidth of one data block to $1/l$ of the corresponding RS codes, making them preferable to achieve efficiently degraded reads for temporary failures. Although LRCs have the advantage of requiring fewer helper nodes compared with RS codes, they treat the data as the same and cannot offer differentiated protection between hot and cold data blocks. LRCs cannot provide higher throughput or lower repair bandwidth for hot data than cold data. Thus, a flexible and adjustable approach to redesign LRCs must improve the degraded read performance of hot data.

In this paper, we propose a scalable local reconstruction code (SLRC) to address the traffic congestion caused by requests for hot data in CSSs. Unlike existing studies, the SLRC adds a scalable index to adjust the sizes of local repairable groups, based on the difference between hot and cold data as measured by the direct read arrival rate (DRAR). We also designed the SLRC structure to reduce the required network bandwidth for degraded reads of hot data and alleviate the loading burden. Further, degraded reads of the SLRCs can improve the local throughput of the hot data block. The main contributions of this paper are summarized as follows.

- We analyze the load and throughput of the LRCs and attempt to reduce the system load during degraded reads. Further, we investigate a new metric called the local maximum throughput (LMT) to measure the maximum throughput of a single data block that the LRC can support.

- We design the SLRCs to provide different maximum throughputs for different data blocks and show that the proposed SLRC can improve the hot data block LMT. Further, we develop three code selection algorithms to determine the appropriate scalable index.

- We implement SLRC on the HDFS-EC platform, which runs in a cluster of 15 virtual machines. Further, we configure RS codes, LRCs, and SLRCs in HDFS-EC and evaluate the LMT of the hot block. The results indicate that, with the same k and l, SLRCs have $l_0 \times \lambda^{\mathrm{th}}$ more LMT of the hot data blocks than LRCs without more penalty. For SLRCs, increasing l_0 improves the LMT of the hot block while decreasing the penalty.

The remainder of this paper is organized as follows: Section 2 presents the related work. Section 3 analyzes the degraded read of the LRC. Section 4 proposes the definition and structure of SLRC. Section 5 elaborates the SLRC performance analysis and code selection algorithms. Section 6 presents the experiment and comparison results. Section 7 concludes with final remarks. Table 1 lists the parameters used in this study.

2 Related work

The studies on erasure codes, degraded read, and system mechanisms are discussed below. Further, we give our motivation for this study.

2.1 Erasure codes

RS codes are the most commonly used codes in practical systems [16]. However, they require as much coding data as the original data in the data reconstruction. To address this, much research has focused on reducing the data transferred to relieve the increased stress on the network during the data reconstruction. The LRCs proposed in [18] reduce the number of data blocks within each repairable group to meet the disk I/O and network bandwidth requirements, which is important for optimizing the performance in case of temporary failure. Dimakis et al. [19] proposed regenerating codes (RCs) that focus on reducing the amount of downloaded data during data repairing. RCs are a class of erasure codes derived from network coding that can reduce the repair bandwidth by combining data linearly and then uploading it from surrounding nodes. Tang et al. [20] proposed a new simple optimal repair strategy of Hadamard minimum storage regenerating codes for RCs, which can considerably reduce the computation during the node repair compared with the original. Additionally, Tang et al. [21, 22] studied the systematic piggybacking design and the optimal access/update property of RCs.

The studies above focus on developing new types of erasure codes to relieve the resource burden of systems. Further, attempts are being made to modify the existing erasure codes used in storage systems. A significant trend is to modify the encoding algorithms. Rashmi et al. [23] proposed the Hitchhiker system that defines the new encoding and decoding algorithms based on the RS codes, which are referred to as Piggybacked-RS codes, to reduce the amount of data transmission in the reconstruction process by matching the function data in the node. The authors [24] reduced the number of I/O required to link by changing data placement and coding rules.

2.2 Degraded read

Degraded read means that, if some data blocks are temporarily inaccessible, in the coded storage system, the system reads the data from the surviving nodes and reconstructs it [25]. Horizontal-Vertical code is proposed in [26], which is designed to reduce I/O in degraded reads by shortening the parity chain length and placing sequential data elements on horizontal parity chains. Li et al. [27] arranged for the degraded reads in map tasks to avoid bandwidth competition and improve the MapReduce performance. Khan et al. [28] proposed the rotating RS codes, which inherit the reliability and performance properties of RS codes while performing degraded reads more efficiently. The authors proposed a new code called the EC-FRM-Code in [29], which can improve the reading speed of normal and degraded reads. The EC-FRM-Code is implemented based on the RS codes and LRCs.

Some researches focus on improving degraded read performance based on system heterogeneity. The authors [30] also considered the arrival rates of different files and proposed a novel caching framework that meets the requirements of reduced latency. Zhang et al. [31] proposed an optimization scheme that uses the node evaluation method and distance calculation to optimize degraded reads while considering the dynamic heterogeneity of the current server busy level. The authors in [32] identified the capacity of different links in the storage cluster to avoid data transmission through the congested links.

2.3 System mechanism

The system can improve performance by choosing an appropriate caching strategy (selecting data for caching) [33]. There are also some studies on system mechanisms for degraded reads, such as parallel processing and active caching. Li et al. [34] proposed the collective reconstruction read (CRR) method for parallel processing, which uses parallel reconstruction to reduce the read delay. Data reading and decoding in CRR are shared in parallel among all participating nodes, reducing the time complexity of degraded reads. Further, recent studies have focused on active recovery techniques. The author in [35] proposed deploying a proactive cache based on an active caching technique that actively copies objects from the fault prediction machine to the cache layer. The system uses various failure prediction methods to predict hardware failure. An accurate failure prediction and proactive caching can minimize read latency, reduce degraded reads, and improve throughput.

2.4 Design motivation

From the above-mentioned analysis, we observed that a small amount of hot data in CSS occupies huge requests, which causes temporary failure of the system. In the erasure-coded CSSs, temporary failure will be resolved by degraded reads. In the existing literature, the research on erasure codes focuses on reducing the bandwidth consumption during the data reconstruction but not on how to increase the throughput. Further, some schemas are concerned with the impact of system heterogeneity in degraded reads, but are less concerned with the heterogeneity of hot and cold data. Our work in this paper is based on the LRC and focuses on the heterogeneity of hot and cold data. Our primary concern is how to improve the LMT of hot data by redesigning the local group's structure of LRCs.

3 Analysis of local reconstruction code

3.1 Degraded reads of local reconstruction code

In order to analyze the traffic load variations during degraded reads, we adopt the indices used in [36]. To quantify the number of read requests per block of data per unit time, we define the DRAR. Normally, the system retrieves content by direct read, and hot blocks have more read requests than their counterparts. It is assumed that each data block stores data and has different DRARs. There is no direct read of global parity blocks or local parity blocks, so the DRARs of the parity blocks are zero. The DRAR of the i-th data block is defined as λ_i, which indicates the number of direct read requests of the i-th data block per unit time, expressed in λ_i^{d}, where $i = 1, 2, \ldots, k$.

The actual read arrival rate (ARAR) is defined as the number of read requests per second of the i-th data block after degraded reads, i.e., $\lambda_i^{\mathrm{d}\prime}$, where $i = 1, 2, \ldots, k$. Due to the need to help degraded read, parity block also has the actual read arrival rate, ARAR of the i-th local parity block is represented by $\lambda_i^{\mathrm{lp}\prime}$, while the ARAR of the i-th global parity block is represented by $\lambda_i^{\mathrm{gp}\prime}$.

Now, we define some parameters to represent these three parts. Let p_i denote the proportion of the i-th block direct read, p_i^{l} denote the proportion of the i-th block degraded read through the local repairable group, and p_i^{g} denote the proportion of the i-th block degraded read through global repairable group. For the degraded reading of the i-th data block, we define the probabilities that other $k-1$ blocks are part in this task through the local repairable group by $\{q_{1,i}^{\mathrm{l}}, q_{2,i}^{\mathrm{l}}, \ldots, q_{i-1,i}^{\mathrm{l}}, q_{i+1,i}^{\mathrm{l}}, \ldots, q_{k,i}^{\mathrm{l}}\}$. The corresponding values for the global repairable group are defined as $\{q_{1,i}^{\mathrm{g}}, q_{2,i}^{\mathrm{g}}, \ldots, q_{i-1,i}^{\mathrm{g}}, q_{i+1,i}^{\mathrm{g}}, \ldots, q_{k,i}^{\mathrm{g}}\}$. The probabilities that l local parity blocks are involved in this task are defined as $\{q_{1,i}^{\mathrm{lp}}, q_{2,i}^{\mathrm{lp}}, \ldots, q_{l,i}^{\mathrm{lp}}\}$. And the probabilities that all r global parity blocks in this task are defined as $\{q_{1,i}^{\mathrm{gp}}, q_{2,i}^{\mathrm{gp}}, \ldots, q_{r,i}^{\mathrm{gp}}\}$. One local parity block is the only part in the degraded reading of the data blocks in its group, so for other blocks the value is zero.

3.2 Actual read arrival rate of local reconstruction code

Based on the parameters defined above, we can formulate the ARAR of the i-th data block $\lambda_i^{\mathrm{d}\prime}$, which can be expressed as[1)]

$$\lambda_i^{\mathrm{d}\prime} = p_i^{\mathrm{d}}\lambda_i + \left(\sum_{m=i+1}^{i+k/l-1} p_m^{\mathrm{l}} q_{i,m}^{\mathrm{l}} \lambda_m^{\mathrm{d}}\right) + \left(\sum_{m=1}^{k} p_m^{\mathrm{g}} q_{i,m}^{\mathrm{g}} \lambda_m^{\mathrm{d}} - p_i^{\mathrm{g}} q_{i,i}^{\mathrm{g}} \lambda_i^{\mathrm{d}}\right), \text{ where } i = 1, 2, \ldots, k. \tag{1}$$

Eq. (1) is written directly from the components of ARAR. As we can see, the ARAR of the hot block is added in three parts.

(1) The first part, $p_i^{\mathrm{d}}\lambda_i$, is the DRAR of the i-th data block.

(2) The second part, $(\sum_{m=i+1}^{i+k/l-1} p_m^{\mathrm{l}} q_{i,m}^{\mathrm{l}} \lambda_m^{\mathrm{d}})$, is the read arrival rate generated by local repairable group.

(3) The third part, $(\sum_{m=1}^{k} p_m^{\mathrm{g}} q_{i,m}^{\mathrm{g}} \lambda_m^{\mathrm{d}} - p_i^{\mathrm{g}} q_{i,i}^{\mathrm{g}} \lambda_i^{\mathrm{d}})$, is the read arrival rate generated by global repairable group.

1) For convenience, we define the other data blocks in the same local repairable group as the i-th data block by the $(i+1)$-th data block, ... , and $(i+k/l-1)$-th data block.

Then, the j-th local parity block in the local repairable group that needs to participate in degraded read is equal to the sum of degraded read tasks incurred by data blocks in this group, i.e.,

$$\lambda_i^{\mathrm{lp}\prime} = \sum_{m=1}^{k/l} p_m^{\mathrm{l}} q_{i,m}^{\mathrm{lp}} \lambda_m^{\mathrm{d}}, \text{ where } i = 1, 2, \ldots, l. \tag{2}$$

For the LRC, there is only one local parity block in each local repairable group, which means that any degraded read from a block through a local repairable group requires the help of all the other blocks in the local repairable group. So $q_{i,m}^{\mathrm{lp}}$ is set to one.

The ARAR of the j-th global parity block is expressed as the sum of the tasks undertaken by the degraded reads of all data blocks, which is

$$\lambda_i^{\mathrm{gp}\prime} = \sum_{m=1}^{k} p_m^{\mathrm{g}} q_{i,m}^{\mathrm{gp}} \lambda_m^{\mathrm{d}}, \text{ where } i = 1, 2, \ldots, r. \tag{3}$$

For the global repairable group of the LRC, the situation is similar to the RS code. In each block, the degraded reads that can be realized by the global repairable group require any k blocks from the other $k + r - 1$ blocks, so $q_{i,m}^{\mathrm{gp}}$ is $k/(k + r - 1)$.

How to perform degraded reading between data blocks is determined by the encoding structure and not affected by the DRARs of the data blocks. Let us consider a simple homogeneous case to briefly analyze the impact of the degraded reads of the LRC. Considering that each data block's DRAR is the same, the subscription notation in (1)–(3) can be eliminated and Eqs. (1)–(3) reduce to

$$\lambda^{\mathrm{d}\prime} = p\lambda^{\mathrm{d}} + (k/l - 1)p^{\mathrm{l}}\lambda^{\mathrm{d}} + (k - 1)\frac{k}{k + r - 1}p^{\mathrm{g}}\lambda^{\mathrm{d}}, \tag{4}$$

$$\lambda^{\mathrm{lp}\prime} = (k/l)p^{\mathrm{l}}\lambda^{\mathrm{d}}, \tag{5}$$

$$\lambda^{\mathrm{gp}\prime} = k\frac{k}{k + r - 1}p^{\mathrm{g}}\lambda^{\mathrm{d}}. \tag{6}$$

Theorem 1. In an LRC(k, l, r) storage system under the homogeneous condition, the system load of all nodes after degraded reads can be shown as

$$k\lambda^{\mathrm{d}} \leqslant k\lambda^{\mathrm{d}} + [(k/l - 1)p^l + (k - 1)p^g]k\lambda^{\mathrm{d}}. \tag{7}$$

Proof. In homogeneous LRC storage systems, the each part's load after the degraded read is measured as (4)–(6). Considering the number of each node, the total system load can be obtained as $k\lambda^d \leqslant [p + kp^l/l + kp^g]k\lambda^{\mathrm{d}}$. Since $p + p^{\mathrm{l}} + p^{\mathrm{g}} = 1$, we can get (7).

Theorem 2. In an LRC(k, l, r) storage system under the homogeneous condition, the load of each node after degraded reads can be shown as (8). And when Eq. (9) is satisfied, the node load is reduced after degraded reads.

$$\lambda^{\mathrm{d}\prime} = \left[(k/l - 2)p^l + \frac{(k - 1)^2 - r}{k + r - 1}p^g\right]\lambda^{\mathrm{d}}, \tag{8}$$

$$(k/l - 2)p^l + \frac{(k - 1)^2 - r}{k + r - 1}p^g < 1. \tag{9}$$

Proof. When $\lambda^{\mathrm{d}\prime} < \lambda^{\mathrm{d}}$, the ARAR of the data block node after degraded read is less than the DRAR. Substituting $p + p^{\mathrm{l}} + p^{\mathrm{g}} = 1$ into (4) can obtain the result in (8). Then, when Eq. (9) is satisfied, degraded read can reduce the node of a single data block.

From the analysis of (9), it can be seen that the k/l in (9) is considered first, which represents the number of data blocks in each local repairable group, and reducing it can help reduce the load.

And $\frac{(k-1)^2-r}{k+r-1}$ is less than zero, only when $(k - 1)(k - 2) < 2r$. It can be seen that for a general value of k, the magnitude of r must reach a level k^2, where it is difficult to simply increase the value of r to reduce the additional load of a global parity group's degraded read.

It can be concluded as follows. (1) By reducing the number of data blocks in each local repairable group, the total system load and the individual data node load after degraded read can be reduced.

(2) Reducing the values of p^l and p^g helps to reduce the load of a single node and all nodes. And because of the general k value, the size of r would need to be an order of k^2 magnitude to reduce the additional load of a global parity group's degraded read. Therefore, we should consider reducing this portion of the load by the reduced p^g. Eq. (1) can be converted to

$$\lambda_i^{\mathrm{d}\prime} = p_i\lambda_i^{\mathrm{d}} + \left(\sum_{m=i+1}^{i+k/l-1} p_m^{\mathrm{l}} q_{i,m}^{\mathrm{l}} \lambda_m^{\mathrm{d}} + p_m^{\mathrm{g}} q_{i,m}^{\mathrm{g}} \lambda_m^{\mathrm{d}}\right) + \left(\sum_{m=1}^{k} p_m^{\mathrm{g}} q_{i,m}^{\mathrm{g}} \lambda_m^{\mathrm{d}} - \sum_{m=i}^{i+k/l-1} p_m^{\mathrm{g}} q_{i,m}^{\mathrm{g}} \lambda_m^{\mathrm{d}}\right), \tag{10}$$

where $i = 1, 2, \ldots, k$. For a node in which a data block is located, its ARAR consists of three parts, namely the direct read of the data block, the degraded read of the same group of data blocks, and the degraded read of the data blocks outside the group. If the degraded read of one data block is through a local or global repairable group, it will increase the load on the other data blocks in the same group. The local and global repairable groups contain some same data blocks. Since both the local and global repairable groups need to use these blocks, it is bound to double the load of them. At the same time, due to the limited threshold for the load of these data blocks, the degraded reads of two repairable groups are the same as that of a single repairable group.

3.3 Definition of local maximum throughput

In a practical CSS, the DRAR of the data blocks varies. Because the system will not allow the simultaneous existence of multiple hot data blocks, it is reasonable to consider the case of a single hot data block [36].

In the previous part, we calculate the ARAR of each data block, without considering the affordability of the data block. In this subsection, we first define a threshold that separates the hot and cold data according to the DRAR. We then define the throughput of a single data block using the threshold and the cost of achieving the maximum throughput.

Definition 1. The threshold λ^{th} is defined to determine the hot data. It also indicates the maximum read arrival rate that a data block can accommodate. A data block with a DRAR greater than λ^{th} is regarded as a hot data block. The maximum throughput is the maximum possible number of requests that can be served by the system per unit time [37]. Here, we define a similar concept for a single data block.

Definition 2. The LMT is defined as the maximum throughput of one block for the achievable direct read and degraded read services. In a local repairable group of the LRC, the DRARs of the data blocks are $\lambda_1^{\mathrm{d}} > \cdots > \lambda_{k/l}^{\mathrm{d}}$. The LMT of the first block (hot block) is

$$\mathrm{LMT} = \lambda^{\mathrm{th}} + \left(\lambda^{\mathrm{th}} - \lambda_2^{\mathrm{d}}\right). \tag{11}$$

For the LRC, the LMT of a hot data block is determined by the DRARs of the other data blocks in the same local repairable group, alongside threshold λ^{th}. The maximum direct read task for a single block is λ^{th}, which is a preset threshold. The degraded read of a block in a group requires all the other blocks in this group, and the most heavily burdened block among the other blocks decides the amount of degraded read task that can be completed, i.e., λ_2^{d}. After the second block completes its own direct read task λ_2^{d}, the rest can be used for the degraded read of the hot block, i.e., $\lambda^{\mathrm{th}} - \lambda_2^{\mathrm{d}}$.

Definition 3. Penalty (denoted by P) is defined as the additional read arrival rate. That is, the additional cost that all the other data blocks incur when the LMT of the first block is achieved. In a local repairable group of the LRC, the DRARs of the data blocks are $\lambda_1^{\mathrm{d}} > \cdots > \lambda_{k/l}^{\mathrm{d}}$. The LMT of the first block is achieved, and the penalty is

$$P = \left(\lambda^{\mathrm{th}} - \lambda_2^{\mathrm{d}}\right)(k/l - 1). \tag{12}$$

In order to share the load of $\lambda^{\mathrm{th}} - \lambda_2^{\mathrm{d}}$ for the first block, for each of the other k/l blocks in the same group (the other $k/l - 1$ data blocks in the group and a local parity block), the read arrival rate must increase by $\lambda^{\mathrm{th}} - \lambda_2^{\mathrm{d}}$. But, the additional load on a local parity block incurs extra influence. So penalty P is $\left(\lambda^{\mathrm{th}} - \lambda_2^{\mathrm{d}}\right)(k/l - 1)$.

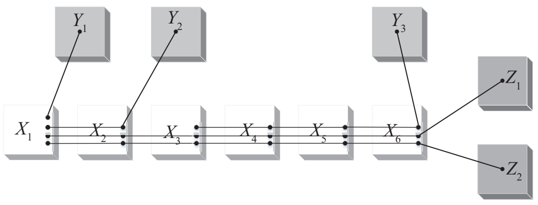

Figure 1 Structure of SLRC$(6,3,1,2)$.

4 Scalable local reconstruction code

4.1 Degraded reads of scalable local reconstruction code

We focus primarily on the case of a single hot data block, and assist degraded reading in increasing its LMT. When the node where the hot data block is located temporarily fails, the SLRC can easily use other blocks to obtain the content. An SLRC(k,l,l_0,r) consists of k data blocks. The DRARs of the k data blocks are $\lambda_1^{\mathrm{d}}, \lambda_2^{\mathrm{d}}, \ldots, \lambda_k^{\mathrm{d}}$, where $\lambda_1^{\mathrm{d}} > \lambda^{\mathrm{th}} > \lambda_2^{\mathrm{d}}$. In the 1st kind of local parity blocks, there are l_0 local parity blocks generated from the 1st data block that has the biggest DRAR, and l_0 is the scalable index. The 2nd kind of local parity blocks has $l-l_0-1$ parity blocks, each of which is generated from two data blocks, that is, the 1st data block and the 2nd to $(k-(l-l_0-1)+1)$-th data block, respectively. This process is illustrated as follows:

$$\begin{cases} \text{the } l_0\text{-th local parity block} \\ \quad \Leftarrow \ \text{the } k\text{-th data block} + \text{the 1st data block;} \\ \text{the } (l_0+1)\text{-th local parity block} \\ \quad \Leftarrow \text{the } (k-1)\text{-th data block} + \text{the 1st data block;} \\ \vdots \\ \text{the } (l-2)\text{-th local parity block} \\ \quad \Leftarrow \text{the } (k-(l-l_0-1)+1)\text{-th data block} + \text{the 1st data block.} \end{cases} \tag{13}$$

After the basic parameters (k,l,r) are determined, the scalable index l_0 can be changed according to the requirements of the actual situation. The structure of SLRC$(6,3,1,2)$ is as shown in Figure 1. In the figure, X_i represents the data block, Y_i represents the local parity block, Z_i represents the global parity block. Specifically, Y_1 is formed from X_1, Y_2 is formed from X_1 and X_2, Y_3 is formed from X_3 to X_6, and both Z_1 and Z_2 are formed from X_1 to X_6.

4.2 Code structure of scalable local reconstruction code

The code structure of the SLRC is explained as follows.

Part 1. For SLRC(k,l,l_0,r), we regard the first data block (e.g., the hot data block that has the biggest DRAR and a greater LMT), and l_0 local parity blocks as the first part.

Part 2. The number of local parity blocks in Part 2 is $l-l_0-1$. Each data block in Part 2 with the first data block generates a local parity block. The numbers of data blocks and local parity blocks in the second part are both $l-l_0-1$.

Part 3. Finally, one local parity block is generated from the last data blocks. The number of last data blocks is $k-1-(l-l_0-1)=k-l+l_0$. To ensure reliability without reducing the minimum distance, we need to have each data block covered at least once by a local repairable group.

In SLRC, the first part can be viewed as a direct copy of the first data block to meet the oversized DRAR requirements. The second part of the design is to better balance the ratio of local check blocks to data blocks, and can better use other light load data blocks to share the pressure. Based on the conclusion of Section 3, having fewer blocks in a locally repairable group helps reduce load, so in Part 2, only one block is added per locally repairable group. As we can see from Section 3, it is critical to avoid using the same data block in different locally repairable groups. Therefore, in Part 2, the blocks of data in each locally repairable group (except for the first block) are different. In addition, we should choose the data blocks with lower DRAR in Part 2 to share more of the read task. In addition, to ensure the failure repair capability of SLRC, each data block needs to be overwritten by a locally repairable group at least once, so the remaining data blocks form a locally repairable group. Under certain (k, l, r) parameters, we can choose the appropriate l_0 according to the performance requirements, and expandably adjust the corresponding cost of LMT and hot data block.

Remark 1. The SLRC(k, l, l_0, r), in which the DRARs of k data blocks are $\lambda_1^{\mathrm{d}}, \lambda_2^{\mathrm{d}}, \ldots, \lambda_k^{\mathrm{d}}$ ($\lambda_1^{\mathrm{d}} > \lambda^{\mathrm{th}} > \lambda_2^{\mathrm{d}}$), has the following properties:

(1) The LMT of the hot data block is $(l_0+1)\lambda^{\mathrm{th}} + \sum_{i=k-l+l_0+2}^{k} \left(\lambda^{\mathrm{th}} - \lambda_i^{\mathrm{d}}\right)$;

(2) The penalty P is $\sum_{i=k-l+l_0+2}^{k} \left(\lambda^{\mathrm{th}} - \lambda_i^{\mathrm{d}}\right)$.

Proof. The local throughput of the hot data block described in Part 1 is $(l_0+1)\lambda^{\mathrm{th}}$. Through $l - l_0 - 1$ local repair groups, the throughput of hot data block described in Part 2 is $\sum_{i=k-(l-l_0-1)+1}^{k} \left(\lambda^{\mathrm{th}} - \lambda_i^{\mathrm{d}}\right)$, so the LMT of the hot data block is $(l_0+1)\lambda^{\mathrm{th}} + \sum_{i=k-l+l_0+2}^{k} \left(\lambda^{\mathrm{th}} - \lambda_i^{\mathrm{d}}\right)$.

Part 1 employs the method of direct replication, which incurs no extra penalty. However, as Part 2 forms a local repairable group through data blocks, it will incur a corresponding penalty, which is influenced by the direct read arrival rates of the data blocks in Part 2. According to the previous part, we select the data blocks with the lower DRAR to read directly, i.e., $l - l_0 - 1$ data blocks.

4.3 Generator matrix of scalable local reconstruction code

For SLRC(k, l, l_0, r), we set each data block to at least one local repairable group, and they are protected by the same global parity block. This is to ensure that it has the same maximum decodable failover as the LRC, so that SLRC(k, l, l_0, r) can also decode any block failure.

For SLRC$(6, 3, 1, 2)$, its generator matrix satisfies

$$G = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 \\ a_1 & b_1 & c_1 & c_2 & c_3 & c_4 \\ a_1^2 & b_1^2 & c_1^2 & c_2^2 & c_3^2 & c_4^2 \end{pmatrix}, \tag{14}$$

and encoding equation is shown as

$$G \times \begin{pmatrix} x_1 & x_2 & x_3 & x_4 & x_5 & x_6 \end{pmatrix}^{\mathrm{T}} = \begin{pmatrix} y_1 & y_2 & y_3 & z_1 & z_2 \end{pmatrix}^{\mathrm{T}}. \tag{15}$$

The data blocks and local parity blocks of SLRC$(6, 3, 1, 2)$ are divided into three local repairable groups. According to (14), x_1 and y_1 constitute the first local repairable group, in which y_1 is the copy of x_1. The x_1, x_2 and y_2 constitute the second local repairable group, in which y_2 is generated by x_1 and x_2. The x_3, x_4, x_5, x_6 and y_3 constitute the third local repairable group, in which y_3 is generated by x_3 to x_6. For each failure pattern, we can derive a simplified encoding matrix based on a reduced decoding Tanner graph. Tanner graph can be used to determine whether a failure patter is decodable on information-theoretic. For the selection of SLRC encoding parameters, it is necessary to successfully decode the information-theoretically decodable four failures. According to the analysis of various situations, the following four failures should be decodable: x_1, x_2, x_3, x_4 fail, x_1, x_3, x_4, x_5 or x_2, x_3, x_4, x_5 fail, x_1, x_2, x_3, y_2 fail, and x_1, x_2, x_3, y_3 fail.

By analyzing different cases separately, we can get the following conditions:

$$a_1, b_1, c_i \neq 0, \tag{16}$$

$$a_1 \neq b_1, \ c_i \neq c_j, \tag{17}$$

$$a_1 + b_1 \neq c_i + c_j, \tag{18}$$

$$a_1 + b_1 \neq c_i. \tag{19}$$

It is easy to meet these conditions by choosing a_1, b_1, and c_i from a finite field GF(2^5). For instance, let $a_1 = 00011$, $b_1 = 00010$, $c_1 = 11000$, $c_2 = 01000$, $c_3 = 01100$ and $c_4 = 10000$. The above conditions are then satisfied.

Table 2 A summary of performance metrics

Metrics	LRC(k, l, r)	SLRC(k, l, l_0, r)
Reconstruction cost	$\frac{(r+1)k+k^2/l}{k+l+r}$	$\frac{(4l-3l_0-3)+(k-l+l_0+1)(k-l+l_0)+kr}{k+l+r}$
Storage overhead	$(k+l+r)/k$	$(k+l+r)/k$
Any $(r+1)$ failures tolerance	Yes	Yes
LMT of the hot block	$\lambda^{\rm th} + (\lambda^{\rm th} - \lambda_2^{\rm d})$	$(l_0+1)\lambda^{\rm th} + \sum\limits_{i=k-l+l_0+2}^{k} (\lambda^{\rm th} - \lambda_i^{\rm d})$
Penalty	$(\lambda^{\rm th} - \lambda_2^{\rm d})\,(k/l-1)$	$\sum\limits_{i=k-l+l_0+2}^{k} (\lambda^{\rm th} - \lambda_i^{\rm d})$

5 Performance analysis and code selection

In this section, we compare the performance of SLRC and LRC in several aspects. Based on the performance of SLRC in each performance. The analysis of code word selection is given. Table 2 shows the SLRC and LRC in terms of the LMT, storage overhead, and reconstruction cost.

5.1 Performance analysis

5.1.1 *Local maximum throughput and penalty*

As aforementioned, a single hot data block can be obtained from other data blocks and a local parity block in the same local repairable group through degraded reads using the LRC. The DRARs of the data blocks in a group are $\lambda_1^{\rm d} > \lambda^{\rm th} > \lambda_2^{\rm d} > \cdots > \lambda_{k/l}^{\rm d}$. It can be inferred that, the LMT of the hot (first) block is $\lambda + \left(\lambda^{\rm th} - \lambda_2^{\rm d}\right)$, and the penalty is $\left(\lambda^{\rm th} - \lambda_2^{\rm d}\right)(k/l-1)$. For the LRC, each data block is considered as homogeneous and cannot provide a larger LMT for the hot data blocks to meet the needs of uneven DRARs. The new SLRCs proposed in this paper perform better in alleviating the local overhead under the same storage parameters (k, l, r). In Figure 2(a), the SLRC$(6, 3, 1, 2)$ has the local repairable group consisting of X_1, X_2 and X_2. We choose ratio $\rho^{\rm th} = 5$ as an example. The LMT of X_1 is $2\lambda^{\rm th} - \lambda_2^{\rm d}$, and the penalty is $\lambda^{\rm th} - \lambda_2^{\rm d}$. For the group of X_1 and Y_1, the LMT of X_1 is $2\lambda^{\rm th}$, and penalty is 0. Thus in the SLRC$(6, 3, 1, 2)$, the LMT of X_1 is $3\lambda^{\rm th} - \lambda_2^{\rm d}$, and the penalty is $\lambda^{\rm th} - \lambda_2^{\rm d}$. We show the relationship of the LMT and the penalty under the same k and l. We can see that the trend is the similar under the same parameter settings. So we take two distinct parameter settings as an example, $k = 9, l = 3$ and $k = 12, l = 4$. We can see that both the LMT and penalty increase with l_0.

5.1.2 *Storage overhead*

When choosing a redundancy mechanism, overhead is one of the top performance indicators. Usually, storage overhead is defined as the ratio of the amount of the actual data stored to the amount of the original data. In storage systems, erasure coding is used to reduce the storage overhead by more than 50%. Compared with the traditional 3-repetition method, MDS codes can significantly reduce the storage overhead to 1.33×, while maintaining high longevity. The LRC(k, l, r) requires additional storage to store l local parity blocks and r global parity blocks for k data blocks. Hence, the normalized storage overhead is $(k+l+r)/k$. The LRC(6, 2, 2) in our example has storage overhead of $1+4/6 = 1.67\times$. Here the same basic parameter as (6, 2, 2) is used, and we aim to only modify the coding method of the coding block without imposing an extra load to achieve the same storage overhead as LRC. The storage overhead of SLRC(k, l, l_0, r) is same as $n/k = (k + l + r)/k$. Figure 2(b) depicts the relationship between the LMT and storage overhead. Given the same storage overhead, a larger value of k can lead to a greater LMT. At the same time, the SLRC can achieve a greater LMT than its LRC counterpart. The LRC(k, l, r) requires additional storage to store l local parity blocks and r global parity blocks for k data blocks. Hence, the normalized storage overhead is $(k + l + r)/k$. The LRC (6, 2, 2) in our example has storage overhead of $1 + 4/6 = 1.67\times$. Here the same basic parameters as (6, 2, 2) are used, and we aim to only modify the coding method of the coding block without imposing an extra load to achieve the same storage overhead as LRC. The storage overhead of SLRC(k, l, l_0, r) is same as $n/k = (k + l + r)/k$. Figure 2(c) depicts the relationship between the LMT and storage overhead. Given the same storage overhead, a larger value of k can lead to a greater LMT. At the same time, the SLRC can achieve a greater LMT than its LRC counterpart.

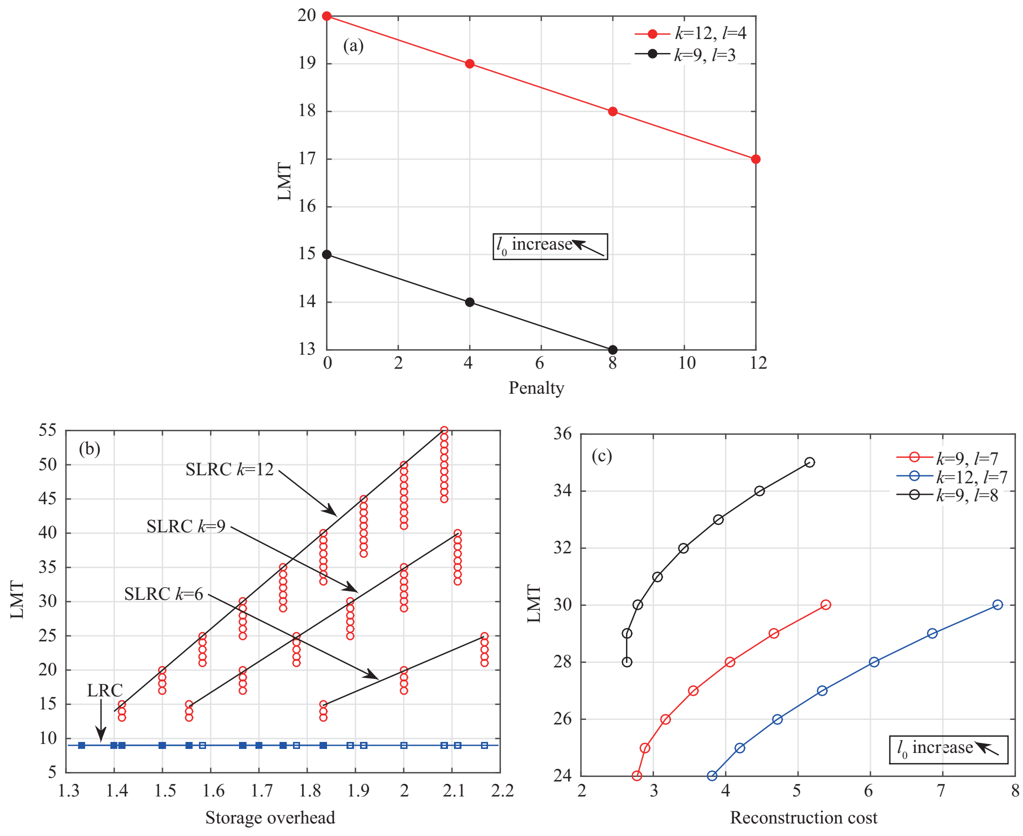

Figure 2 (Color online) LMT versus penalty, storage overhead, and reconstruction cost for the SLRC with a variety of values of parameters. (a) LMT versus penalty when l_0 varies; (b) LMT versus storage overhead with different values of k; (c) LMT versus reconstruction cost when l_0 varies.

5.1.3 *Reconstruction cost*

Another important indicator for LRC codes is the reconstruction cost, which is defined as the number of blocks required to reconstruct an unavailable data block. For example, for an RS$(6,3)$ the reconstruction cost equals 6.

We take SLRC$(6,3,1,2)$ in Figure 1 as an example to introduce the reconstruction cost. SLRC$(6,3,1,2)$ has five parity blocks, including three local parity blocks (belong to three local parity groups respectively) and two global parity blocks. To reconstruct X_1, since Y_1 is a copy of X_1, we can just use Y_1, and similarly, to reconstruct Y_1, we just need X_1. To reconstruct X_2, we should use X_1 and Y_2, which is the same for reconstructing Y_2. To reconstruct andy block from X_3 to X_6, we need Y_3 and all of the data blocks from X_3 to X_6 except for itself. Similar to the RS codes, the reconstruction of any global parity block requires six blocks. So we can derivate that the reconstruction cost of the first local parity group is l_0+1, the reconstruction cost of the second local parity group is $4(l-l_0-1)$, the reconstruction cost of the third local parity group is $(k-l+l_0+1)(k-l+l_0)$. Therefore, the normalized reconstruction cost is $((4l-3l_0-3)+(k-l+l_0+1)(k-l+l_0)+kr)/n$.

Figure 2(c) shows that with the same parameters k and l, an LRC system has increased maximum throughput and reconstruction cost with larger l_0. In addition, it can be seen that the LMT depends only on l_0 but is independent of i and l. However, given a fixed l_0, larger k and l can lead to a better trade-off among the other performance measures.

5.1.4 *Tradeoff and lower bound*

Through varying parameters k and l, we can obtain a lower bound. Figure 3(a) compares the LRC, RS code, and SLRC. Both the SLRC and LRC achieve better trade-offs than the RS codes across the entire range of coding parameters. At the expense of a better LMT, the SLRC will be slightly inferior to the LRC in terms of the trade-off lower bound between the storage overhead and reconstruction cost. In Figure 3(b), we have marked some typical codewords. Through the horizontal and vertical comparisons,

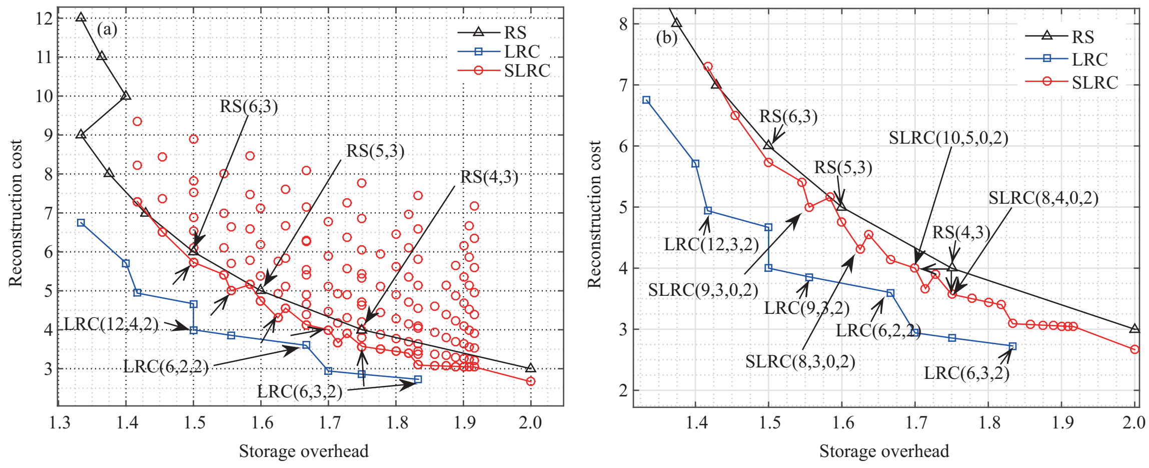

Figure 3 (Color online) Lower bound of the trade-off between the storage overhead and reconstruction cost of the SLRC. (a) Lower bounds of SLRC, LRC, and RS codes; (b) typical codes on lower bounds.

some points represented by the corresponding parameter values are selected and shown in Figure 3(b).

5.2 Code selection based on DRAR of hot and cold data blocks

In the subsection, we proposed three code selection algorithms, namely, minimum reconstruction cost, minimum storage overhead, and minimum penalty to adapt to different demands in the real world. We referred to some existing studies in the algorithm design. For example, code selection for availability and reconstruction cost is discussed in [38], respectively. Wei et al. [39] proposed an adaptive coding selection method, which can select the l parameter in LRC(k,l,r) according to the I/O penalty, and select the code with lower storage overhead when the I/O penalty is the same. Of these three algorithms, the minimum reconstruction cost algorithm takes k,l as the input parameters and adjusts l_0 to minimize the reconstruction cost under the condition that meets the DRAR, that is, the reconstruction cost is minimized with the same storage overhead. The minimum storage overhead algorithm takes k as the input parameter, and adjusts l_0 to meet the DRAR, in this case, $l=l_0+2$ should be set to minimize the storage overhead. The minimum penalty algorithm takes k,l as the input parameters and gives the formula of l_0 which can minimize the penalty under different conditions.

5.2.1 *Typical point of SLRC*

Each set of LRC parameters (k,l,r) yields one set of values of the reconstruction cost and storage overhead. For the LRC$(6,2,2)$, the reconstruction cost is three, and the storage overhead is 1.67×. We obtain many sets of values by varying the parameters. Since each block has to reside on a different node, the number of nodes in a cluster limits the total number of blocks in the code. The maximum value of k adopted in our simulation is 12. To calculate the LMT, we normalize the DRARs of all cold data blocks to one, denoted by $\lambda_0^{\rm th}\rho^{\rm th}$. The larger the $\rho^{\rm th}$, the smaller the load on the system caused by the other cold data blocks. We simulate all possible SLRCs from $k=6$ to $k=12$. We compare the performance of these SLRCs with the possible parameter sets commonly used in the LRC and RS codes, and analyze the results as follows. In this part, we choose ratio $\rho^{\rm th}=5$ as an example. $\rho^{\rm th}$ has no effect on the storage overhead and reconstruction cost, and does not alter the corresponding relationship among the code parameters. Figure 4(a) plots the storage overhead and the LMT of the SLRC and some typical LRC. As can be seen from the figure, the SLRC covers a large range of memory loads. Corresponding to the storage loads achieved by the typical LRC, there are some point sets that can be implemented, as the arrows in Figure 4. Meanwhile, Figure 4(b) shows the SLRC corresponding to the LMT that can be achieved by the storage overhead. In Figure 4(c), the SLRC$(12,6,5,2)$ can achieve the same storage overhead as the LRC$(6,2,2)$ (i.e., 1.67×) while the SLRC codes achieve more than three times of the LMT than the LRC. The reconstruction cost and the LMT performance are depicted in Figure 4(c) and (d). The LRC reconstruction cost for each SLRC is marked in Figure 4(d). As can be observed from the figure, the SLRC$(11,8,3,2)$ can achieve the same storage overhead as the LRC$(6,2,2)$, while the SLRC is able to achieve more than three times of the LMT than the LRC.

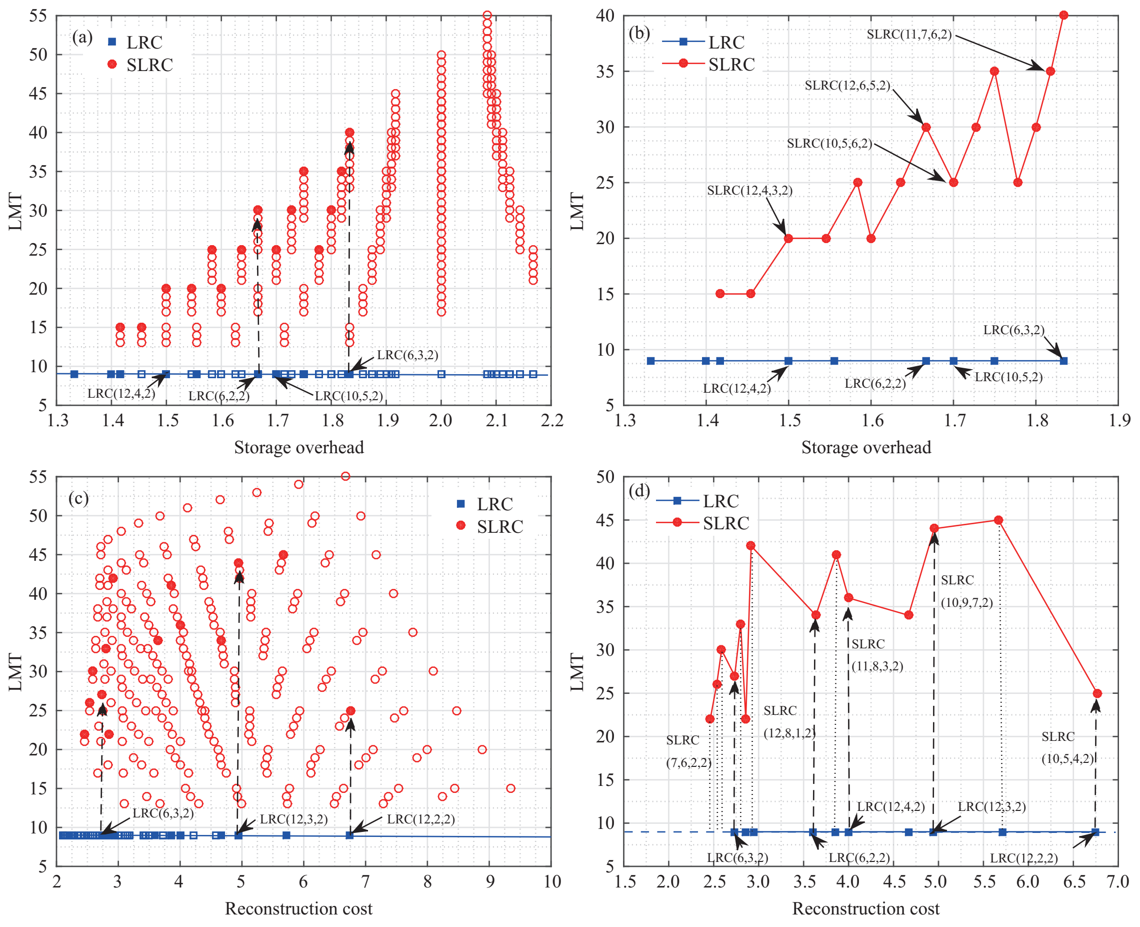

Figure 4 (Color online) SLRC performance (LMT corresponds to various storage overhead and reconstruction cost) with a variety of parameters. (a) LMT corresponding to various storage overhead; (b) typical codes corresponding to various storage overhead; (c) LMT corresponding to various reconstruction cost; (d) typical codes corresponding to various reconstruction cost.

5.2.2 *Minimum reconstruction cost code selection*

In this subsection, we present an algorithm that can determine the value of l_0 through SLRC parameters, so as to obtain the lowest reconstruction cost.

Algorithm 1 shows how to minimize the reconstruction cost.

The output of Algorithm 1 is the SLRC scalable index l_0, when the DRAR, the number of data blocks k, and the number of local parity blocks l, are determined. l_0 will correspond to different structures of SLRC and affect the number of data blocks in each local repairable group.

Firstly, in order to realize the LMT and to ensure the minimum reconstruction cost, the values of k and l need to be given to define the range. In order to satisfy the 3-part division of the SLRC, k should not be less than four, and l should not be less than three and not greater than $k-1$. The DRARs of all blocks are given to determine the specific value of l_0 as follows:

(1) The number of data blocks is integer k ($k > 4$);

(2) The number of local parity blocks is integer l ($l \in [3, k-1]$);

(3) The direct read rates of all data blocks are $\lambda_1^{\mathrm{d}} > \lambda^{\mathrm{th}} > \lambda_2^{\mathrm{d}} > \cdots > \lambda_k^{\mathrm{d}}$.

Next, to minimize the reconstruction cost, each local repairable group should have fewer blocks, so l_0 should be small. The value of l_0 can be chosen according to the predetermined $\lambda_1^{\mathrm{d}}, \lambda_2^{\mathrm{d}}, \ldots, \lambda_k^{\mathrm{d}}$ to meet this requirement. Last, according to the simulation and theoretical results, the LMT of the hot data block under a coding method depends only on l_0. After l_0 is determined, the smaller l is chosen, the more storage overhead can be reduced.

Algorithm 1 Minimum reconstruction cost algorithm

```
Input: The number of data blocks $k$, the number of local parity blocks $l$, the maximum affordable read arrival rate of hot data
    blocks $\lambda^{\rm th}$, and the DRARs of $k$ data blocks.
Output: $l_0$
 1: Sort the data blocks by their max throughput. Then we have $\lambda_1^{\rm d} > \lambda_2^{\rm d} > \cdots > \lambda_k^{\rm d}$.
 2: for $i = k$ do
 3:   if $l\lambda^{\rm th} \leqslant \lambda_1^{\rm d}$ then
 4:     $l_0 = l - 1$;
 5:     break;
 6:   else
 7:     if $l\lambda^{\rm th} - \sum_{m=i}^{k} \lambda_m^{\rm d} > \lambda_1^{\rm d}$ then
 8:       $i \leftarrow i - 1$;
 9:       if $l\lambda^{\rm th} - \sum_{m=i}^{k} \lambda_m^{\rm d} \leqslant \lambda_1^{\rm d}$ then
10:         $l_0 = l + i - k - 1$;
11:       end if
12:     end if
13:   end if
14: end for
15: return $l_0$.
```

5.2.3 *Minimum storage overhead code selection*

We now present the second algorithm to compute l_0 with the objective of achieving the minimum storage overhead in Algorithm 2.

Algorithm 2 Minimum storage overhead algorithm

```
Input: the number of data blocks $k$, the maximum affordable read arrival rate of hot data blocks $\lambda^{\rm th}$, DRAR of $k$ data blocks
    $\lambda_1^{\rm d}, \lambda_2^{\rm d}, \ldots, \lambda_k^{\rm d}$.
Output: $l_0$, $l$.
 1: Sort as the value of max throughput and change the serial number of corresponding blocks (including its max throughput)
    which are labeled as $1, \ldots, k$. Then we get $\lambda_1^{\rm d} > \lambda_2^{\rm d} > \cdots > \lambda_k^{\rm d}$;
 2: for $i = k$ do
 3:   if $l\lambda^{\rm th} - \sum_{m=i}^{k} \lambda_m^{\rm d} > \lambda_1^{\rm d}$ then
 4:     $i \leftarrow i - 1$;
 5:     if $l\lambda^{\rm th} - \sum_{m=i}^{k} \lambda_m^{\rm d} \leqslant \lambda_1^{\rm d}$ then
 6:       $l_0 = l + i - k - 1$;
 7:       $l = l_0 + 2$;
 8:     end if
 9:   end if
10: end for
11: return $l_0$, $l$.
```

In order to minimize storage overhead, we need to decide on the size of l_0 based on the given k and DRAR, and then determine the value of l. Firstly, the parameters should satisfy the following requirements. (1) The number of data blocks is integer k ($k > 4$); (2) the actual read rates of all data blocks satisfy $\lambda_1^{\rm d} > \lambda^{\rm th} > \lambda_2^{\rm d} > \cdots > \lambda_k^{\rm d}$. After determining the value of k, we use the given $\lambda_1^{\rm d}, \lambda_2^{\rm d}, \ldots, \lambda_k^{\rm d}$ to obtain the minimum l_0 that can satisfy this requirement. Next, l should be chosen as small as possible in order to minimize the storage overhead, and we can let $l = l_0 + 2$.

5.2.4 *Minimum penalty code selection*

The parameters should satisfy the following requirements. (1) The number of data blocks is integer k ($k > 4$); (2) the number of local parity blocks is integer l ($l \in [3, k-1]$); (3) the actual read rates of all data blocks satisfy $\lambda_1^{\rm d} > \lambda^{\rm th} > \lambda_2^{\rm d} > \cdots > \lambda_k^{\rm d}$. According to Remark 3, under the premise that LMT of hot data block satisfy $\lambda_1^{\rm d} \geqslant (l_0+1)\lambda^{\rm th} + \sum_{i=k-l+l_0+2}^{k} \left(\lambda^{\rm th} - \lambda_i^{\rm d}\right)$, $\sum_{i=k-l+l_0+2}^{k} \left(\lambda^{\rm th} - \lambda_i^{\rm d}\right)$ should be made as small as possible to achieve the minimum cost. First, when $k \leqslant \rho^{\rm th}$, for satisfying the minimum LMT requirements for hot data blocks, let $\lambda_1^{\rm d} = (l_0+1)\lambda^{\rm th} + \sum_{i=k-l+l_0+2}^{k} \left(\lambda^{\rm th} - \lambda_i^{\rm d}\right)$, so that $(l_0+1)\lambda^{\rm th} = \lambda_1^{\rm d}$, and $l_0 = \lambda_1^{\rm d}/\lambda^{\rm th} - 1$. And according to $\lambda^{\rm th} \geqslant \lambda_2^{\rm d}$, we can get $l_0 = \lceil \lambda_1^{\rm d}/\lambda_2^{\rm d} \rceil - 1$. Then for the given $\lambda_1^{\rm d} > \lambda^{\rm th} > \lambda_2^{\rm d} > \cdots > \lambda_k^{\rm d}$, there is $l_0 = \lceil \lambda_1^{\rm d}/\lambda_2^{\rm d} \rceil - 1 = \rho^{\rm th} - 1$. And when we set $\lambda_2^{\rm d} = \cdots = \lambda_k^{\rm d}$, there is $\lambda^{\rm th} = \frac{l-l_0-1+\rho^{\rm th}}{l}\lambda_2^{\rm d}$. So $\lambda^{\rm th} = \lambda_2^{\rm d}$, $l_0 = \rho^{\rm th} - 1$. When $k \geqslant \rho^{\rm th}$, it can be seen that due to the limit of parameter range, $l_0 = \rho^{\rm th} - 1$ cannot be satisfied. In order to make $\sum_{i=k-l+l_0+2}^{k} \left(\lambda^{\rm th} - \lambda_i^{\rm d}\right)$ as small as

Table 3 HDFS parameter

Parameter	Value
DFS_BLOCK_SIZE_KEY	134217728
DFS_NAMENODE_EC_POLICIES_MAX_CELLSIZE_KEY	1048576
DFS_NAMENODE_HANDLER_COUNT_KEY	10
DFS_STREAM_BUFFER_SIZE_KEY	4096
DFS_DISK_BALANCER_ENABLED	False
DFS_DN_EC_RECONSTRUCTION_THREADS_KEY	8

possible, $l - l_0 - 2 = 0$ should be satisfied, then it should have $l = l_0 + 2$, due to $l_0 = \frac{\rho^{\rm th}}{l}\lambda_2^{\rm d}$.

6 Experiment

6.1 Implementation methodology

In this subsection, we implemented the above coding schema in HDFS erasure coding (HDFS-EC). The HDFS is a functional module that is used to store files in a Hadoop distributed cluster. The Hadoop cluster consists of a NameNode and several DataNodes, which serve not only HDFS's distributed storage capabilities, but also MapReduce's distributed computing capabilities. For HDFS, the NameNode is responsible for storing metadata (e.g., file information, block location), whereas the DataNodes is responsible for storing data. In Hadoop 1.x and 2.x version, HDFS uses n-replication for fault-tolerance. The erasure codes, which include RS and XOR codes, are built into Hadoop 3.x (HDFS-EC). We have deployed LRC and SLRC coding schemas on Hadoop 3.1.3 based on RS codes.

We installed the VMware virtual machine (VM) cluster on a Dell T7920 workstation, which is equipped with two 20-core Intel Xeon Silver 4214 2.2 GHz processors, 128 GB RAM, and 4×1 T SSD. There are 15 VMs in this cluster with an 8-core processor, 4 GB RAM, and 100 GB disk, each of which runs CentOS 7. For convenience, the maximum uplink bandwidth of each VM is set to 100 Mbps, which is considered the node's maximum throughput of the node. HDFS-EC encodes the file into multiple blocks with a size of blocksize through EC mode and stores them in the DataNodes. Taking RS(k, r) as an example, HDFS-EC divides the file into multiple groups with the size of $k \times$ blocksize bytes, which is called block groups. Each block group is encoded into k data blocks and r parity blocks, each of which has a size of blocksize. If the entire file or remaining data after grouping is less than $k \times$ blocksize, it will be divided into equal k parts based on the actual size of the data. The data will then be encoded into k data blocks and r parity blocks. LRC(k, l, r) and SLRC(k, l, l_0, r) encoding procedures are similar to RS code, except that they add l local parity blocks on basis of RS code.

The parameter configuration of HDFS-EC exists in the DFSConfigKeys source file. Table 3 shows the parameter of this experiment. However, other parameters not listed in the table are all configured by default. We used several typical erasure codes such as RS$(6, 3)$, LRC$(6, 3, 2)$, SLRC$(6, 3, 1, 2)$, SLRC$(6, 4, 1, 2)$, and SLRC$(6, 4, 2, 2)$ for comparison experiments. These codes have similar k and can tolerate any three failures. To facilitate the statistics of experimental results, we used the test file with the size of $k \times$ blocksize (768 MB) to ensure that each DataNode has only one block.

The situation of multiple blocks in one DataNode is exactly the same in practice. Native HDFS does not provide a degraded read mechanism. To evaluate degraded reads performance, we designed the equivalent mechanism to implement degraded read using the following repair processes.

(1) Set the uplink bandwidth of the hot block to 0 to simulate its full load so that it cannot provide external services.

(2) Use the hot block to trigger the repair mechanism of HDFS.

(3) Record network I/O of each node using the iptraf-ng tool during the experiment process.

We also configured the background traffic of each VM to ensure that the remaining bandwidth of the cold data blocks varies between 10 and 100 Mbps, i.e., $1/\rho^{\rm th}$ between 0.9 and 0. We conducted ten experiments and averaged the results to get a more convincing result.

6.2 Experimental results analysis

We compared LMT, penalty, and efficiency (i.e., LMT/penalty) for several typical LRCs and SLRCs. As the baseline, we also tested the native RS codes of HDFS. RS codes are MDS codes that can read

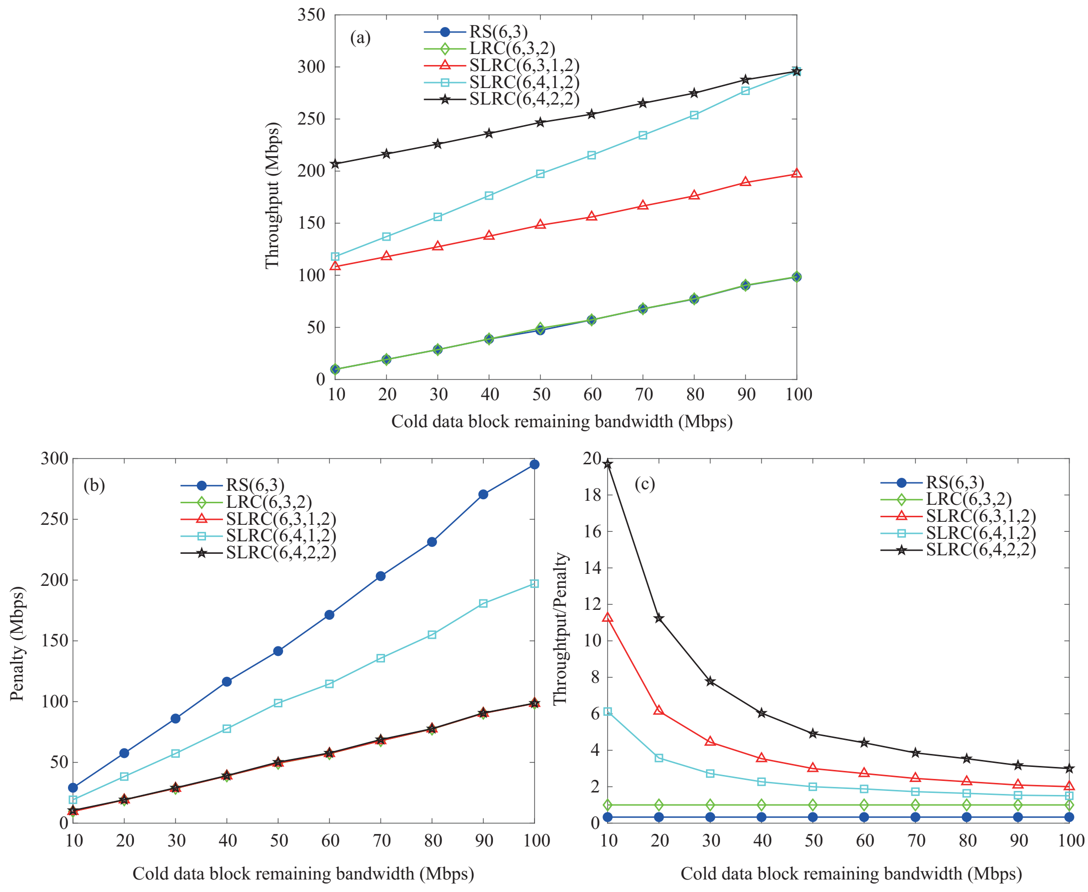

Figure 5 (Color online) Experiment results comparison between RS, LRC, and SLRC in HDFS-EC. (a) LMT of the hot block; (b) penalty; (c) efficiency.

arbitrary k blocks to recover the hot block in the degraded reads. To reduce the bandwidth consumption of data blocks, we used $k-r$ data blocks and r parity blocks (i.e., all parity blocks); thus, the penalty can be given by (20), and the proof is the same as Definition 12.

$$P = \left(\lambda^{\mathrm{th}} - \lambda_2^{\mathrm{d}}\right)(k-r). \tag{20}$$

Figure 5 shows the LMT of the hot block, penalty, and efficiency of the four different erasure coding modes under different remaining bandwidths of the cold data blocks. Figures 5(a) and (b) show that RS$(6,3)$ and LRC$(6,3,2)$ have the same LMT of the hot block, but LRC$(6,3,2)$ has one-third of the penalty of RS$(6,3)$. This is because LRC adds local parity blocks based on RS code so that the number of data blocks that must be connected is only $1/l$ of RS code. SLRC$(6,3,1,2)$ adjusts the structure of local repairable groups based on LRC$(6,3,2)$ while providing one local parity block as a copy of the hot data block. Figure 5(a) shows that SLRC$(6,3,1,2)$ has approximately 100 Mbps more throughput than LRC$(6,3,2)$, which is equivalent to λ^{th}. As SLRC$(6,3,1,2)$ does not increase the number of parity blocks, it has the same storage parameters as LRC$(6,3,2)$. SLRC$(6,4,l_0,2)$ adds one local parity block on the basis of SLRC$(6,3,1,2)$. If the parity block being added belongs to the part one local repairable block group, i.e., SLRC$(6,4,2,2)$, it can increase the LMT of the hot block by 1 times λ^{th} compared with SLRC$(6,3,1,2)$, while the penalty remains constant. Further, if the parity block being added belongs to the part two local repairable block group, i.e., SLRC$(6,4,1,2)$, the increased LMT of the hot block is equivalent to the remaining bandwidth of the cold data block compared with SLRC$(6,3,1,2)$. However, the degraded reads, in this case, require the cold data blocks, which increases the penalty. Thus, Figure 5(c) shows that, SLRC$(6,4,1,2)$ has a higher penalty than SLRC$(6,4,2,2)$. However, the penalty of LRC$(6,3,2)$, SLRC$(6,3,1,2)$, and SLRC$(6,4,2,2)$ are the same because they use one cold data block's remaining bandwidth during the degraded reads. The increased LMT of the hot block in RS code and LRC must be swapped by sacrificing the throughput of cold data blocks, so changing of ρ^{th} affects the LMT of the hot block but not the efficiency. Because of the existence of part one local repairable block group in SLRC, the smaller ρ^{th}, i.e., the larger the cold data block load, the more obvious the efficiency advantage of SLRC. Two conclusions can be drawn from the above analysis. For LRCs and SLRCs, which have similar k and l, SLRCs have $l_0 \times \lambda^{\text{th}}$ more LMT of the hot data block than LRCs without additional penalty. For SLRCs that differ only in l_0, increasing l_0 while decreasing the penalty improves the LMT of the hot block.

7 Conclusion

In this paper, we proposed the SLRC to improve the LMT of hot data read in a CSSs. By modifying the code structure, the block with the lower DRAR can help offload the reading requests for the hottest block by degraded reads. On this basis, we presented several algorithms to select the code structure, which includes the minimum reconstruction cost, minimum storage overhead, and the minimum penalty algorithms. We implemented SLRCs in HDFS to verify its performance in practical CSSs. Extensive experiments in HDFS show that our work has accomplished superior performance for improving the LMT of the hot data block and reducing the network bandwidth consumption. In the future, we will focus on the repair bandwidth of SLRCs in heterogeneous clusters to expand its application scenarios.

Acknowledgements This work was supported by National Natural Sciences Foundation of China (Grant Nos. 61831008, 62027802, 62271165), Guangdong Science and Technology Planning Project (Grant No. 2021A1515011572), Shenzhen Natural Science Fund (Grant No. JCYJ20200109112822953), Shenzhen Natural Science Fund (Stable Support Plan Program) (Grant No. GXWD20201230155427003-20200824081029001), and Major Key Project of PCL (Grant No. PCL2021A03-1).

References

1 Ananthanarayanan G, Agarwal S, Kandula S, et al. Scarlett: coping with skewed content popularity in MapReduce cluster. In: Proceedings of ACM SIGOPS/EuroSys European Conference Computer Systems, 2011. 287–300

2 Tan X Y, Guo Y C, Chen Y S, et al. Accurate inference of user popularity preference in a large-scale online video streaming system. Sci China Inf Sci, 2018, 61: 018101

3 André F, Kermarrec A, Merrer E L, et al. Archiving cold data in warehouses with clustered network coding. In: Proceedings of ACM SIGOPS/EuroSys European Conference Computer Systems, 2014. 1–14

4 Ghosh M, Raina A, Xu L, et al. Popular is cheaper: curtailing memory costs in interactive analytics engines. In: Proceedings of ACM SIGOPS/EuroSys European Conference Computer Systems, 2018. 1–14

5 Hu D, Feng D, Xie Y, et al. Efficient provenance management via clustering and hybrid storage in big data environments. IEEE Trans Big Data, 2020, 6: 792–803

6 Balakrishnan S, Black R, Donnelly A, et al. Pelican: a building block for exascale cold data storage. In: Proceedings of USENIX Conference Operating Systems Design and Implementation, 2014. 351–365

7 Schroeder B, Gibson G A. Disk failures in the real world: what does an MTTF of 1,000,000 hours mean to you? In: Proceedings of the 5th USENIX Conference File Storage Technology, 2007. 1–16

8 Ford D, Labelle F, Popovici F I, et al. Availability in globally distributed storage systems. In: Proceedings of USENIX Conference Operating Systems Design and Implementation, 2010. 61–74

9 Fang J, Wan S, Huang P, et al. Early identification of critical blocks: making replicated distributed storage systems reliable against node failures. IEEE Trans Parallel Distrib Syst, 2018, 29: 2446–2459

10 Calder B, Wang J, Ogus A, et al. Windows azure storage: a highly available cloud storage service with strong consistency. In: Proceedings of ACM Symposium on Operating Systems Principles, 2011. 143–157

11 Ghemawat S, Gobioff H, Leung S T. The Google file system. In: Proceedings of ACM Symposium on Operating Systems Principles, 2003. 29–43

12 Abulibdeh H, Princehouse L, Weatherspoon H. RACS: a case for cloud storage diversity. In: Proceedings of the 1st ACM Symposium on Cloud Computing, 2010. 229–240

13 Muralidhar S, Lloyd W, Roy S, et al. F4: Facebook's Warm BLOB storage system. In: Proceedings of USENIX Conference Operating Systems Design and Implementation, 2014. 383–398

14 Wang J Z, Luo Y, Shum K W. Storage and repair bandwidth tradeoff for heterogeneous cluster distributed storage systems. Sci China Inf Sci, 2020, 63: 122301

15 Zhou L Y, Zhang Z F. Explicit construction of minimum bandwidth rack-aware regenerating codes. Sci China Inf Sci, 2021. doi: 10.1007/s11432-021-3304-6

16 Balaji S B, Krishnan M N, Vajha M, et al. Erasure coding for distributed storage: an overview. Sci China Inf Sci, 2018, 61: 100301

17 Hou H X, Han Y S. A class of binary MDS array codes with asymptotically weak-optimal repair. Sci China Inf Sci, 2018, 61: 100302

18 Huang C, Simitci H, Xu Y K, et al. Erasure coding in windows azure storage. In: Proceedings of USENIX Annual Technical Conference, 2012. 2

19 Dimakis A G, Godfrey P B, Wu Y, et al. Network coding for distributed storage systems. IEEE Trans Inform Theor, 2010, 56: 4539–4551

20 Tang X, Yang B, Li J, et al. A new repair strategy for the Hadamard minimum storage regenerating codes for distributed storage systems. IEEE Trans Inform Theor, 2015, 61: 5271–5279

21 Yang B, Tang X, Li J. A systematic piggybacking design for minimum storage regenerating codes. IEEE Trans Inform Theor, 2015, 61: 5779–5786

22 Li J, Tang X, Parampalli U. A framework of constructions of minimal storage regenerating codes with the optimal access/update property. IEEE Trans Inform Theor, 2015, 61: 1920–1932

23 Rashmi K, Shah N B, Gu D K, et al. A hitchhiker's guide to fast and efficient data reconstruction in erasure-coded data centers. In: Proceedings of ACM Conference SIGCOMM, 2014. 331–342
24 Shen Z, Lee P P C, Shu J, et al. Encoding-aware data placement for efficient degraded reads in xor-coded storage systems: algorithms and evaluation. IEEE Trans Parallel Distrib Syst, 2018, 29: 2757–2770
25 Zhu Y, Lin J, Lee P P C, et al. Boosting degraded reads in heterogeneous erasure-coded storage systems. IEEE Trans Comput, 2015, 64: 2145–2157
26 Shen Z, Shu J, Fu Y. HV Code: an all-around MDS code for RAID-6 storage systems. IEEE Trans Parallel Distrib Syst, 2015, 27: 1674–1686
27 Li R, Lee P P C, Hu Y. Degraded-first scheduling for map-reduce in erasure-coded storage clusters. In: Proceedings of the 44th Annual IEEE/IFIP International Conference Dependable Systems and Networks, 2014. 419–430
28 Khan O, Burns R, Plank J S, et al. Rethinking erasure codes for cloud file systems: minimizing I/O for recovery and degraded reads. In: Proceedings of the 10th USENIX Conference on File and Storage Technologies, 2012. 251–264
29 Fu Y, Shu J, Shen Z. EC-FRM: an erasure coding framework to speed up reads for erasure coded cloud storage systems. In: Proceedings of 2015 44th International Conference on Parallel Processing Workshops, 2015. 480–489
30 Aggarwal V, Chen Y F R, Lan T, et al. Sprout: a functional caching approach to minimize service latency in erasure-coded storage. IEEE/ACM Trans Networking, 2017, 25: 3683–3694
31 Zhang X J, Cai Y, Liu Y F, et al. NADE: nodes performance awareness and accurate distance evaluation for degraded read in heterogeneous distributed erasure code-based storage. J Supercomput, 2020, 76: 4946–4975
32 Chowdhury M, Kandula S, Stoica I. Leveraging endpoint flexibility in data-intensive clusters. In: Proceedings of ACM SIGCOMM Conference, 2013. 231–242
33 Fu M, Feng D, Hua Y, et al. Reducing fragmentation for in-line deduplication backup storage via exploiting backup history and cache knowledge. IEEE Trans Parallel Distrib Syst, 2016, 27: 855–868
34 Li P, Jin X T, Stones R J, et al. Parallelizing degraded read for erasure coded cloud storage systems using collective communications. In: Proceedings of IEEE Trustcom/BigDataSE/ISPA, 2016. 1272–1279
35 Nachiappan R, Javadi B, Calheiros R N, et al. ProactiveCache: on reducing degraded read latency of erasure coded. In: Proceedings of IEEE International Conference on Cloud Computing Technology and Science (CloudCom), 2019. 223–230
36 Shuai Q, Li V O K. Delay performance of direct reads in distributed storage systems with coding. In: Proceedings of IEEE 17th International Conference on High Performance Computing and Communication, 2015. 184–189
37 Lee K, Shah N B, Huang L, et al. The MDS queue: analysing the latency performance of erasure codes. IEEE Trans Inform Theor, 2017, 63: 2822–2842
38 Hu Y, Liu Y, Li W, et al. Unequal failure protection coding technique for distributed cloud storage systems. IEEE Trans Cloud Comput, 2021, 9: 386–400
39 Wei B, Xiao L M, Wei W, et al. A new adaptive coding selection method for distributed storage systems. IEEE Access, 2018, 6: 13350–13357