IEEE Access
Multidisciplinary : Rapid Review : Open Access Journal

Received June 8, 2020, accepted August 17, 2020, date of publication August 24, 2020, date of current version September 10, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3019108

Towards Building Reliable and Cost-Efficient Distributed Storage Systems

YICHUAN QI[1,2], **DAN FENG**[1,2], **(Senior Member, IEEE), AND BINBING HOU**[3]
[1]Key Laboratory of Information Storage System, Wuhan National Laboratory for Optoelectronics, Wuhan 430074, China
[2]School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan 430074, China
[3]LinkedIn Inc., Sunnyvale, CA 94085, USA

Corresponding author: Yichuan Qi (yichuan0707@hust.edu.cn)

ABSTRACT Reliability and cost are two important targets for distributed storage systems. For many years, numerous schemes have been proposed to improve the reliability or cost of distributed storage systems, and they can be divided into three categories: (1) data redundancy schemes; (2) data placement schemes; and (3) data repair schemes. However, it is still unclear regarding how to build a reliable and cost-efficient distributed storage system, because (i) insufficient considerations on the combinations of different schemes; and (ii) insufficient considerations on failures and recoveries of different subsystems (racks, nodes, disks, and sectors). To measure the reliability and cost caused by different schemes, we design and implement CR-SIM, a Comprehensive Reliability SIMulator for distributed storage systems. It considers various affecting factors, such as the system topology, the data redundancy scheme, the data placement scheme, the data repair scheme, and the failure/recovery models of different subsystems. By using CR-SIM, we conduct various simulation-based experiments, and the experimental results reveal several important findings, which are helpful to build reliable and cost-efficient distributed storage systems. For public use, we have open-sourced our source code at **`https://github.com/yichuan0707/CR-SIM`**.

INDEX TERMS Reliability, cost-efficient, distributed storage system, simulation, data redundancy scheme, data placement scheme, data repair scheme.

I. INTRODUCTION

Today's distributed storage systems generally consist of thousands of commodity servers to provide storage services, such as the Google File System (GFS) [1], the Hadoop Distributed File System (HDFS) [2], and the OpenStack Swift object storage (Swift) [3].

Reliability is critical for distributed storage systems. The services provided by distributed storage systems must guarantee a certain reliability standard. For example, cloud storage services like Windows Azure Storage [4] and Amazon S3 [5] aim to achieve a yearly reliability of 11 9's, i.e., 99.999999999%. Such high reliability is often guaranteed by massive resource consumption (cost), such as large storage overheads or data transferring.

In current distributed storage systems, numerous schemes have been proposed for reliability and cost purposes. And these schemes can be divided into three categories: (i) *data redundancy schemes*, such as the replication (REP) [6], Reed-Solomon (RS) codes [7], Local Repairable Codes (LRC) [8], and Regenerating Codes [9], [10]); (ii) *data placement schemes*, such as the spread placement scheme (SSS) [11], partitioned placement scheme (PSS) [11], and CopySet [12] for data placement on nodes, and flat data placement (Flat) and hierarchical data placement (Hier) [13] for data placement on racks; (iii) *data repair schemes*, such as eager repair (Eager), lazy repair (Lazy) [14], risk-aware failure identification repair (RAFI) [15], and the combination of Lazy and RAFI (Lazy+RAFI) [15].

However, a high-reliability scheme results in a high system cost (such as the cost of storage or network bandwidth) while a low-cost scheme cannot satisfy the reliability requirement. For example, more redundant data greatly improves the reliability at the price of additional storage cost, and the low repair cost of Lazy is based on great reliability loss. Thus, building a reliable and cost-efficient distributed storage system is challenging.

Although prior studies have focused on improving system reliability and reducing system cost, they can not provide sufficient guidelines for building reliable and

The associate editor coordinating the review of this manuscript and approving it for publication was Roberto Nardone.

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/

cost-efficient distributed storage systems. First, there are insufficient considerations on *combinations*. A combination (e.g., RS+SSS+Flat+Eager) is the collection of the data redundancy scheme, data placement scheme, and data repair scheme, which is used to indicate one system's data redundancy, placement, and repair. Most prior studies [8], [10], [16] compare schemes in one category, and several prior studies [13], [15] consider the combination of two categories at most. Second, there are insufficient considerations on failures and recoveries of subsystems (racks, nodes, disks, and sectors). Many studies [11], [15], [17], [18] only consider one subsystem (disk or node) for reliability measurements. And the different repair patterns of systems will bring different recovery models to subsystems. All these incomplete considerations make us doubt the existing conclusions on reliability and cost. In addition, researchers trend to display the improvements of the new schemes they proposed, the defects of these schemes are often habitually hidden.

To understand the reliability and cost in depth and provide guidance, we present a comprehensive simulation-based quantitative study on the reliability and cost of distributed storage systems. For this purpose, we build a comprehensive simulator CR-SIM to measure the reliability and cost of distributed storage systems. It is designed to be comprehensive by accounting for various factors as inputs, including the system architecture, the data redundancy scheme, the data placement scheme, the data repair scheme, as well as failure and recovery models of different subsystems. It reports the probability of data loss as the reliability metric and the storage and repair cost as cost metrics.

Using CR-SIM, we conduct plenty of experiments in terms of combinations by adopting recovery models from HDFS and Swift. These two systems represent two widely used repair patterns. HDFS represents the unified repair pattern, and Swift represents no unified repair pattern. Based on the analysis of the experimental results, we obtain several significant findings:

- The choice of data redundancy scheme affects both the reliability and cost, and reliability of the combination mainly depends on the fault tolerance of the redundancy scheme.
- The choice of data placement scheme on nodes affects reliability but basically does not affect the cost, PSS achieves the highest reliability among three data placement schemes on nodes.
- The choice of data placement scheme on racks affects both reliability and cost, compared with Flat, Hier sacrifices reliability in exchange for repair cost reductions.
- The choice of Lazy greatly decreases reliability in exchange for repair cost reductions when combined with RS, but it cannot obtain repair cost reductions with reliability loss when combined with MSR or LRC.
- The choice of RAFI also decreases reliability in exchange for repair cost reductions.
- The effects of Lazy and RAFI will not be accumulated, the reliability and repair cost of Lazy+RAFI are between Lazy and RAFI (the data redundancy scheme is RS).
- The combination achieves higher reliability under no unified repair pattern than under the unified repair pattern, but its repair cost under these two repair patterns is almost the same.
- The minimum cost combination under the given reliability standard is determined by the pricing model of cost. Under the pricing models of Amazon [19], Azure [20], and Alibaba cloud [21], the eligible combination is MSR+PSS+Flat+Eager for HDFS and MSR+PSS+Hier+Eager for Swift.

Our findings not only reveal the impacts of schemes on reliability and cost, but also guide system researchers and designers to build reliable and cost-efficient distributed storage systems. For public use, our simulator CR-SIM is available at `https://github.com/yichuan0707/CR-SIM`.

The rest of this paper is organized as follows. Section II introduces some background information on distributed storage systems, data redundancy schemes, data placement schemes, and data repair schemes. Section III presents the design of CR-SIM. Section IV evaluates the reliability and cost of all combinations. At last, Section V introduces the related work and Section VI concludes.

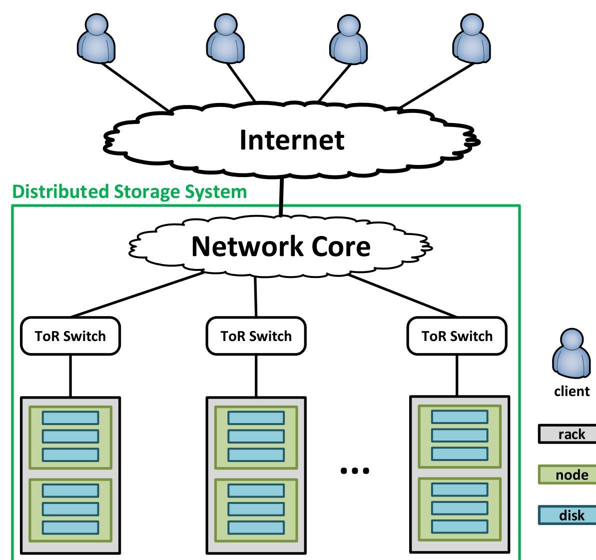


FIGURE 1. Overview of a distributed storage system.

II. BACKGROUND

A. DISTRIBUTED STORAGE SYSTEMS

1) SYSTEM OVERVIEW

Distributed storage systems deliver storage services to clients. Fig. 1 illustrates the overview of a distributed storage system. The system comprises multiple *racks*; each rack holds several to dozens of machines called *nodes* (or *servers*); each node

is attached with one or multiple *disks* that provide storage capacity. Nodes in the same rack are connected by a ToR switch, different racks are connected by a network core which refers to the abstraction of networks. Such a system architecture inherits from previous works [13], [17].

Distributed storage systems organize data as fixed-sized units called *chunks*, and multiple chunks make up a *stripe*. There are millions of stripes in a distributed storage system.

2) RELIABILITY ISSUES

Reliability is a classical problem, but the reliability of a single subsystem (rack, node, disk or sector) is quite different from that of a distributed storage system. Reliability studies [22]–[24] of a single subsystem analysis the failure statistics of the subsystem to obtain its failure distribution and recovery distribution. Root causes of failures and other factors that may affect the distributions will be discussed. Reliability studies [13]–[15] of the distributed storage system adopt the failure/recovery distributions of different subsystems as the inputs to measure the reliability of the whole system. In this paper, we discuss the reliability of the distributed storage system.

The reliability is measured with data loss [12], [13], the corresponding metric is the *probability of data loss*(*PDL*), and *PDL* is defined as:

$$PDL = \frac{N_{lost}}{N_{total}}, \tag{1}$$

N_{lost} is the number of lost chunks, N_{total} is the number of total chunks in the system. Lower *PDL* means the system achieves better reliability.

3) COST ISSUES

In this paper, the cost refers to resource consumption. The total cost of a system can be divided into three parts: computing cost, storage cost, and bandwidth cost. In particular, the bandwidth cost is made up of the bandwidth utilized by the system running in the normal mode and that in the repair mode (i.e., using the data repair scheme to repair failed chunks).

Prior researches [8], [14], [25] only consider storage cost and repair cost because: (i) no matter how to build the system, the read bandwidth cost is fixed; (ii) they aim at the steady-state system - data is once written will not be changed, the write bandwidth cost can be ignored; and (iii) they aim at large file storage, and the computing cost can be treated as fixed because it is not the bottleneck resource. In this paper, we follow the same rules and treat the total cost as the sum of storage cost and repair cost. The *TSC* is the total storage cost during the mission time and it is measured with PiB•month. For example, if one distributed storage system occupies 1.5 PiB storage space for 10 years, its *TSC* is $1.5\ PiB \times 10\ years = 180\ PiB \cdot month$. The *TRC* is the total repair cost during the mission time, and it is measured in PiBs.

We use the price to unify *TSC* and *TRC*. Suppose the price of storage is α per PiB per month and the price of bandwidth is $\lambda \cdot \alpha$ per PiB. λ is the unit price ratio between bandwidth and storage. So, *TC* can be figured out through (2).

$$TC = (TSC + \lambda \cdot TRC) \cdot \alpha \tag{2}$$

We can obtain the values of α and λ from the pricing model of cloud service providers [19]–[21].

B. RELIABILITY AND COST OF SCHEMES

The reliability and cost of one distributed storage system are determined by the adopted combination. Each combination is made up of three parts: redundancy, placement, and repair. In the following, they will be separately introduced.

1) DATA REDUNDANCY SCHEMES

In distributed storage systems, it is essential to maintain data redundancy. The redundant chunks are generated by a given data redundancy scheme and they are the alternatives for failed chunks, which is higher reliability builds upon higher storage cost.

We introduce the concept of *recovery penalty* [15] to explain data redundancy schemes. The recovery penalty is the data transfer for recovering a stripe. And the *recovery penalty factor* is the ratio between the recovery penalty and the size of failed data.

We concentrate on four kinds of data redundancy schemes:

- **Replication (REP):** Replication is a popular data redundancy scheme, it maintains n replicas for each original data chunk. Recovering a failed chunk can be finished by fetching another copy of it, so its fault tolerance is $n - 1$ and recovery penalty factor is 1.
- **Reed-Solomon (RS) codes:** To relieve the n-fold storage overheads of REP, RS codes [7], [26] have been adopted as alternatives. There are two important parameters in RS codes: n and k, where $k < n$. For every k raw uncoded chunks, RS codes encode them into n coded chunks, these n chunks compose a stripe and distribute to n distinct nodes. For RS(n, k), its fault tolerance is $n - k$ and its recovery penalty for single failure or multiple failures is k chunks.

  But, the high recovery penalty of RS codes becomes unbearable in large-scale distributed storage systems [27]. To relieve it, the data redundancy scheme has been improved from two directions: (i) reduce the number of nodes participating in the repair; (ii) reduce the amount of data provided by each repair participating node. The representative of the former is Local Repairable Codes (LRC), and the representative of the latter is regenerating codes.
- **Local Repairable Codes (LRC):** LRC [8], [27] exploits locality to reduce repair cost. In this paper, we focus on the LRC in Windows Azure Storage [8]. It divides k original data chunks into l groups (suppose k is divisible by l) and creates one local parity chunk in each group,

in addition, $n-k-l$ global parity chunks are generated by calculations with k original chunks. In LRC(n, k, l), recovering an original data chunk or a local parity chunk needs to retrieve $\frac{k}{l}$ available chunks from the same group, while recovering a global parity chunk or multiple chunks still needs to retrieve k available chunks from the stripe, i.e., the recovery penalty for single failure is $\frac{k}{l}$ or k chunks. In general, LRC's recovery penalty factor is explained as its average recovery penalty factor, e.g., LRC(10, 6, 2)'s recovery penalty factor for single failure is $\frac{k}{l} \cdot \frac{k+l}{n} + k \cdot \frac{n-k-l}{n} = 3.6$. Similarly, LRC($n, k, l$) has $n-k$ parity chunks in total, but it is unable to recover from all situations of $n-k$ failures. Its fault tolerance is defined as the average fault tolerance, e.g., LRC(10, 6, 2) can repair all three failures and 85.7% four failures [8], so its fault tolerance is 3.857.

- **Minimize Storage Regenerating (MSR) codes:** MSR codes [9], [10] have the same stripe layout and fault tolerance ($n-k$) with RS codes, but it needs an additional parameter d ($k \leq d < n$). When recovering one failed chunk, if the available chunks in the stripe are no less than d, MSR codes read $\frac{1}{d-k+1}$ part from d available chunks (regenerating), the recovery penalty factor is $\frac{d}{d-k+1}$; if the available chunks in the stripe are less than d (but no less than k), the regenerating is disabled, the recovery penalty factor is downgraded to k.

Except these four data redundancy schemes, many other data redundancy schemes will not be discussed, like RAID codes, other regenerating codes, and hybrid schemes. Some RAID codes (e.g., EVENODD [28] and RDP [29]) are double-erasure correcting codes and they cannot provide sufficient reliability for distributed storage systems. And some RAID codes like generalized RDP [30] focus on improving the encoding or decoding speed which is not the key point of distributed storage systems. As for other kinds of regenerating codes (e.g., minimize bandwidth regenerating (MBR) codes [31], simple regenerating codes (SRC) [32], and else), they are more complicated than MSR codes and consume too much more storage resources. The hybrid schemes (e.g., MICS [33]) are built based on the aforementioned representatives, and they put too much emphasis on performance improvement which is not the focus of this paper. Therefore, in this paper, we adopt REP, RS codes, LRC, and MSR codes as the representatives of data redundancy schemes.

The trade-offs between reliability and cost are changing with the data redundancy scheme. From REP to RS codes, higher reliability and lower storage cost come with a much higher repair cost [34], [35]. Compare with RS codes, LRC declares achieving higher reliability and lower repair cost with small additional storage cost [8]; MSR codes achieve higher reliability and lower repair cost under the same storage cost [10]. In general, the trade off between reliability and cost always exists.

2) DATA PLACEMENT SCHEMES

The distributed storage system organizes data with chunks and stripes, so the data placement scheme determines how to place stripes and chunks to nodes and racks. In this part, we introduce data placement schemes on nodes and racks, respectively.

To introduce data placement schemes on nodes, a metric scatter width (w) is defined. It represents the number of nodes for participating single node failure repairs, i.e., one node repair needs to fetch data from w nodes. The reliability and cost of data placement schemes on nodes are affected by w.

In distributed storage systems, there are three data placement schemes on nodes:

- **Spread Placement Scheme (SSS):** In SSS, each stripe's n chunks have been stored on n nodes which are randomly selected from N nodes (suppose the system has N storage nodes and $N \gg n$). Typically, number of chunks in one node is greater than N, so $w = N - 1$. SSS has been adopted by many systems, like QFS [36] and RAMCloud [37].
- **Partitioned Placement Scheme (PSS):** PSS [11] divides N nodes into disjoint partitions, each partition contains n nodes, and each stripe is stored on a designated partition, i.e., $w = n - 1$. PSS has been adopted by Facebook [38].
- **CopySet:** CopySet [12] is the moderate data placement scheme between SSS and PSS, it distributes stripes to nodes on the principle of ensuring the given $w(n-1 < w < N-1)$, $w = 2(n-1)$ is a choice which has been repeatedly mentioned.

The trade-offs between reliability and cost are changing with the data placement scheme on nodes. From SSS to CopySet to PSS, w becomes smaller and smaller, more intensive data placement decreases the data loss frequency but increases the losing chunks in each incident. These two conflicting indicators make the changes on reliability and cost unclear.

For data placement schemes on racks, a metric r is defined. It means chunks in each stripe reside in $r(r \leq n)$ distinct racks. The reliability and cost of data placement schemes on racks are determined by r.

There are two data placement schemes on racks in distributed storage systems:

- **Flat Placement Scheme (Flat):** In Flat, $r = n$, i.e., n chunks in each stripe have been put on n different nodes which belong to n distinct racks (one chunk per rack). Flat has been widely used in practical distributed storage systems [8], [39], [40].
- **Hierarchical Placement Scheme (Hier):** In Hier, $r < n$, i.e., the n chunks in each stripe have been put on n different nodes which belong to r distinct racks, each of them holds $\frac{n}{r}$ chunks. Recovering any failed chunk can get part of chunks from the same rack which stores the failed chunk, thus reducing the repair cost. Hier has been adopted by HDFS [2].

The trade-offs between reliability and cost are changing with the data placement scheme on racks. Using Flat, reliability benefits from maximum fault tolerance against node and rack failures. Compared with Flat, Hier reduces the repair cost, but its reliability change is uncertain because the reduced repair cost improves data reliability and the reduced rack-level fault tolerance reduces data reliability.

3) DATA REPAIR SCHEMES

The data repair scheme determines when to start data recovery, and the start time of data recovery influences both reliability and cost.

In distributed storage systems, there are four representative data repair schemes:

- **Eager repair scheme (Eager):** Eager is the default data repair scheme in many distributed storage systems, which is the system repairs the failed stripe as soon as it is detected. It ensures high reliability, but the repair cost is high.
- **Lazy repair scheme (Lazy):** Adopting Lazy [14], the system will not repair the failed stripe until the number of failures in one stripe reaches a given threshold r_{th} ($r_{th} < n-k$). If $r_{th} = 2$, stripes with one failed chunk will not be repaired.
- **Risk awareness failure identification repair scheme (RAFI):** RAFI [15] defines T_i ($i = 1, 2, \cdots, n-k$) as the timeout threshold for stripe which has i failures. The more failures in the stripe, the higher lost risk it has. So, RAFI gives a long timeout threshold to low risk stripes to reduce repair cost and a short timeout threshold to high risk stripes to improve reliability (T_i decreases with i increase).
- **The combination of Lazy and RAFI (Lazy+RAFI):** Lazy and RAFI can be enabled at the same time, that is Lazy+RAFI.

The trade-offs between reliability and cost are changing with the data repair schemes. On the basis of Eager, Lazy obtains repair cost reductions at the expense of reliability decrease. RAFI improves the data loss and repair cost caused by node failures by adopting different timeout thresholds for stripes with different risks [15], but the effects caused by failures of other subsystems are uncertain. As for Lazy+RAFI, its reliability and cost is also unclear.

At last, we summarize the reliability and cost of all the above mentioned schemes in Table 1. In each category, the first line displays the default scheme (the widely used scheme) in the category, "–" means the corresponding scheme has the same reliability or cost with the default scheme, ↑ means increase and ↓ means decrease. Some cells contain both ↑ and ↓, it means the consequences of the scheme include both increase factor and decrease factor. The final consequences should be obtained through analysis. What's more, the variations of reliability and cost are unclear when consider from the perspective of combinations (schemes from different categories work together). These two points are the main contributions of this paper.

TABLE 1. Collections of schemes.

Category	Scheme	Reliability	Storage cost	Repair cost
Data Redundancy Schemes	RS(default)	–	–	–
	REP	↓	↑	↓
	LRC	↑	↑	↓
	MSR	↑	–	↓
Data Placement Schemes on Nodes	SSS(default)	–	–	–
	PSS	↑↓	–	↑↓
	CopySet	↑↓	–	↑↓
Data Placement Schemes on Racks	Flat(default)	–	–	–
	Hier	↑↓	–	↓
Data Repair Schemes	Eager(default)	–	–	–
	Lazy	↓	–	↓
	RAFI	↑↓	–	↓
	Lazy + RAFI	↑↓	–	↓

III. DESIGN OF CR-SIM

To measure the reliability and cost of distributed storage systems, we build a comprehensive simulator CR-SIM. We are free to set the data redundancy scheme, the data placement scheme, and the data repair scheme in CR-SIM.

A. ARCHITECTURE OVERVIEW

CR-SIM has three main parts which work for data distribution, event generating, and event handling. Accordingly, CR-SIM implements three corresponding modules: *data distribution*, *event generator*, and *event handler*. The data distribution module distributes stripes to disks based on the adopted combination. Like other simulators [13], [14], CR-SIM uses *events* to represent the failures and recoveries of subsystems. The event generator generates failure and recovery events of all subsystems. The event handler is responsible for handling all the events, and it outputs the final *metrics* on reliability and cost (*PDL*, *TSC*, and *TRC*).

Fig. 2 illustrates the architecture of CR-SIM. At the top, CR-SIM performs simulation over a fully large number of iterations. In each iteration, CR-SIM conducts the following three steps to get the final metrics:

1) CR-SIM takes the basic simulation settings, system architecture, data redundancy settings, data placement settings, and data repair settings as inputs to build the system model and distribute chunks (①). All these settings are recorded in a configuration file.
2) CR-SIM generates failure and recovery events of all subsystems (racks, nodes, disks, and sectors) based on the corresponding models (recorded in an XML file) until the mission time is reached (②③). All generated events are put into an event queue with chronological order (④).
3) CR-SIM handles all events in the event queue and collects the information about data loss and repair cost (⑤⑥). In this process, the subsystem's recovery event has to be transformed into actual data recovery event based on the repair settings and inserts into the event queue again (⑦). With simple calculations, CR-SIM gets the reliability and cost metrics in one iteration (⑧).

Finally, the final values of reliability and cost metrics are averages of all iterations.

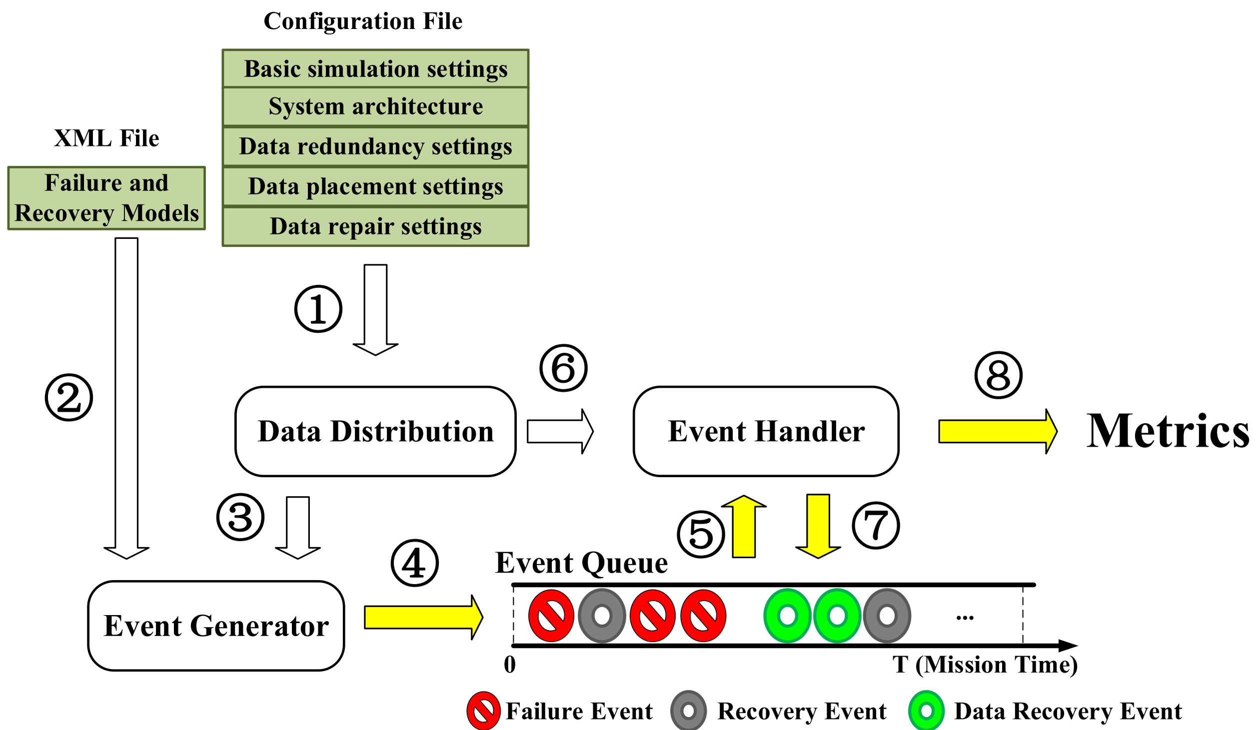


FIGURE 2. Architecture of CR-SIM.

All parameters in the configuration file are divided into five parts. Table 2 lists the symbols and definitions of all parameters.

TABLE 2. Parameters in the configuration file.

Basic simulation settings	S	total data amount, in PiBs
	T	the mission time
	N_i	the number of iterations
System architecture	C	total capacity of system, in PiBs
	D_n	the disks per nodes
	N_r	the nodes per rack
	R	the rack count
	b	the chunk size, in MiBs
	B	the bandwidth
Data redundancy settings	n	number of chunks in each stripe
	k	number of original data chunks in each stripe
	l	number of groups (only for LRC)
	d	number of participating nodes in regenerating (only for MSR)
Data placement settings	w	the scatter width
	r	number of distinct racks
Data repair settings	r_{th}	the recovery threshold (only for Lazy)
	T_i	failure identification time thresholds (only for RAFI)

The total chunks in the system (N_{total}) and the total storage cost (TSC) can be figured out through these parameters, which is (3) and (4).

$$N_{total} = \lceil \frac{2^{30} \cdot n \cdot S}{k \cdot b} \rceil \quad (3)$$

$$TSC = \frac{n}{k} \cdot S \cdot T \quad (4)$$

B. DATA DISTRIBUTION

Based on the parameters in the configuration file, CR-SIM distributes chunks to subsystems. First, CR-SIM builds a tree structure to represent the architecture of the system. The root of the tree represents the system. And its R children are racks; each rack's N_r children are nodes in it; each node's D_n children are disks in it. Second, CR-SIM calculates the total stripe amount ($\lceil \frac{2^{30} S}{kb} \rceil$) and distributes chunks of all stripes to disks. The indexes of stripes will be recorded as the children of disks where these stripes are stored. Note that the data distribution varies across iterations.

C. EVENT GENERATING

CR-SIM considers failure and recovery events from four levels of subsystems: racks, nodes, disks, and sectors. Failures can be either *transient* or *permanent*. The transient failure means a subsystem is temporary unavailable without actual data loss (e.g., due to network breakdown, reboots, or node maintenance). The permanent failure means a subsystem failure brings permanent data loss (e.g., due to disk or node crashes). All disk and sector failures are permanent failures. Node failures include both transient and permanent failures. Only transient rack failures are taken into account like previous works [13], [14]. For recoveries, CR-SIM adopts realistic recovery models that come from practical systems to generate recovery events. For each subsystem, the time-to-repair (TTR) is made up with two parts: the failure identification time (T_{iden}) and data transferring time (T_{tran}), that is (5). For transient failures, T_{iden} obtains from failure statistics, and $T_{tran} = 0$; for permanent failures, T_{iden} is

determined by the failure identification time threshold of different subsystems, T_{tran} can be figured out through (6), $a \cdot B$ is the aggregate bandwidth. The value of a is determined by the subsystem and the data placement scheme. For sector repairs, whatever the data placement scheme on nodes is, when the data placement on racks is Flat, $a = k$, when the data placement scheme on racks is Hier, $a = r - 1$. For disk or node repairs, when the data placement scheme on racks is Flat, $a = min(w, R)$; when the data placement scheme on racks is Hier, if the data placement scheme on nodes is SSS, $a = R$, if the data placement scheme on nodes is PSS or CopySet, $a = r - 1$.

$$TTR = T_{iden} + T_{tran} \tag{5}$$

$$T_{tran} = \frac{\text{recovery penalty}}{a \cdot B}. \tag{6}$$

The considerations on failures and recoveries are more comprehensive than prior studies [8], [13], [15].

In CR-SIM, there are five event types: (i) transient failure; (ii) permanent failure; (iii) transient failure recovery; (iv) permanent failure recovery; and (v) actual data recovery. And each event is made up with three attributes: (i) the event occurrence timestamp (t_o); (ii) the event type (*type*); and (iii) the chunks affected by the event (*c_ids*).

The former four event types correspond to the subsystem's failure or recovery, they are generated by the *event generator* (④ in Fig. 2). Each subsystem's failure event and recovery event appear in pairs (one subsystem's transient failure and transient failure recovery, or its permanent failure and permanent failure recovery). If the failure occurrence timestamp is t_o^F, the corresponding recovery timestamp is $t_o^R = t_o^F + TTR$. All generated events are put into the event queue with chronological order.

The actual data recovery will be introduced in the next section.

D. EVENT HANDLING

CR-SIM maintains the real-time states of chunks in all stripes. Each chunk is associated with three states during the simulation: (i) *normal* (chunk is accessible and no failure); (ii) *unavailable* (chunk encounters transient failure); and (iii) *lost* (chunk encounters permanent failure). In terms of severity, *normal* is the least severe, *unavailable* is the middle severe, *lost* is the largest severe. If one node fails, the states of chunks on it will be updated only if the states becomes more severe, that is, the *normal* or *unavailable* states become *lost* states for a permanent node failure; or *normal* states become *unavailable* states for a transient node failure, but the *lost* states (the corresponding chunks are already hit by disk failure or sector failure) remain unchanged.

The event is handled as follows. The event handler module pops event (t_o, *type*, *c_ids*) from the event queue. If the event type (*type*) is permanent failure recovery, the event will be translated into an actual data recovery event and inserted into the event queue again (⑦ in Fig. 2). The reason that promotes us to introduce the actual data recovery event into CR-SIM is the support of Lazy, RAFI, and Lazy+RAFI. These schemes consciously change the start time of actual data recovery, which make the subsystem recovery and the actual data recovery not synchronized. If *type* is not permanent failure recovery, CR-SIM handles the event as follows:

1) collects all indexes of stripes which have chunk in *c_ids* (stripes affected by the event) and also collects the corresponding chunks' indexes (⑥ in Fig. 2);
2) changes the states of chunks in *c_ids* based on *type*;
3) during the handling process, if the lost chunk has been recovered, the recovery penalty is added to *TRC*, if one stripe becomes lost (the number of lost chunks in it exceed the stripe's fault tolerance), the stripe's information will be recorded.

Based on the recorded information, CR-SIM outputs reliability and cost metrics.

IV. EVALUATIONS

In this section, we present the reliability and cost of different combinations. In the following, we introduce all required parameters for simulations at first. Then, we study the trade-offs between reliability and cost for changing each category in the combination, which is changing the data redundancy scheme, the data placement scheme or the data repair scheme. At last, we discuss the reliability and cost for systems with different repair patterns. What's more, we point out the minimum cost combination under the given reliability standard for different repair patterns.

A. SIMULATION PARAMETERS

In this section, we introduce the effects of different parameters at first, then explain why we adopt the corresponding parameters for simulations. When we discuss the influence of one parameter, all other parameters remain unchanged.

1) BASIC SIMULATION SETTINGS

Larger data scale (S), longer mission time (T) or higher number of iterations (N_i) can help us to obtain more accurate simulation results, but all of them establish on much higher simulation time. To obtain accurate simulation results in an acceptable time, we give moderate values to the three parameters, i.e., $S = 1\ PiB$, $T = 10\ years$, and $N_i = 10,000$. The three values have been adopted by prior study [14].

2) SYSTEM ARCHITECTURE

The reliability and cost are hard to be affected by the system architecture. The simulation results show that higher rack count (R) incurs higher reliability due to higher aggregate bandwidth (see (6) and (5)), while higher nodes per rack (N_r) or higher disks per node (D_n) cannot affect reliability because the failure and recovery time of subsystems remain unchanged. Besides, the reliability variation brought by the change of R is not significant. As for the cost, it doesn't change with the system architecture (R, N_r or D_n) because the total data amount (S) and the data redundancy scheme

remain the same. Therefore, we set $R = 60$, $N_r = 6$, $D_n = 3$ to facilitate data distribution. Suppose each disk's capacity is 2TB, we can figure out that $C = 2\ PiB$. Both the bandwidth (B) and chunk size (b) affect reliability. We get their values from practical systems, i.e., $B = 200Mbps$ [17] and $b = 256\ MiB$ [2].

3) DATA REDUNDANCY SETTINGS

For any data redundancy scheme, higher fault tolerance ($n-k$) brings higher data reliability at the price of higher storage cost; under the same $n-k$, larger k brings lower storage cost at the price of reliability loss and higher repair cost. Therefore, practical distributed storage systems [6], [36], [39] adopt moderate $n-k$ and k, i.e., $k = 6$ or $k = 10$, and $n-k$ between 2 and 4. In our simulations, we follow the same principle. For replicated data redundancy scheme, if n is not given, REP means triple replication ($n = 3$), which is the default redundancy scheme for many systems [2], [6], [12], and the N_r increases to 12 to provide enough capacity (to ensure $C > n \cdot S$). For erasure coded data redundancy scheme, if (n, k) is not given, RS represents RS(9, 6) [36], MSR represents MSR(9, 6, 8) ($d = n-1$ the optimal choice because it has the lowest repair cost [41]), and LRC represents LRC(10, 6, 2) [8]. In the discussions of data redundancy schemes, other (n, k)s will be mentioned for comparisons, like RS(10, 6) [42], RS(14, 10) [39], MSR(14, 10, 13), and LRC(16, 12, 2) [13].

4) DATA PLACEMENT SETTINGS

For data placement schemes, only CopySet and Hier need to specify the parameter. For CopySet, simulation results show that the reliability decreases with the increase of w, so we use a low w value as the default. Specifically, if w is not given, CopySet means CopySet($w = 2(n-1)$) [12]. For Hier, simulation results show that reliability and repair cost increase with the increase of r, so we use a moderate value as the default. If r is not given, Hier means Hier($r = 3$) [17].

5) DATA REPAIR SETTINGS

For Lazy, a larger recovery threshold (r_{th}) greatly damages the reliability in exchange for more repair cost reductions. To guarantee reliability and $r_{th} \le n-k$, $r_{th} = 2$ [14] is the default for Lazy, it means system launches the repair process when one stripe has at least two failures.

For RAFI, higher T_i reduces repair cost but increases the risk of data loss (lower reliability). Therefore, we cite the moderate T_is from prior study [15] as the default, i.e., $T_1 = 1\ hour$, $T_2 = 15\ minutes$, $T_i = 2\ minutes (2 < i \le n-k)$. It means stripe will be recovered when it has one failure and lasts more than 1 hour, or it has two failures and lasts more than 15 minutes, or it has more than two failures (no more than $n-k$) and lasts more than 2 minutes.

6) FAILURE AND RECOVERY MODELS

The failure and recovery models of different subsystems (contents in the XML file) come from production traces and production systems. Table 3 summarizes all these models. For recovery models, only T_{iden}s are given. TTRs for all subsystems can be figured out through (5) and (6). Except that, it should be noted that we give T_{iden}s of two different systems: HDFS and Swift, which represent two repair patterns. HDFS represents unified repair pattern, which means it has unified knowledge of failures, so all storage nodes periodically report its failure information to the metadata server and the metadata server launches repair operations. On the contrary, Swift represents no unified repair pattern, which means it has no unified knowledge of failures, so each storage node periodically detects and repairs the failures of its adjacent node. Both Lazy and RAFI establish on unified knowledge of failures, so HDFS supports all the four data repair schemes, but Swift only supports Eager.

Failure models are distributions which are the statistical results of traces.

- **Sector failure:** It has been considered that sector failures follow a Possion process [23], [43], so we can use Exponential distribution (mean-time-to-failure (MTTF) of sectors is 1 year [43]) as the sector failure model.
- **Disk failure:** The MTTF of disks ranges from few years [27] to tens of years [14], [16], [17]. We use a Weibull distribution with a characteristic life of 10 years to model the time-to-failure of disks [13].
- **Node failure:** The statistics of Yahoo! cluster [44] indicate that 0.8% nodes permanently fail each month. Thus, we set the MTTF of permanent node failures as 125 months. According to Google's research [16], the MTTF of transient node failures is about 4 months. We set the time-to-failure of the permanent and transient node failure as exponentially distributed with means 125 months and 4 months, respectively.
- **Transient rack failure:** We follow prior studies [13], [14], [16] to set the MTTF of rack failures as 10 years and use the corresponding Exponential distribution as the failure model.

For recovery models, the T_{iden}s of permanent failures and transient failures are come from production systems and production traces, respectively.

- **Sector failure recovery:** Both HDFS and Swift have been widely deployed, so we adopt their settings directly. That is, HDFS [2] scans all disks every 3 weeks for sector failures, and each Swift node [3] scans disks every 40 hours for sector failures. Due to the randomness of sector failures, T_{iden} of HDFS is a random value between 0 and 3 weeks; T_{iden} of Swift is a random value between 0 and 40 hours.
- **Disk failure recovery:** Similarly, HDFS refreshes states of all disks every 12 hours, and each Swift node detects the states of disks every hour. For disk failure recovery, T_{iden} of HDFS is a random value between 0 and 12 hours; T_{iden} of Swift is a random value between 0 and 1 hours.
- **Node failure recovery:** In distributed storage systems, communications between nodes are frequent so that

the node failure will be detected very soon (in several seconds). We just need a fixed unavailable time threshold (i.e., 15 minutes [16], [45]) to distinguish transient and permanent node failure. Therefore, the T_{iden} of permanent node failure recovery is 15 minutes. The T_{iden} of transient node failure recovery follows the Weibull distribution with a characteristic life of 0.1 hour [14].

- **Transient rack failure recovery:** The T_{iden} follow the Weibull distribution with a characteristic life of 24 hours [13], [14].

Simulation results are presented with the following principles. We use the log scale to display *PDL* and the normal scale to display cost metrics. We set $S = 1PiB$ for all combinations, so *TSC* is determined by $\frac{n}{k}$ (see (4)) which is very clear. It will not be specifically illustrated. And *TC* is represented by the value of $\frac{TC}{\alpha}$.

B. EFFECTS OF SCHEMES

We study the reliability and cost trade-offs in terms of combinations. We use the HDFS as the system paradigm because it supports all-round combinations. The comparisons between HDFS and Swift are put in the next section. Combinations follow the same rules will be moderately omitted. For simplicity, the default scheme/schemes (see Table 1) in one combination can be omitted, e.g., MSR+SSS+Flat+Lazy+RAFI can be denoted as MSR+Lazy+RAFI. And RS+SSS+Flat+Eager is the *baseline*, it can be denoted as anyone of RS, SSS, Flat, and Eager when compares with other combinations.

1) EFFECTS OF DATA REDUNDANCY SCHEMES

First, we study the reliability and cost of combinations when the data redundancy scheme changes. For each combination, its data redundancy scheme indicates three important targets: storage overheads ($\frac{n}{k}$), fault tolerance ($\leq n-k$), and recovery penalty factor ($\leq \ k$) (see Section II-B1). Theoretically, the reliability and cost will be affected by all of them. Higher fault tolerance, higher storage overheads or lower recovery penalty factor brings higher reliability. Higher storage overheads brings more repairs, and higher recovery penalty factor means higher recovery penalty for each repair, so both of them enlarge the repair cost. And higher storage overheads mean higher storage cost. In this part, we compare plenty of combinations which cover different data redundancy schemes. The impacts of the three targets will be compared and testified. The results are shown in Fig. 3.

We can testify the theoretical results on reliability and repair cost from Fig. 3. From Fig. 3a, we can see the *PDL* is mainly determined by the fault tolerance. When the fault tolerance is increased by 1, the *PDL* decreases by 2 to 3 orders of magnitude. The only exception is LRC(16, 12, 2), its fault tolerance is close to LRC(10, 6, 2) (3.862 for LRC(16, 12, 2), 3.857 for LRC(10, 6, 2)), but its *PDL* is about 1 order of magnitude higher than LRC(10, 6, 2). This is because LRC(16, 12, 2) has lower storage overheads (1.33 for LRC(16, 12, 2), 1.67 for LRC(10, 6, 2)) and higher recovery penalty factor (6.75 for LRC(16, 12, 2), 3.6 for LRC(10, 6, 2)) than LRC(10, 6, 2). Both of them reduce the reliability, so its *PDL* increases. Either lower recovery penalty factor or higher storage overheads brings lower *PDL*, we can observe from the comparisons of MSR(9, 6, 8) and RS(9, 6), MSR(14, 10, 13) and RS(14, 10), and RS(14, 10) and RS(10, 6). The experimental results are in complete agreement with the theoretical results. However, the reliability change brought by changing the storage overheads or recovery penalty factor is much less than that brought by changing the fault tolerance. From Fig. 3b, we can see the *TRC* is strictly proportional to storage overheads and recovery penalty factor. It is also consistent with the theoretical results.

In summary, **the choice of data redundancy scheme in one combination affects both reliability and cost, and reliability of the combination mainly depends on the fault tolerance of the redundancy scheme (the fault tolerance increases by 1, *PDL* decreases by 2 to 3 orders of magnitude).**

2) EFFECTS OF DATA PLACEMENT SCHEMES

Second, we discuss the reliability and cost of combinations when the data placement scheme changes. We discuss data placement schemes on nodes at first, then data placement schemes on racks.

As we mentioned in Section II-B2, the data placement scheme on nodes affects the data loss frequency and the lost chunks in each incident, which makes its effects on reliability and cost hazy. Fig. 4 illustrates the reliability and cost of multiple combinations which contain different data placement schemes on nodes (SSS, CopySet or PSS).

We can see the reliability variations for changing the data placement scheme on nodes from Fig. 4a. To understand the effects of the data placement scheme on nodes in depth, we define the *failed iteration*. It means the iteration which encounters data loss, no matter how many lost stripes in the iteration. The right-Y axis of Fig. 4a shows the number of failed iterations for each combination (CR-SIM runs N_i = 10, 000 iterations for each combination, see Section IV-A). From SSS to CopySet to PSS, w becomes smaller and smaller. The system contains more disjoint partitions, and multiple failures are less likely to accumulate in one partition. It means there are less failed iterations (see read part in Fig. 4a) but some failed iterations contains more failures. However, SSS has a higher *PDL* than CopySet and CopySet has a higher *PDL* than PSS regardless works with any data redundancy scheme, and more specifically the *PDL* reduces by up to 1 order of magnitude from SSS to PSS. It tells us the reliability is mainly determined by the number of failed iterations.

We can see the cost variations for changing the data placement scheme on nodes from Fig. 4b. As we can see from the figure, the gap of *TRC* between the three schemes is very tiny (less than 1%). Such a tiny gap can be explained from two aspects. First, the storage overheads and recovery penalty factor are not affected by changing the data placement

TABLE 3. Failure and Recovery Models.

Failure Type	Time-to-failure(TTF)	T_{iden} in HDFS	T_{iden} in Swift
Sector Failures	Exp($\frac{1}{1\ \text{year}}$)	Rand(0, 3 weeks)	Rand(0,40 hours)
Disk Failures	W(0, 10 years, 1.12)	Rand(0, 12 hours)	Rand(0,1 hour)
Permanent Node Failures	Exp($\frac{1}{125\ \text{months}}$)	15 minutes	
Transient Node Failures	Exp($\frac{1}{4\ \text{months}}$)	W(0.5, 0.1 hour, 1)	
Transient Rack Failures	Exp($\frac{1}{10\ \text{years}}$)	W(10, 24 hours, 1)	

W(γ, η, β) denotes a Weibull distribution with the location parameter γ, the characteristic life η, and the shape parameter β;
Exp(λ) denotes a Exponential distribution with the rate parameter λ.
Rand(a, b) denotes a random value between a and b.

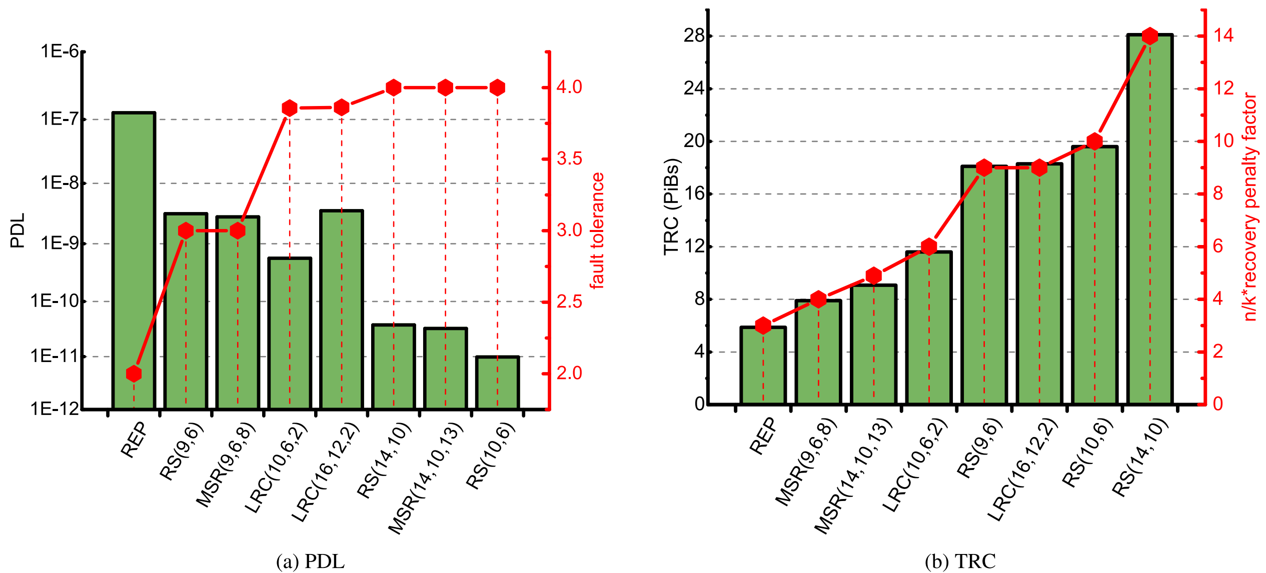


FIGURE 3. Reliability and cost for changing the data redundancy scheme.

scheme on nodes, so *TRC* basically keeps steady. Second, failure models for different combinations are the same, so the degraded stripe amount during T is fixed. The vast majority of of these stripes are recovered thus increasing *TRC*, others are lost. From SSS to CopySet to PSS, data loss becomes less and less, and *TRC* should be increased. But in fact, the incremental on *TRC* which caused by reduced data loss is too tiny (less than 0.1%) to be observed, and it is even less than the deviations between iterations. So, we deem the *TRC* of SSS, CopySet, and PSS is almost the same. The *TSC* is unchanged because the storage overheads ($\frac{n}{k}$) remains unchanged.

In summary, **the choice of data placement scheme on nodes in one combination affects reliability but basically not affect cost, PSS achieves the highest reliability among the three data placement schemes (*PDL* can reduce by up to 1 order of magnitude).**

Then, we study the reliability and cost for changing the data placement scheme on racks. From Section II-B2, we know that Hier sacrifices tolerance against rack failures for repair cost reductions. These two factors have opposite effects on reliability. To figure out the results, we compare a lot of combinations contain different data placement schemes on racks (Flat or Hier). The results are collected by Fig. 5.

We can see the reliability results from Fig. 5a. Either working with default schemes or other non-default schemes, Hier can reduce *PDL* by up to 1 order of magnitude compared with Flat. It means the *PDL* decrease brought by repair cost

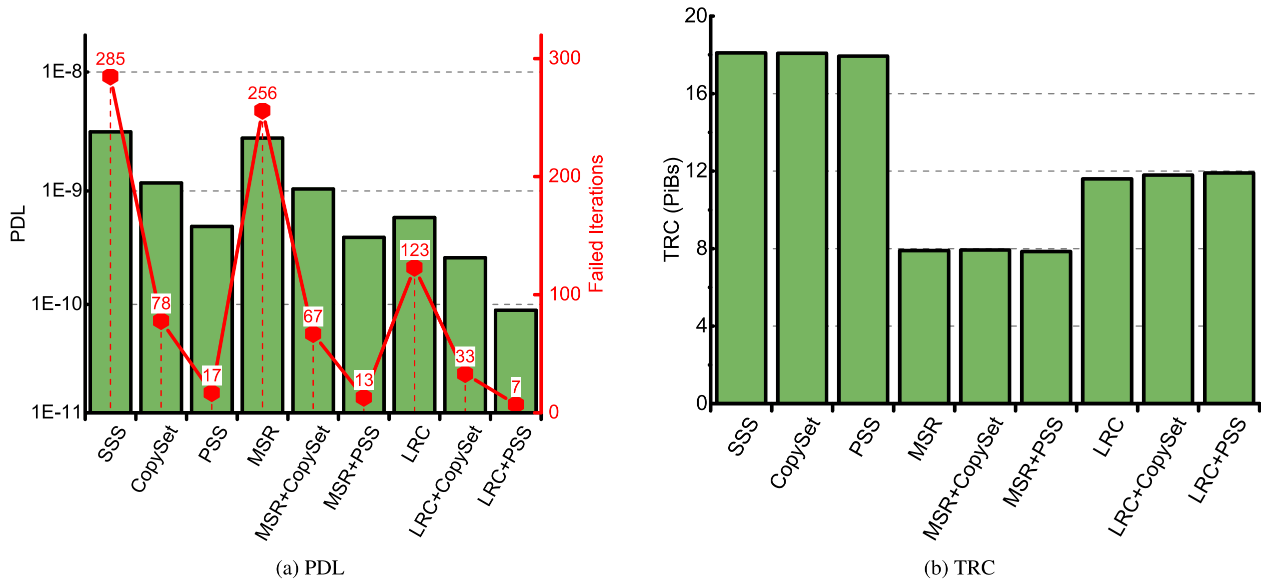


FIGURE 4. **Reliability and repair cost for changing the data placement scheme on nodes.**

reductions is much less than the *PDL* increase brought by reduced rack-level fault tolerance.

As analyzed, we can see Hier reduces *TRC* from Fig. 5b. In detail, the *TRC* reductions are determined by the data redundancy scheme in the combination. When working with MSR(n, k, d), RS(n, k), LRC(n, k, l), and REP, the number of repair required chunks is d, k, $\frac{k}{l}$, and 1, respectively. Among these chunks, Hier reduces $\frac{n}{r} - 1$ chunks. So, compared with Flat, Hier reduces *TRC* by about 25%, 34%, 50%, and 50% for MSR, RS, LRC and REP$(r = 2)$, respectively. It means Hier obtains more *TRC* reductions when the combination has less repair required chunks.

In summary, **the choice of data placement scheme on racks in one combination affects both reliability and cost. Compared with Flat, Hier decreases reliability (*PDL* increases by 1 order of magnitude) in exchange for repair cost reductions (*TRC* decreases by about 25%∼50%), and more repair cost reductions are brought by less repair required chunks.**

3) EFFECTS OF DATA REPAIR SCHEMES

At last, we study the reliability and cost of the combination for changing the data repair scheme. From Section II-B3, we know that Lazy sacrifices reliability for repair cost reductions and RAFI [15] claims to improve reliability and repair cost at the same time. To figure out the results, we compare plenty of combinations which contain different data repair schemes (Eager, Lazy, RAFI, or Lazy+RAFI), and Fig. 6 shows the results.

We can see the reliability and repair cost for different combinations which contain different data repair schemes from Fig. 6a and Fig. 6b, respectively. Compared with Eager, Lazy increases *PDL* by 3 orders of magnitude and reduces *TRC* by 47.5%. This observation is the same with prior study [14]. But what is not mentioned in prior studies is that MSR(9, 6, 8)+Lazy (denotes as MSR+Lazy in Fig. 6) has the identical *PDL* and *TRC* with RS(9, 6)+Lazy (denotes as Lazy in Fig. 6), and LRC(10, 6, 2)+Lazy has the identical *PDL* (3.78e-8) and *TRC* (10.7 PiB) with RS(10, 6)+Lazy. From these observations, we find the advantages of MSR/LRC and Lazy can not be simultaneously obtained. It can be explained with their data repair process. For Lazy, $r_{th} = 2$, but MSR cannot repair two failures with regenerating, and LRC cannot repair two failures with the local repair. To repair two failures, RS+Lazy, MSR+Lazy, and LRC+Lazy have to read k available chunks out of n chunks, so they have the same *PDL* and *TRC* under the same (n, k).

In summary, **the choice of Lazy in one combination greatly decreases reliability in exchange for repair cost reductions when combined with RS, but it cannot obtain repair cost reductions with reliability loss when combined with MSR or LRC.**

Compared with Eager, RAFI increases *PDL* by about 40%∼50% and reduces *TRC* by about 7%∼10%. Unlike the conclusion in prior study [15], RAFI decreases but not increases the reliability (*PDL* growth). RAFI only improves the node failures, so it only reduces the data loss and repair cost caused by node failures. But, the conscious delayed node repairs bring more data loss when they encounter other subsystem failures. The prior study [15] only considered node failures but ignored others, so it came to a different conclusion.

In summary, **the choice of RAFI in one combination also decreases reliability in exchange for repair cost reductions.**

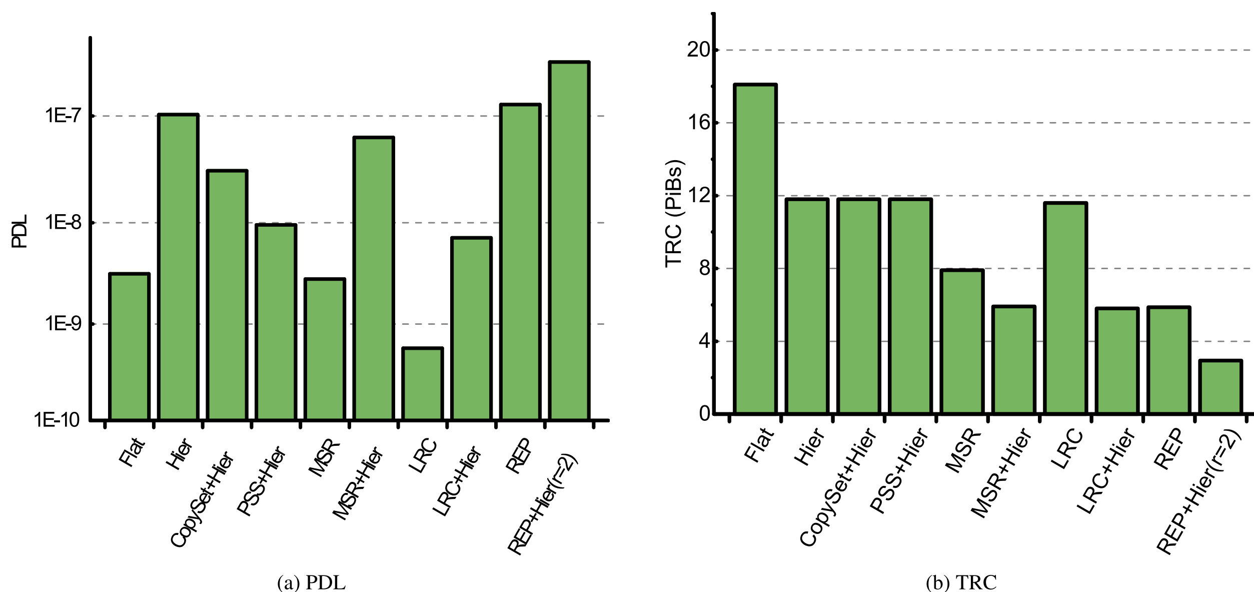


FIGURE 5. Reliability and repair cost for changing the data placement scheme on racks.

Both Lazy and RAFI sacrifice reliability in exchange for repair cost reductions, but Lazy+RAFI is not sacrificing more reliability to reduce more repair cost (MSR/LRC will change the effects of Lazy, so we suppose the data redundancy scheme is RS). On the contrary, *PDL* and *TRC* of Lazy+RAFI are between Lazy and RAFI. In other words, the effects of Lazy and RAFI cannot be accumulated. Lazy+RAFI has more repair opportunity than Lazy but less repair opportunity than RAFI, and less simultaneous repair operations than Lazy but more than RAFI, so its *PDL* and *TRC* are between Lazy and RAFI.

In summary, **the effects of Lazy and RAFI cannot be accumulated in one combination, the reliability and repair cost of Lazy+RAFI are between Lazy and RAFI (the data redundancy scheme is RS).**

C. COMBINATIONS IN HDFS AND SWIFT

Next, we compare the reliability and cost of the same combination in two different system paradigms (HDFS and Swift) which represent two repair patterns. Through the comparisons, we want to achieve three goals: (i) understand the reliability and cost of combinations when the repair pattern changes; (ii) testify the above findings (except the findings of data repair schemes) are valid or not in Swift; (iii) figure out the minimum cost combination under a given reliability standard in the two repair patterns. The three goals are accomplished by the following three subsections, respectively.

1) EFFECTS OF REPAIR PATTERNS

At first, we compare the reliability and cost of the same combination in HDFS and Swift, and these two systems represent two repair patterns. Fig. 7 depicts the reliability and repair cost in the two systems for many combinations. The combination which contains Lazy, RAFI, or Lazy+RAFI is excluded because they are not supported by Swift. Only part of the remaining combinations is listed for lack of space.

We can see from Fig. 7 that one combination's *PDL* is about 1∼2 orders of magnitude lower in Swift than in HDFS, and it basically has the identical *TRC* in Swift and HDFS. From Table 3, we know that Swift has lower T_{iden} than HDFS for permanent failures, so its *PDL* is lower. As for their nearly identical *TRC*, it can be explained by the same reason as data placement schemes on nodes. The corresponding contents can be seen from Section IV-B2, we will not repeat it here.

Another observation is that the combination which has lower recovery penalty factor brings more *PDL* reductions from HDFS to Swift. For example, from HDFS pattern to Swift pattern, the *PDL* reductions for MSR or LRC are larger than that for RS (see Fig. 6) because MSR/LRC has a lower recovery penalty factor than RS. And we have the same observation between PSS+Hier and MSR+PSS+Hier.

In summary, **one combination achieves higher reliability under no unified repair pattern than under the unified repair pattern, but its cost under these two repair patterns is almost the same. From unified repair pattern to no unified repair pattern, the combination which has lower recovery penalty gets more reliability improvements.**

2) FINDINGS IN SWIFT

All findings about the effects of schemes in combinations (see Section IV-B) are established on HDFS pattern, so we check the effects of data redundancy schemes and data placement schemes in Swift pattern. Although the values of *PDL* change, but all findings are still valid in Swift. That means all findings remain regardless of the system repair pattern.

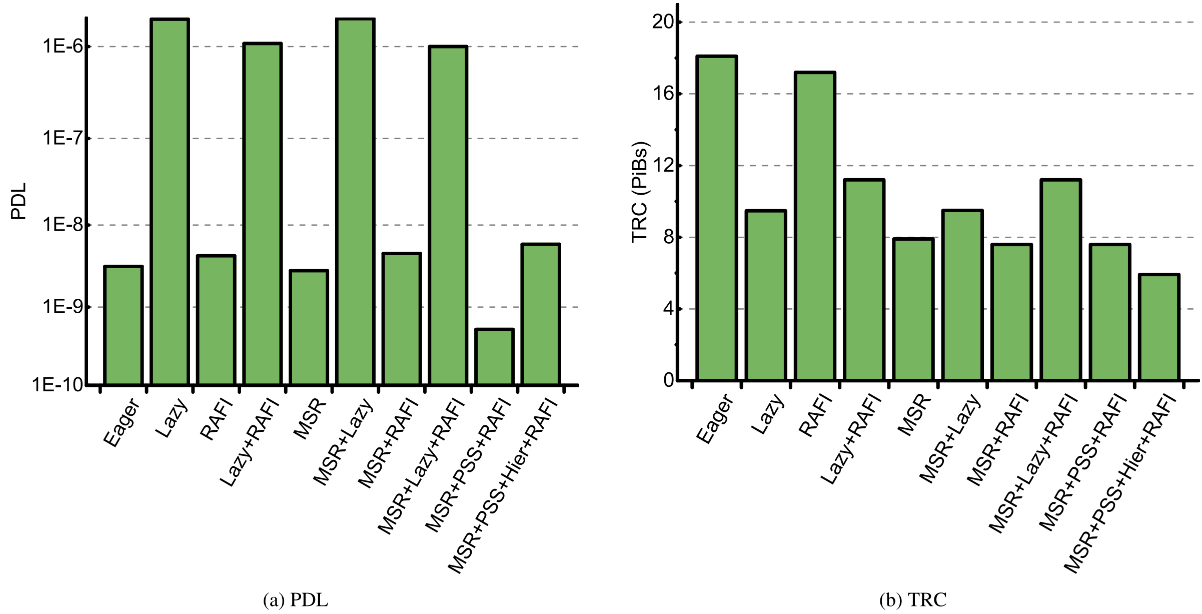


FIGURE 6. Reliability and repair cost for changing the data repair scheme.

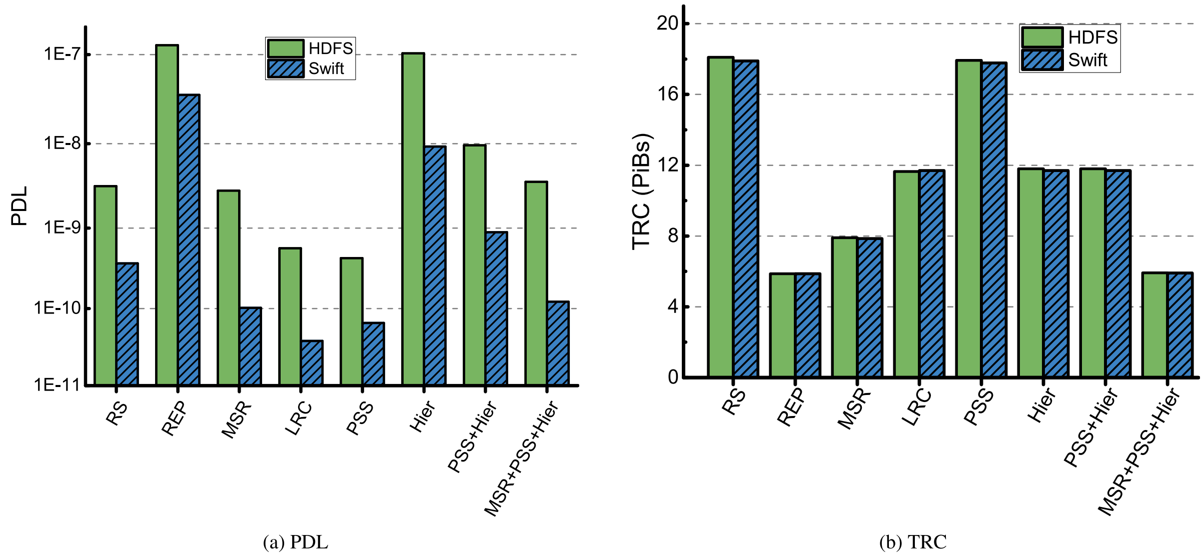


FIGURE 7. Reliability and cost in HDFS and Swift.

3) THE MINIMUM COST COMBINATION UNDER GIVEN RELIABILITY STANDARD

At last, we discuss the minimum cost combination under a given reliability standard, in both HDFS and Swift. In this paper, we adopt four data redundancy schemes, three data placement schemes on nodes, two data placement schemes on racks, and four data repair schemes as representatives. In total, there are 96 combinations in HDFS and 24 combinations in Swift. Our goal is to find the minimum cost combination under the given reliability standard in HDFS and Swift.

At first, we specify a fixed reliability standard as the reliability level which has to be achieved by the distributed storage system. Given a reliability standard solves the problem of the interaction between reliability and cost. We indicate the fixed reliability standard as 11 9's (i.e., $PDL \leq 1E-11$), which is the reliability level in many cloud service providers' service level agreements [4], [5].

The procedure is as follows. First, for each combination, we adjust its fault tolerance (by adding or removing the number of its parity chunks, i.e., increase or decrease $n-k$) to ensure it satisfies $PDL \leq 1E-11$ with as little TSC as possible. Second, we select some candidates from all combinations, these candidates have the minimum TSC or minimum TRC. Table 4 shows the candidates for the minimum cost combination. Third, we illustrate the TC of these candidates with the increasing of λ, Fig. 8 shows the results. The minimum cost combination under a given reliability standard can be found from the figure.

The candidates in Table 4 confirm some principles which can be proven by the simulation results from Section IV-B: (i) PSS has the highest reliability among all the three data

placement schemes on nodes, one combination can obtain the benefits of PSS and other schemes at the same time; (ii) the benefits of Lazy and MSR/LRC can not be obtained at the same time; (iii) both Hier, Lazy, and RAFI sacrifice reliability for repair cost reductions. Due to the first principle, every candidate contains PSS; due to the second principle, there are no combinations contain MSR/LRC+Lazy have been selected as candidates; due to the third principle, the choice of Hier, Lazy or RAFI causes the increase of *TSC* sometimes (to guarantee $PDL \leq 1E-11$). Except for the combinations in the table, the remaining combinations have higher *TSC* or higher *TRC* or both of them. For example, *TSC* of RS(11, 6)+PSS+Hier+Lazy+RAFI is the same with candidate MSR(11, 6, 10)+PSS+Hier+RAFI, but *TRC* of it is 10.56 PiB which is higher than the candidate.

TABLE 4. Candidates for the minimum cost combination under $PDL \leq 1E-11$.

Combination	TRC (PiB)	TSC (PiB·month)
HDFS		
REP($n=5$)+PSS+Hier	2.08	600
MSR(10, 6, 9)+PSS	7.41	200
MSR(11, 6, 10)+PSS+Hier+RAFI	4.32	220
Swift		
REP($n=4$)+PSS	7.79	480
REP($n=5$)+PSS+Hier	2.08	180
MSR(10, 6, 9)+PSS+Hier	5.43	200

From Fig. 8, we can find the minimum cost combination under the given reliability standard in both HDFS and Swift. In the figure, the Y-axis is $\frac{TC}{\alpha}$ which represents the value of TC. The combination which has the lowest TC is the eligible combination. In HDFS (see Fig. 8a), when $\lambda \leq 6.47$, MSR+PSS is the eligible combination; when $6.47 < \lambda \leq 169.64$, MSR+PSS+Hier+RAFI is the eligible combination; when $\lambda > 169.64$, REP+PSS+Hier is the eligible combination. In Swift (see Fig. 8a), when $\lambda \leq 119.4$, MSR+PSS+Hier is the eligible combination; when $\lambda > 119.4$, REP+PSS+Hier is the eligible combination. In total, with the increasing of λ, the combination which has a lower recovery penalty factor becomes more and more cost-efficient.

We also display three special values of λ, which come from three cloud services providers: Azure, Amazon, and Alibaba. All pricing models are collected in October, 2019. For Azure [20], $\lambda = 0.25 (\alpha = 41943)$; for Amazon [19], $\lambda = 0.4 (\alpha = 26214)$; for Ali [21], $\lambda = 2.62 (\alpha = 44040)$. In HDFS, MSR+PSS is the minimum cost combination under the pricing model of Azure, Amazon, and Ali. While in Swift, MSR+PSS+Hier is the minimum cost combination under the pricing model of Azure, Amazon, and Ali.

In general, **the minimum cost combination under the given reliability standard ($PDL \leq 1E-11$) is determined by the pricing model of cost. Under the pricing model of Amazon, Azure, and Alibaba cloud, the eligible combination is MSR+PSS for HDFS and MSR+PSS+Hier for Swift.**

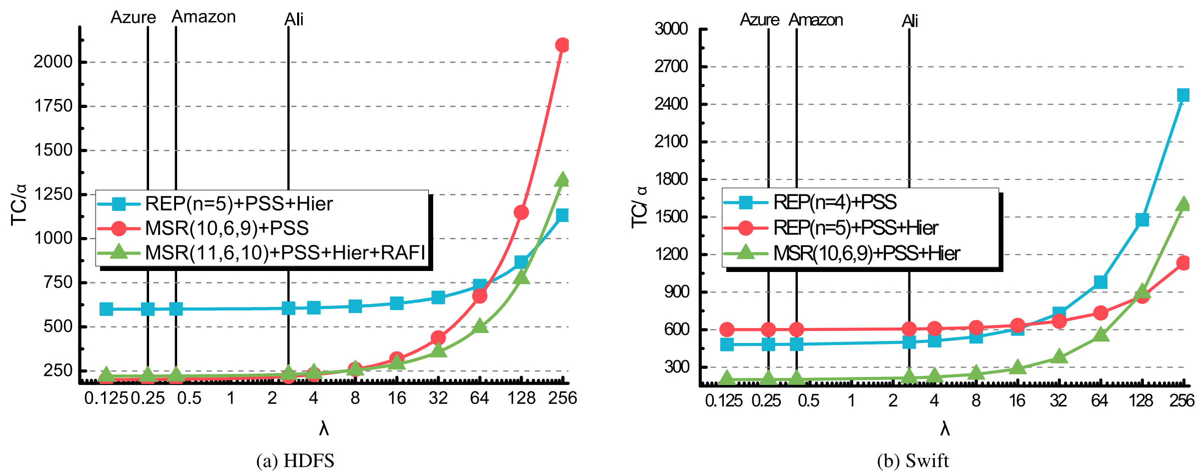
V. RELATED WORK

We summarize the related works on the reliability and cost of distributed storage systems.

In the early stage, the reliability and cost of storage systems have been measured with simple models based on permutation and combination. Weatherspoon and Kubiatowicz [34] show via modeling that erasure codes bring less repair and storage cost than replication under the same reliability. Rodrigues and Liskov [35] model the erasure codes and replication in DHTs. They conclude the benefits of erasure codes are less than replication in some cases. Lin *et al.* [46] model the availability and reliability in storage and communication systems. They also conclude replication performs better when node availability is lower or unknown. The latter two studies aim at peer-to-peer storage systems, real-time node joining and departure are taken into considerations, so they make different conclusions. However, these simple models cannot finish a comprehensive reliability study for the failures of different subsystems or combinations of different schemes.

Recently, the Markov model has been widely used for reliability measurements. The Markov model assumes both TTF and TTR follow the exponential distribution. Most studies about the reliability of data redundancy schemes have been accomplished with the Markov model. Huang *et al.* [8] use the Markov model to compare the reliability of LRC and RS codes. Sathiamoorthy *et al.* [27] show via modeling that the reliability benefits of XORBAS codes (another construction of LRC). In addition, a lot of constructions of MSR codes (including but not limited to FMSR [10], PMSR [47], and Butterfly codes [9]) use Markov model to display their reliability and cost benefits. Except the data redundancy schemes, some studies use the Markov model to cope with the reliability of data placement schemes. Venkatesan and Iliadis [48] discuss the reliability of clustered and declustered data placement. Furthermore, some studies discuss the schemes that come from two different categories. Hu *et al.* [17] analyze the reliability and repair cost of two data placement schemes on racks. They combine Hier and Flat with the same data redundancy scheme (RS or MSR), and then compare their effects. Arslan [49] uses the Markov model to study the reliability of disk arrays under different Maximum Distance Separable (MDS) erasure codes, different data allocations, and different repair rates. Node and rack failures are not included because the study is about the reliability of disk arrays but not distributed storage systems, and no discussions about the combination of different schemes. But, the correctness of the Markov model for reliability analysis is questionable from two aspects [50], [51]: (i) the disk failures fit Weibull distribution rather than Exponential distribution - failures accord with Exponential distribution is the fundamental assumption in Markov model; (ii) Markov model is memory-less - all the replaced and unreplaced disks are treated as the same.

The reliability simulator supports to generate failures with Weibull distribution and solves the memory-less



(a) HDFS (b) Swift

FIGURE 8. The minimum cost combination under $PDL \leq 1E-11$.

shortcomings of the Markov model, so it is more accurate and has been widely used in many studies. Green [52] implements the High Fidelity Reliability Simulator (HFRS) for reliability simulation on disk arrays. Zhang *et al.* [13] extend HFRS for data center environments and two data placement schemes on racks (Flat and Hier), the new simulator is named SIMEDC. SIMEDC considers the combination of the data redundancy scheme and data placement scheme on racks. Silberstein *et al.* [14] develop a new reliability simulator DS-SIM to display the effectiveness of Lazy in the distributed storage system. DS-SIM discusses the combination of Lazy and the data redundancy scheme. Fang *et al.* [15] develop a new reliability simulator to show the effectiveness of RAFI in the distributed storage system, the combination of RAFI and the data redundancy scheme has been studied. Hall [11] presents a simulator framework called CQSim-R, which evaluates the reliability of the distributed storage system, and studies the effects of data placement schemes on nodes. Epstein *et al.* [42] take the available network bandwidth into account to study the reliability of the distributed storage system, they combine simulation and combinatorial computations to measure the reliability.

Our work differs from previous simulators by specially considering the combination of the data redundancy scheme, the data placement scheme on nodes, the data placement scheme on racks, and the data repair scheme. In addition, we consider more complicated failure and repair patterns.

VI. CONCLUSION

In order to build a reliable and cost-efficient distributed storage system, we present a comprehensive simulation analysis to measure the reliability and cost of distributed storage systems. Our analysis covers data redundancy schemes, data placement schemes (on both nodes and racks), and data repair schemes. To achieve an overall analysis, we design and implement a comprehensive event-based reliability simulator CR-SIM. Using CR-SIM, we conduct various simulations under HDFS and Swift, which represent two different repair patterns. Through the simulation results, we find several important findings, which are useful to guide the development of reliable and cost-efficient storage systems. The source code of our CR-SIM is available at `https://github.com/yichuan0707/CR-SIM`.

REFERENCES

[1] S. Ghemawat, H. Gobioff, and S.-T. Leung, "The Google file system," in *Proc. ACM SIGOPS Oper. Syst. Rev.*, vol. 37, 2003, pp. 29–43.
[2] *HDFS*. Accessed: Mar. 2019. [Online]. Available: https://hadoop.apache.org/
[3] *Openstack Swift Object Storage System*. Accessed: Mar. 2019. [Online]. Available: http://docs.openstack.org/developer/swift/
[4] (Mar. 2019). *SLA for Azure Storage*. [Online]. Available: https://docs.microsoft.com/en-us/azure/storage/common/storage-redundancy
[5] *Amazon S3*. Accessed: Mar. 2019. [Online]. Available: https://www.amazonaws.cn/en/s3/
[6] R. Bhagwan, K. Tati, Y. Cheng, S. Savage, and G. M. Voelker, "Total recall: System support for automated availability management," in *Proc. USENIX Symp. Netw. Syst. Design Implement. (NSDI)*, vol. 4, 2004, p. 25.
[7] I. S. Reed and G. Solomon, "Polynomial codes over certain finite fields," *J. Soc. Ind. Appl. Math.*, vol. 8, no. 2, pp. 300–304, Jun. 1960.
[8] C. Huang, H. Simitci, Y. Xu, A. Ogus, B. Calder, P. Gopalan, J. Li, and S. Yekhanin, "Erasure coding in windows azure storage," in *Proc. USENIX Annu. Tech. Conf. (USENIX ATC)*, 2012, pp. 15–26.
[9] L. Pamies-Juarez, F. Blagojevic, R. Mateescu, C. Gyuot, E. E. Gad, and Z. Bandic, "Opening the chrysalis: On the real repair performance of MSR codes," in *Proc. 14th USENIX Conf. File Storage Technol. (FAST)*, 2016, pp. 81–94.
[10] Y. Hu, H. C. Chen, P. P. Lee, and Y. Tang, "NCCloud: Applying network coding for the storage repair in a cloud-of-clouds," in *Proc. FAST*, 2012, p. 21.
[11] R. J. Hall, "Tools for predicting the reliability of large-scale storage systems," *ACM Trans. Storage*, vol. 12, no. 4, pp. 1–30, Aug. 2016.
[12] A. Cidon, S. M. Rumble, R. Stutsman, S. Katti, J. Ousterhout, and M. Rosenblum, "Copysets: Reducing the frequency of data loss in cloud storage," in *Proc. USENIX Conf. Tech. Conf.*, 2013, pp. 37–48.
[13] M. Zhang, S. Han, and P. P. C. Lee, "A simulation analysis of reliability in erasure-coded data centers," in *Proc. IEEE 36th Symp. Reliable Distrib. Syst. (SRDS)*, Sep. 2017, pp. 144–153.
[14] M. Silberstein, L. Ganesh, Y. Wang, L. Alvizi, and M. Dahlin, "Lazy means smart: Reducing repair bandwidth costs in erasure-coded distributed storage," in *Proc. Int. Conf. Syst. Storage (SYSTOR)*, 2014, pp. 1–7.
[15] J. Fang, S. Wan, and X. He, "RAFI: Risk-aware failure identification to improve the RAS in erasure-coded data centers," in *Proc. USENIX Annu. Tech. Conf. (USENIXATC)*, 2018, pp. 495–506.
[16] D. Ford, F. Labelle, F. Popovici, M. Stokely, V.-A. Truong, L. Barroso, C. Grimes, and S. Quinlan, "Availability in globally distributed storage systems," in *Proc. OSDI*, 2010, pp. 61–74.
[17] Y. Hu, X. Li, M. Zhang, P. P. C. Lee, X. Zhang, P. Zhou, and D. Feng, "Optimal repair layering for erasure-coded data centers: From theory to practice," *ACM Trans. Storage*, vol. 13, no. 4, pp. 1–24, 2017.

[18] C. Huang, M. Chen, and J. Li, "Pyramid codes: Flexible schemes to trade space for access efficiency in reliable data storage systems," *ACM Trans. Storage*, vol. 9, no. 1, p. 3, 2013.
[19] (2019). *Amazon AWS*. [Online]. Available: https://aws.amazon.com/ec2/pricing/on-demand/
[20] (2019). *Microsoft Azure*. [Online]. Available: https://azure.microsoft.com/en-us/pricing/details/managed-disks/
[21] (2019). *Alibaba Cloud*. [Online]. Available: https://www.alibabacloud.com/product/ecs
[22] B. Schroeder and G. A. Gibson, "Understanding disk failure rates: What does an MTTF of 1,000,000 hours mean to you?" *Trans. Storage*, vol. 3, no. 3, p. 8, 2007.
[23] L. N. Bairavasundaram, G. R. Goodson, S. Pasupathy, and J. Schindler, "An analysis of latent sector errors in disk drives," *ACM SIGMETRICS Perform. Eval. Rev.*, vol. 35, no. 1, pp. 289–300, 2007.
[24] B. Schroeder and G. A. Gibson, "A large-scale study of failures in high-performance computing systems," *IEEE Trans. Dependable Secure Comput.*, vol. 7, no. 4, pp. 337–350, Oct. 2010.
[25] A. G. Dimakis, P. B. Godfrey, Y. Wu, M. J. Wainwright, and K. Ramchandran, "Network coding for distributed storage systems," *IEEE Trans. Inf. Theory*, vol. 56, no. 9, pp. 4539–4551, Sep. 2010.
[26] J. S. Plank, J. Luo, C. D. Schuman, L. Xu, and Z. Wilcox-O'Hearn, "A performance evaluation and examination of open-source erasure coding libraries for storage," in *Proc. Fast*, vol. 9, Feb. 2009, pp. 253–265.
[27] M. Sathiamoorthy, M. Asteris, D. Papailiopoulos, A. G. Dimakis, R. Vadali, S. Chen, and D. Borthakur, "XORing elephants: Novel erasure codes for big data," *Very Large Data Based Endowment*, vol. 6, no. 5, pp. 325–336, Mar. 2013.
[28] M. Blaum, J. Brady, J. Bruck, and J. Menon, "EVENODD: An efficient scheme for tolerating double disk failures in RAID architectures," *IEEE Trans. Comput.*, vol. 44, no. 2, pp. 192–202, 1995.
[29] P. F. Corbett, B. English, A. Goel, T. Grcanac, S. R. Kleiman, J. Leong, and S. Sankar, "Row-diagonal parity for double disk failure correction," in *Proc. 3rd USENIX Conf. File Storage Technol.*, 2004, pp. 1–14.
[30] Z. Huang, H. Jiang, and K. Zhou, "An improved decoding algorithm for generalized RDP codes," *IEEE Commun. Lett.*, vol. 20, no. 4, pp. 632–635, Apr. 2016.
[31] S. Pawar, N. Noorshams, S. El Rouayheb, and K. Ramchandran, "DRESS codes for the storage cloud: Simple randomized constructions," in *Proc. IEEE Int. Symp. Inf. Theory*, Jul. 2011, pp. 2338–2342.
[32] D. S. Papailiopoulos, J. Luo, A. G. Dimakis, C. Huang, and J. Li, "Simple regenerating codes: Network coding for cloud storage," in *Proc. IEEE INFOCOM*, Mar. 2012, pp. 2801–2805.
[33] Y. Tang, J. Yin, W. Lo, Y. Li, S. Deng, K. Dong, and C. Pu, "MICS: Mingling chained storage combining replication and erasure coding," in *Proc. IEEE 34th Symp. Reliable Distrib. Syst. (SRDS)*, Sep. 2015, pp. 192–201.
[34] H. Weatherspoon and J. D. Kubiatowicz, "Erasure coding vs. replication: A quantitative comparison," in *Proc. Int. Workshop Peer-Peer Syst.* Berlin, Germany: Springer, 2002, pp. 328–337.
[35] R. Rodrigues and B. Liskov, "High availability in DHTs: Erasure coding vs. replication," in *Peer-to-Peer Systems IV*. Berlin, Germany: Springer, 2005, pp. 226–239.
[36] M. Ovsiannikov, S. Rus, D. Reeves, P. Sutter, S. Rao, and J. Kelly, "The quantcast file system," *Proc. VLDB Endowment*, vol. 6, no. 11, pp. 1092–1101, Aug. 2013.
[37] D. Ongaro, S. M. Rumble, R. Stutsman, J. Ousterhout, and M. Rosenblum, "Fast crash recovery in RAMCloud," in *Proc. 23rd ACM Symp. Oper. Syst. Princ. (SOSP)*, 2011, pp. 29–41.
[38] D. Borthakur, S. Rash, R. Schmidt, A. Aiyer, J. Gray, J. S. Sarma, K. Muthukkaruppan, N. Spiegelberg, H. Kuang, K. Ranganathan, D. Molkov, and A. Menon, "Apache Hadoop Goes realtime at facebook," in *Proc. Int. Conf. Manage. Data (SIGMOD)*, 2011, pp. 1071–1080.
[39] S. Muralidhar, W. Lloyd, S. Roy, C. Hill, E. Lin, W. Liu, S. Pan, S. Shankar, V. Sivakumar, and L. Tang, "F4: Facebook's warm blob storage system," in *Proc. 11th USENIX Symp. Oper. Syst. Design Implement. (OSDI)*, 2014, pp. 383–398.
[40] K. Rashmi, N. B. Shah, D. Gu, H. Kuang, D. Borthakur, and K. Ramchandran, "A hitchhiker's guide to fast and efficient data reconstruction in erasure-coded data centers," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 4, pp. 331–342, 2014.
[41] S. Jiekak, A.-M. Kermarrec, N. Le Scouarnec, G. Straub, and A. Van Kempen, "Regenerating codes: A system perspective," *ACM SIGOPS Oper. Syst. Rev.*, vol. 47, no. 2, pp. 23–32, 2013.
[42] A. Epstein, E. K. Kolodner, and D. Sotnikov, "Network aware reliability analysis for distributed storage systems," in *Proc. IEEE 35th Symp. Reliable Distrib. Syst. (SRDS)*, Sep. 2016, pp. 249–258.
[43] J. Elerath and M. Pecht, "A highly accurate method for assessing reliability of redundant arrays of inexpensive disks (RAID)," *IEEE Trans. Comput.*, vol. 58, no. 3, pp. 289–299, Mar. 2009.
[44] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The Hadoop distributed file system," in *Proc. IEEE 26th Symp. Mass Storage Syst. Technol.*, May 2010, pp. 1–10.
[45] A.-M. Kermarrec, E. L. Merrer, G. Straub, and A. V. Kempen, "Availability-based methods for distributed storage systems," in *Proc. IEEE 31st Symp. Reliable Distrib. Syst.*, Oct. 2012, pp. 151–160.
[46] W. K. Lin, D. M. Chiu, and Y. B. Lee, "Erasure code replication revisited," in *Proc. 4th Int. Conf. Peer-Peer Comput.*, 2004, pp. 90–97.
[47] K. Rashmi, P. Nakkiran, J. Wang, N. B. Shah, and K. Ramchandran, "Having your cake and eating it too: Jointly optimal erasure codes for i/o, storage, and network-bandwidth," in *Proc. 13th USENIX Conf. File Storage Technol. (FAST)*, 2015, pp. 81–94.
[48] V. Venkatesan and I. Iliadis, "A general reliability model for data storage systems," in *Quantitative Evaluation of Systems*. Los Alamitos, CA, USA: IEEE Computer Society, 2012, pp. 209–219.
[49] S. S. Arslan, "A reliability model for dependent and distributed MDS disk array units," *IEEE Trans. Rel.*, vol. 68, no. 1, pp. 133–148, Mar. 2019.
[50] K. M. Greenan, J. S. Plank, and J. J. Wylie, "Mean time to meaningless: MTTDL, Markov models, and storage system reliability," in *Proc. HotStorage*, 2010, pp. 1–5.
[51] P. Karmakar and K. Gopinath, "Are Markov models effective for storage reliability modelling?" 2015, *arXiv:1503.07931*. [Online]. Available: http://arxiv.org/abs/1503.07931
[52] K. M. Greenan, "Reliability and power-efficiency in erasure-coded storage systems," M.S. thesis, Storage Syst. Res. Center, Baskin School Eng., Univ. California, Santa Cruz, Santa Cruz, CA, USA, 2009.

YICHUAN QI received the B.S. degree in computer science from Sichuan University, Chengdu, China, in 2010. He is currently pursuing the Ph.D. degree with the School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan, China. His work has published in conference proceedings, such as ICPADS. His research interests include distributed storage systems, reliability and availability, and cloud storage services.

DAN FENG (Senior Member, IEEE) received the B.E., M.E., and Ph.D. degrees in computer science and technology from the Huazhong University of Science and Technology (HUST), Wuhan, China, in 1991, 1994, and 1997, respectively. She is currently a Professor and the Dean of the School of Computer Science and Technology, HUST. She has over 100 publications in major journals and international conferences, including the IEEE TC, the IEEE TPDS, ACM-TOS, FAST, USENIX ATC, EuroSys, ICDCS, HPDC, SC, ICS, IPDPS, DAC, and DATE. Her research interests include computer architecture, non-volatile memory technology, distributed and parallel file systems, and massive storage systems. She is a member of the Association for Computing Machinery and the Chair of the Information Storage Technology Committee, Chinese Computer Academy. She served on the program committees of multiple international conferences, including SC, in 2011 and 2013, and MSST, in 2012 and 2015.

BINBING HOU received the B.S. degree from the Wuhan University of Technology, Wuhan, China, in 2011, the M.S. degree from the Huazhong University of Science and Technology, Wuhan, in 2014, and the Ph.D. degree in computer science from Louisiana State University, Baton Rouge, LA, USA, in 2019. He is currently a Software Engineer with LinkedIn. His work has published in major journals, such as *ACM Transactions on Storage (TOS)* and conference proceedings, such as ICPP, MSST, and MASCOTS. His research interests include distributed storage systems, blockchain systems, cloud services, and non-volatile memory.