Unequal Failure Protection Coding Technique for Distributed Cloud Storage Systems

Yupeng Hu, *Senior Member, IEEE*, Yonghe Liu, *Member, IEEE*, Wenjia Li, Keqin Li, *Fellow, IEEE*, Kenli Li, *Senior Member, IEEE*, Nong Xiao, *Member, IEEE*, and Zheng Qin, *Member, IEEE*

Abstract—In recent years, erasure codes have become the de facto standard for data protection in large scale distributed cloud storage systems at the cost of an affordable storage overhead. However, traditional erasure coding schemes, such as Reed-Solomon codes, suffer from high reconstruction cost and I/Os. The recent past has seen a plethora of efforts to optimize the tradeoff between the reconstruction cost, I/Os and storage overhead. Quiet different from all prior studies, in this paper, our erasure coding technique makes the first attempt to take advantage of the unequal failure rates across the disks/nodes to optimize the system reliability and reconstruction performance. Specifically, our proposed technique, the Unequal Failure Protection based Local Reconstruction Code (UFP-LRC) divides the data blocks into several unequal-sized groups with local parities, assigning the data blocks stored on more failure-prone disks/nodes into the smaller-sized group, so as to provide unequal failure protection for each group. In this way, by exploiting the nonuniform local parity degrees, the proposed UFP-LRC enables the data blocks that are stored on more failure-prone disks/nodes to tolerate a greater number of failures while suffering from less repair cost than others, leading to a substantial improvement of the overall reliability and repair performance for cloud storage systems. We perform numerical analysis and build a prototype storage system to verify our approach. The analytical results show that the UFP-LRC technique gradually outperforms LRC along the increase of failure rate ratio. Also, extensive experiments show that, when compared to LRC, UFP-LRC is able to achieve a 10 to 15 percent improvement in throughput, and an 8 to 12 percent reduction in decoding latency, while retaining a comparable overall reliability.

Index Terms—Cloud storage system, erasure codes, unequal failure protection, reconstruction

1 Introduction

1.1 Motivation

Erasure codes have become the *de facto* standard for data protection in large scale distributed cloud storage systems. In view of potential disk/nodes failures in Petabyte or Exabyte data centers operating 7×24 [1], [2], [3], [4], [5], data redundancy is indispensable for ensuring reliability and availability. Conventional triple replication can quickly become too expensive when facing exponential growth of data [6], [7], [8] in data centers. In contrast, erasure codes can achieve higher levels of reliability as well as much lower storage overhead [9], [10]. An increasing number of cloud storage systems such as Windows Azure [11], Amazon S3 [12], Google GFSII [13], HDFS and f4 in Facebook [14], [15], DiskReduce [16] and EMC ATMOS [17] are adopting erasure codes to achieve high data availability with significant reduced cost.

In this regard, Reed-Solomon (RS) codes become a popular choice recently owing to their maximum distance separable (MDS) property. For example, RS(6, 3) is used in Google GFSII [13], RS(10, 4) in HDFS-RAID in Facebook [18], RS(10, 6) in DiskReduce [16], and RS(9, 3) in EMC ATMOS [17], respectively. However, stemming from the traditional communication systems, the Reed-Solomon codes suffer from tremendous network bandwidth and I/O cost in the following reconstruction manner.

A reconstruction process for disks/nodes failure, generally launched as a background job, can take a few hours to days in a large scale cloud storage system [19]. Read requests for currently unavailable data may also trigger reconstruction operations on-the-fly. These result in *degraded* performance in that more time is needed to recover the lost data. Motivated thereby, a plethora of efforts have been devoted recently to optimize the reconstruction performance both in theory and practice [19], [20].

Network-coding based regenerating codes are theoretically shown to be near optimal erasure codes [5], [21], [22]. Minimum-storage-regenerating (MSR) codes are optimal with regard to storage overhead and network cost. MSR codes achieve identical redundancy-reliability tradeoff to RS codes, while significantly lowering network load as it

- *Y. Hu, K. Li, and Z. Qin are with the College of Computer Science and Electronic Engineering, Hunan University, Changsha 410082, P.R. China. E-mail: {yphu, lkl, zqin}@hnu.edu.cn.*
- *Y. Liu is with the Department of Computer Science and Engineering, University of Texas at Arlington, Arlington, TX 76019 USA. E-mail: yonghe@cse.uta.edu.*
- *W. Li is with the Department of Computer Science, New York Institute of Technology, New York, NY 10023 USA. E-mail: wli20@nyit.edu.*
- *K. Li is with the Department of Computer Science, State University of New York, New Paltz, NY 12561 USA. E-mail: lik@newpaltz.edu.*
- *N. Xiao is with the School of Data and Computer Science, Sun Yet-Sen University, Guangzhou 510006, P.R. China. E-mail: xiao-n@vip.sina.com.*

Manuscript received 28 Mar. 2017; revised 28 Aug. 2017; accepted 15 Dec. 2017. Date of publication 19 Dec. 2017; date of current version 5 Mar. 2021. (Corresponding author: Yupeng Hu.)
Recommended for acceptance by J. Wang.
Digital Object Identifier no. 10.1109/TCC.2017.2785396

can recover an unavailable block by merely downloading a small fraction of the data from any d blocks. Unfortunately, in addition to the difficulty of overcoming coding and exact recovery challenges [23], [24], [25], MSR codes also face considerable additional I/Os overhead induced by reading all d available blocks in order to compute a linear combination for reconstruction. Recently proposed product-matrix-MSR codes (PM-MSR) based PM-RBT code [20], [26], can achieve a joint optimality over I/O-storage and network cost while suffering from a high storage redundancy.

A new class of practical erasure codes named local reconstruction codes (LRC) has been widely employed in major cloud storage systems [11], [14], [27]. LRC introduces local parities along with a few global parities in order to reduce the minimal number of blocks needed to reconstruct a single block failure [11]. Indeed, single block failures dominate data unavailability (*often more than* 99 percent) in real systems [3], [28] that fits LRC designing goal. Other prevailing distributed storage systems such as ceph [27] and HDFS-Xorbas of Facebook [14] also exploit similar local reconstruction strategies.

Nonetheless, existing erasure codes offer identical level of failure protection for all blocks by trading off the storage and reconstruction cost. However, a number of observations show that not all disks/nodes are of equal failure rates in reality. For instance, the failure rates of disks may be correlated with the wear (e.g., for solid state disks) and usage during different periods of their lifetimes [1]. As a result, data loss and subsequent reconstructions are mainly triggered by those failure-prone diks/nodes. Therefore, it is desirable, from a system performance point of view, to have an erasure code where blocks stored on more failure-prone diks/nodes that are better protected.

1.2 Our Contributions

Motivated thereby, in this paper, we present an Unequal Failure Protection based Local Reconstruction Code (a.k.a., UFP-LRC) for distributed cloud storage systems. Different from existing designs, UFP-LRC, by protecting blocks with different redundancy based on their likelihood of failure, optimizes systematically the reliability and reconstruction performance of the storage system.

First, UFP-LRC divides data blocks into unequal-sized groups, the smaller-sized of which consists of data blocks that will be stored on more failure-prone disks/nodes. *For simplicity, in this paper, we term those blocks (either data or parity blocks) more failure-prone blocks*. Specifically, by computing one local parity in each group and the global parities for all data blocks, UFP-LRC enables more failure-prone blocks within the smaller group to have a higher protection level and thus can tolerant more failures than others. Since data unavailability mainly results from more failure-prone blocks, under UFP-LRC, the enhancement of their erasure tolerance will lead to an improvement in overall data availability. Second, as there are unequal local parity degrees in different groups, the parities with lower degrees can facilitate more efficient reconstruction. In other words, the more failure-prone the blocks are, the parity with lower degree they share, and thus the less reconstruction cost they entail. Intuitively, the reconstructions for the more failure-prone blocks will outweigh all other contributions to the overall reconstruction cost under certain conditions (e.g., when the failure rates of these more failure-prone blocks exceed those of other blocks to a certain extent). Therefore, under UFP-LRC, the savings of the reconstruction cost for the more failure-prone blocks substantially achieve a reduction in the overall reconstruction cost.

Our main contributions are summarized as follows:

(1) To the authors' best knowledge, the UFP-LRC is the first work to provide unequal failure protection for disks/nodes of distributed cloud storage systems. The UFP-LRC improves the overall reliability as well as the reconstruction efficiency.
(2) We provide a generalized construction of the generator matrix for the UFP-LRC. The generalized generator matrix facilitates the implementation of UFP-LRC by searching the coefficients over a small finite field.
(3) We implement the UFP-LRC over a Hadoop based prototype cloud storage system. Evaluations on our prototype storage system validate that, compared to LRC, UFP-LRC can achieve improvement in throughput and reduction in decoding latency, while retaining a comparable overall reliability.

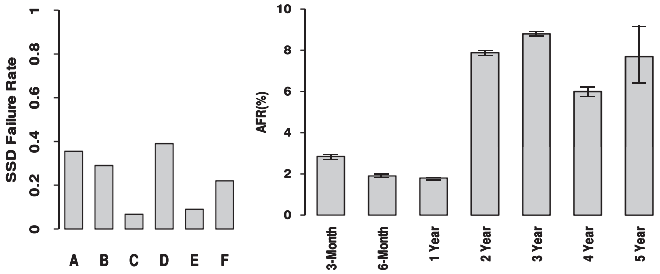
The rest of this paper is organized as follows. Sections 2 and 3 present the related work and the motivation. The key designs and definition of UFP-LRC are introduced in Section 4. The code selection and efficiency analysis are presented in Sections 5 and 6. The Hadoop based prototype system are demonstrated in Section 7. We also conduct extensive experiments to evaluate our approach in Section 8. Section 9 concludes this paper.

2 Related Work

In this section, we review some of the related works that are the starting point of our research.

Under the popular RS(k, r) codes, a file of size D bytes is divided into k equal-sized data blocks, each of size D/k bytes. These data blocks are then encoded into a set (also called *stripe*) of k data blocks and r redundant parity blocks. Any k blocks out the $(k + r)$ blocks suffice to reconstruct the entire original file, i.e., up to r arbitrary block failures can be tolerated. The data and parity blocks will be placed on selected disks/nodes of different fault domains or racks in the cloud storage system according to certain deployment rules. However, they must always read and download all k blocks from any k surviving disks for every recovery even though there is only one block failure.

Motivated by the performance bottleneck in reconstruction of the traditional RS codes, recent erasure codes for distributed storage system focus on the construction of coding schemes providing improved tradeoff between storage overhead and reconstruction cost. In [29], the authors presented pyramid codes which are featured with obviously low repair-cost yet slightly high storage-overhead compared with RS codes. But the pyramid codes rely on a exponentially complex search algorithm to discover coding equation coefficients. To optimize the way pyramid codes used to construct coding equations and then make implementation practical, the authors introduced the LRC [11] which searches the coefficients over a small finite field.



(a) SSDs failure rates [1] (b) AFR observed by Google [35]

Fig. 1. Unequal failure rates of disks. (a) Average SSDs failure rates for various platforms. (b) Annualized hard disks failure rates broken down by age groups.

Meanwhile LRC exploits local parities with lower degrees than the traditional global parities, thus it can reduce the minimal number of data blocks that need to be read from during reconstruction operations. Recently, the LRC has been shipped in Windows Server 2012 R2 and Windows 8.1 [30] owing to their applicability. *As a smart improvement of LRC, however, the UFP-LRC takes advantage of nonuniform local parity degrees to address the unequal failure protection issue while further optimize the data availability and reconstruction performance.* To our knowledge, this is the first work to address the problem of the unequal failure protection in the literature.

The local repair strategies also have been studied in [14], [27], [31]. The HDFS-Xorbas [14] adopted a locally repairable coding strategy based on RS codes and a construction algorithm. The local parities and global RS parities satisfy an alignment equation, so that any single block failure (data or parity block) will have the equal reconstruction cost. In addition to the exponential complexity of the deterministic construction algorithm, another disadvantage of this strategy is the extra storage requirement. Both ceph and redhat [27] provide system-level local repair solutions to reduce network and I/O consumption for recovery.

On the other hand, several works (e.g., [31], [32]) have carried out in-depth theoretical analysis of the locality of codes. In [31], the authors proved the existence of the optimal locally repairable codes, and presented an explicit locally repairable codes. In [32], the authors established a tight bound for the redundancy in terms of the code length, the distance and the locality.

Among available coding schemes, the most relevant but not identical one to UFP-LRC is the unequal error/erasure protection (UEP) code which has been widely used in multi-media and communication systems [33]. Since not all messages are of equal importance for the user, UEP codes protect some of the encoded message symbols against more errors/erasures than others, resulting in their higher recovery probability and earlier recovery. *However, UEP codes are fine-grained bit-level, packet-level or layer-level solutions, which are unaffordable for the data-intensive yet I/O constrained distributed cloud storage systems.* For example, as a layer-level coding technique, UEP-LRC [33] first has to extract the enhancement layer and the more important base layer from the video, so as to provide unequal protection for them by using two different LRC codes. On the other hand, the UFP-LRC also can be adapted to the importance or prioritization based erasure protection for distributed storage systems.

3 Motivation and Design Goals

In this section, we motivate our work by presenting observations in real system of unequal failure rates across the population of disks.

3.1 Unequal Failure Rates in Storage Systems

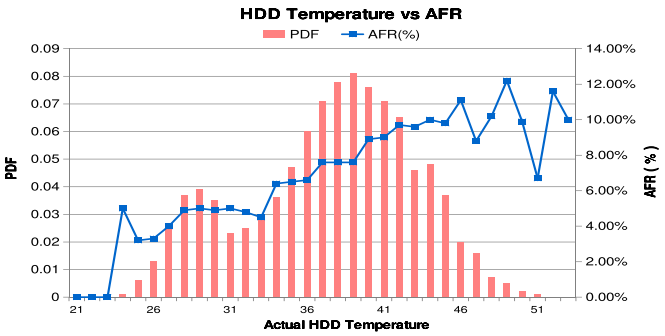
In cloud storage systems, data blocks may be unavailable due to disk/node failures. A disk/node is considered to have failed if it is replaced as part of a repair procedure. This definition for failure implicitly excludes disks/nodes that are replaced due to an upgrade. Although the causes of a disk/node failure can vary significantly (e.g., upgrade replacement, response timeout, and uncorrectable data errors), the statistical failure rates have been well studied from the standpoint of manufactures or cloud end-users [1], [3], [28], [34]. Failure rate λ is the frequency at which a disk/node fails, expressed in failures per unit of time. The relationship between failure rate and the more often reported mean-time-to-failure (MTTF) is $\lambda = 1 - e^{\frac{-1}{MTTF}}$.

The well-known annualized failure rate (AFR) gives the estimated probability that a disk or node will fail during a full year of use. AFR can be approximated as $\lambda \approx \frac{8760 \times 100}{MTTF}$ (expressed in %), assuming a very small AFR. For example, MTTF, as specified in manufacturers' datasheets, is about 1,000,000 hours, suggesting a nominal annual failure rate of at most 0.87 percent [28]. However, most of the available information about MTTF are usually based on manufacturers' own extrapolation from accelerated life test data of small populations [34].

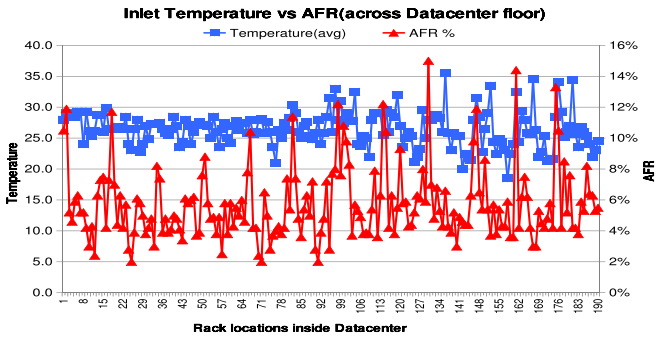
Recent studies [1], [34] based on replacement records and logs of a large population of disks have shed some light on the practical failure rate for cloud storage systems. Fig. 1 illustrates the baseline failure rates for a large population of solid state disks (SSDs) or hard disks in distributed storage systems. Evidently, there is a nonuniform distribution of failure rates across the disks due to various factors (e.g., wear or inherent raw bit error rate). The highest failure rate is about 5-6 times more than the lowest. For instance, in [34], based on a large field sample of disks, the actual AFRs for individual disks range from 1.7 percent for first-year disks to over 8.6 percent for three-year-old disks. Even for the same group (see the baseline in (b)), the failure rates vary within a certain range.

Particularly, it is observed by Microsoft [2] that disks/nodes temperatures significantly impact AFR as shown in Fig. 2. As we can see clearly in Fig. 2a, the AFR rate increases along with disk temperature. Furthermore, Fig. 2b shows that there is a significant correlation between the inlet temperatures observed at a server/node location within a rack and the failure rate of that server/node. The higher the average inlet temperature, the higher the failure rate. More evidence can be found at the online public failure trace archive [35], which will be omitted here due to space limit.

Unfortunately, existing studies have overlooked this fact and usually provide identical desired level of failure/erasure protection for all disks. Motivated by the above observations, we make the first attempt in this paper to improve the reconstruction performance and reliability in cloud storage system by providing unequal protections. Notwithstanding it is difficult to accurately predict failures/failure rates, there are useful indicators of disk health as discussed



(a) Unequal failure rates at different disk temperatures.



(b) Unequal failure rates at different server inlet temperatures.

Fig. 2. Unequal AFR of disks/servers at different temperatures observed by Microsoft [2].

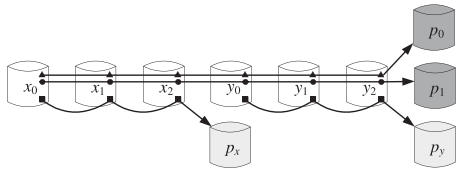
above, including temperature, usage/activity levels and the disk's self monitoring parameters (e.g., scan errors and reallocation counts) [1], [34]. These real-time indicators in cloud storage systems provide the facilitators in the implementation of the proposed coding scheme.

3.2 Design Goals

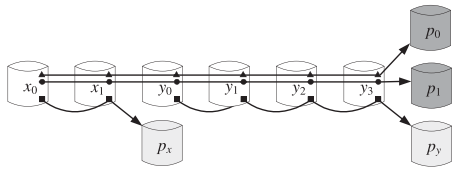
In view of the unequal failure rates as well as the reconstruction challenge, the fundamental design goals of UFP-LRC can be outlined below:

(1) The blocks that are more failure-prone will be protected against more failures/erasures. Note that, we use the terms erasure and failure synonymously: a block is erased if the disk/node on which it is stored fails. Let K denote the set of all disks/nodes storing the data and parity blocks of an encoded stripe. The disks/nodes in H are more failure-prone, and $H \subset K$. If the data blocks on other disks/nodes, i.e., in $K \setminus H$, are able to tolerate f data block failures, then the data blocks on disks/nodes of subset H can tolerate more failure patterns, for example, arbitrary $\tilde{f}$ *data block failures*, where $\tilde{f} \geq f$.

(2) The reconstruction cost incurred by a single failure of data blocks on disks/nodes that are more failure-prone should be as low as possible. Assume that the loss of a data block on a disk/node in H and $K \setminus H$ will result in a reconstruction cost of $\tilde{c}$ and c, we have $\tilde{c} \leq c$. Since the reconstruction operations triggered by the failures of data blocks on disks/nodes in H account for most of the repair cost, reducing repair cost for these data blocks will significantly relieve the performance bottleneck in reconstruction.



(a) LRC(6, 2, 2) [11]



(b) UFP-LRC(2+4, 2, 2) (this paper)

Fig. 3. A comparative example of LRC and UFP-LRC. (a) A *LRC(6, 2, 2)* scheme. 6 data fragments, 2 local parities and 2 global parities. (b) A *UFP-LRC(2+4, 2, 2)* scheme. 2 data blocks in first group x, 4 data blocks in group y, 2 local parity blocks (p_x, p_y) and 2 global parity blocks (p_0, p_1).

4 Unequal Failure Protection Based Local Reconstruction Code

In this section, we use a comparative example to introduce the UFP-LRC strategy, followed by the formal definition and construction of UFP-LRC. Note that, we employ shorter codes for illustration here that are different from real implementations.

4.1 A Comparative Example Study

The comparative example shown in Fig. 3 illustrates the coding strategies of LRC [11] and UFP-LRC.

In local repair, LRC(k, l, r) divides the k data blocks into l equal size group ($\frac{k}{l}$ data blocks in each group), and computes one local parity for each group as well as r global parities. In Fig. 3a, LRC(6, 2, 2) divides the 6 data blocks into two equal size groups, and computes the local parity block p_x with x_0, x_1 and x_2, and p_y with y_0, y_1 and y_2 respectively. It also computes the two global parities (p_0 and p_1) with all data blocks. If x_0 is lost, instead of reading one of the global parity (p_0 or p_1) and the other 5 surviving data blocks like RS(6, 4), the LRC can perform a more efficient reconstruction of x_0 by just reading p_x and two data blocks (x_1 and x_2). However, the local parities compromise the failures tolerance capability. Unlike RS(6, 4), LRC(6, 2, 2) is not MDS code [11] and hence cannot tolerate *arbitrary* 4 failures (at most any $(r+1) = 3$ failures). For instance, simultaneous failures of x_0, x_1, x_2 and p_x is non-decodable since there are only two global parities that can help to decode the 3 missing data blocks. *It is impossible to decode 3 lost data blocks from merely 2 parities, regardless of the coding equations*. Thus LRC is maximally recoverable (MR) code [29], meaning it can decode any failure pattern that is information-theoretically decodable.

UFP-LRC is also maximally recoverable, while achieving desired availability as shown in Sections 5 and 6. Unlike LRC, UFP-LRC divides the data blocks into several unequal size groups, so as to let the most failure-prone data blocks and their local parity form a smallest size group. As shown in Fig. 3b, UFP-LRC(2+4, 2, 2) puts the more failure-prone x_0, x_1, and their local parity p_x into a small size group x, while (y_0, y_1, y_2, y_3, and p_y) into a large size group y. Clearly,

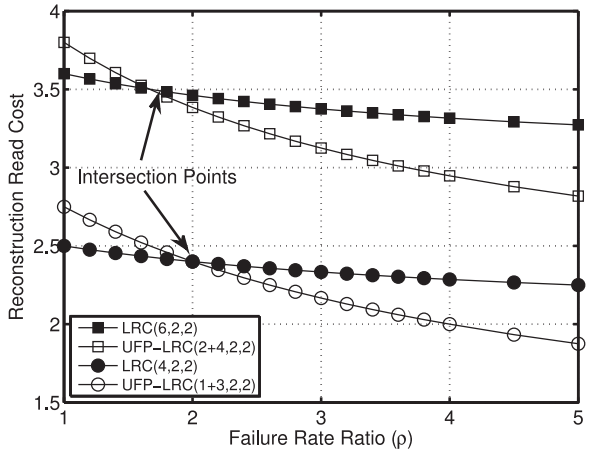


Fig. 4. Comparisons of varying reconstruction cost.

the key idea of UFP-LRC is nonuniform local parity degrees, achieving our two design goals outlined below.

First, UFP-LRC can provide unequal failure protection for different data blocks. In comparison with LRC, in UFP-LRC, group x can tolerate the simultaneous failure of the same 3 data blocks and one local parity, i.e., x_0, x_1, y_0 (i.e., the identical x_2 in Fig. 3a) and p_x. Thanks to the involvement of the local parity p_y associated with y_0, it is possible to decode the 3 data blocks (x_0, x_1, and y_0) via 3 parities (p_y, p_0, and p_1). Actually, as the following Theorem 1 shows, the data blocks in group x can not only tolerate simultaneous failure of any $(r+1)=3$ data blocks and one parity block but also any up to $(r+2)=4$ data blocks. By enumerating all undecodable 4-failure cases, we can find that the undecodability ratio of the first group under UFP-LRC and LRC is 1.4 percent (3 cases) and 3.8 percent (8 cases), respectively. Therefore, for UFP-LRC, the blocks in group x are more reliable than the blocks in group y as well as the same blocks under LRC.

Second, UFP-LRC can reduce the overall reconstruction cost.[1] Compared with LRC(6, 2, 2), UFP-LRC(2+4, 2, 2) further reduces the minimal reconstruction cost from 3 to 2. It suffices to reconstruct the failure data block x_0 merely by reading x_1 and p_x. It is easy to see that UFP-LRC(2+4, 2, 2) achieves this at the cost of enlarging the size of group y to 4, resulting in a higher reconstruction cost for the single failure within y than LRC(6, 2, 2). However, UFP-LRC is committed to optimizing the reconstructions for the most failure-prone data blocks, which account for most of the reconstruction cost. Consequently, the repair cost savings of x_0 and x_1 can lead to a reduction in the overall reconstruction cost under general conditions.

Let ρ be the failure rate ratio, which is defined as the ratio of the failure rate of group x (including the p_x) to the failure rate of the other blocks. Let R be the average reconstruction read cost of single failure. For UFP-LRC(2+4, 2, 2), it needs to take 2 and 4 blocks to repair any of the 3 and 5 blocks (including one local parity block) within group x and y, respectively. Moreover, it takes 6 blocks to repair any of the 2 global parities, and the total number of blocks to be repaired is $(3\times\rho+7)$. Thus, we have $R=\frac{3\times2\times\rho+5\times4+2\times6}{3\times\rho+7}$. Likewise, for LRC(6, 2, 2), set the same number of blocks be more failure-prone, hence $R=\frac{3\times3\times\rho+5\times3+2\times6}{3\times\rho+7}$. As Fig. 4 shows, the reconstruction read cost of UFP-LRC declines more rapidly and becomes lower than LRC at the intersection points (roughly ρ =1.8 and 2, respectively).

1. Note that reconstruction cost in this paper refers to the number of blocks that need to be read/downloaded for recovery operations

4.2 Formal Description

In this section, we formally describe UFP-LRC and present its properties with arbitrary coding parameters.

Definition 1. *A UFP-LRC(k_0, l, r) consists of $\sum_{i=0}^{l-1}(k_0+i)$ data blocks within l unequal-sized groups, l local parities (one for each group), and r shared global parities. There are k_0 data blocks within the 1st smallest group, (k_0+1) within the 2nd group, (k_0+2) within the 3rd group, ..., and (k_0+l-1) within the lth group.*

For UFP-LRC(k_0, l, r), we put the k_0 most failure-prone data blocks into the first group, while the most $l-1$ reliable data blocks into the lth group. The total number of blocks is $n=l+r+\sum_{i=0}^{l-1}k_i$, where k_i is the size of group i, $k_i=k_0+i$. Therefore, the normalized storage overhead is $\frac{n}{\sum_{i=0}^{l-1}k_i}=1+\frac{2(l+r)}{l(2k_0+l-1)}$. For 1.55x.

Definition 2. *ρ denotes the failure rate ratio of the more failure-prone blocks (including the parities) to the other blocks.*

The ρ is the key parameter used for evaluating the performance of UFP-LRC in our analysis and comparisons.

Theorem 1. *The UFP-LRC(k_0, l, r) has the following properties:*

(1) The minimal reconstruction cost for decoding single failure is k_0, and arbitrary $(r+1)$ blocks (data or parity blocks) failures is decodable. (2) The group of size $\hat{r}$ $(\leq r)$ can tolerate arbitrary $(r+1)$ data blocks and one parity block (local or global parity) failures. In other words, these blocks (including the local parity blocks) within the most failure-prone $(r-k_0+1)$ groups are always available when any up to $(r+1)$ data blocks and one parity block erasures occur. (3) The group of size $\hat{r}$ $(\leq r)$ can tolerate arbitrary $(r+2)$ data blocks (not including parity blocks) failures.

Proof. For property (1), first, it needs to take k_0 blocks to repair any single failure within the 1st smallest group, thus the minimal reconstruction cost for decoding single failure is k_0. Then let's consider two worst cases of arbitrary $(r+1)$ block failures: if there are $(r+1)$ data block failures, there are at least one local parity and r global parities that are available for decoding the $(r+1)$ failures; if there are just r data blocks and one parity block failures occurring within only one group, then there are at least r global parity blocks available for decoding. For property (2), let's consider the worst case: all data blocks and the corresponding local parity block within a group of size r are lost. Then at least one data block failure occurs within the other group, which entails at least one local parity block. In this way, there are r global parity blocks and one local parity block available for decoding the $(r+1)$ data blocks successfully. For property (3), similarly, let's consider the worst case: all data blocks within a group of size r are lost, which will entail one local parity for decoding. Furthermore, the last 2 data block failures will occur within the other groups, which entail at least one more local parity. Thus there are r global parities and at least 2

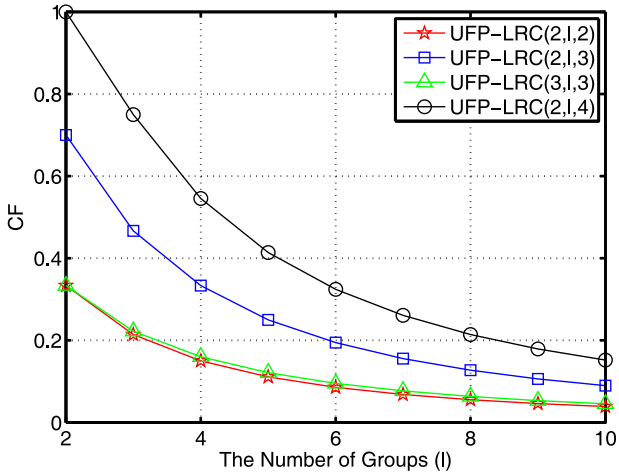


Fig. 5. Fraction of the blocks in the most $(r-k_0+1)$ failure-prone groups.

local parity blocks, which suffice to reconstruct $(r+2)$ data block failures. Therefore, we have proved Theorem 1.

To verify this theorem, let's take UFP-LRC(2, 3, 2) for example: the minimal reconstruction cost for decoding single failure is 2, and any $(r+1)=3$ block failures are decodable; and the first group can tolerate any $(r+1)=3$ data blocks and one parity block failures; also, it can tolerate any $(r+2)=4$ data block failures. In contrast, the LRC(9, 3, 2) consisting of the same number of blocks as UFP-LRC(2, 3, 2) doesn't keep these properties: for the first group, it can neither tolerate the lost of $(r+1)=3$ data blocks and the local parity block within it from merely 2 global parity blocks, nor the lost of $(r+2)=5$ data blocks within the most 2 failure-prone groups from merely 2 local and 2 global parity blocks. We can find more examples, such as LRC(8, 2, 2), LRC (12, 2, 2) and so on.

Particularly, Theorem 1 also explains why we choose to increase the blocks per group by one in Definition 1. Because small size groups can tolerate more failure patterns, we try to put more blocks within the most $(r-k_0+1)$ failure-prone groups so as to obtain high reliability. Let CF be the fraction of the blocks (including data blocks and local parities) within these small groups, $CF=\frac{\sum_{i=0}^{r-k_0}(k_0+i+1)}{l+r+\sum_{i=0}^{l-1}(k_0+i)}$. As Fig. 5 shows, higher storage overhead lead to larger CF which is monotonically decreasing along with l. Furthermore, we have to consider the storage overhead when selecting the coding parameters. The storage overhead of UFP-LRC$(2,l,2)$ and UFP-LRC $(3,l,3)$ is 1.18x$\sim$1.8x and 1.17x$\sim$1.71x, respectively. From the standpoint of tradeoff between CF and storage redundancy, both of them are potential candidates for application. □

4.3 Constructing the Generator Matrix

In this section, we first provide a generalized design of the generator matrix. The generalized coding equation is $G\cdot(x_0,x_1,\ldots,x_{k_0-1},y_0,y_1,\ldots,y_{k_1-1},z_0,z_1,\ldots,z_{k_2-1},\ldots)^T=(p_x,p_y,p_z,\ldots,p_0,p_1,\ldots,p_r)^T$. To simplify the searching of coefficients, based on the Vandermonde Matrix, we can construct the generalized generator matrix below:

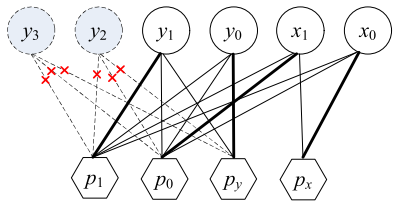


Fig. 6. A reduced Tanner Graph. The broken circles refer to the available data blocks, and the symbol "x" marks the edges to be removed.

$$G=\begin{pmatrix}1&\ldots&1&0&\ldots&0&\ldots&0&0\\0&\ldots&0&1&\ldots&1&0&\ldots&0\\\ldots&\ldots&\ldots&\ldots&\ldots&\ldots&\ldots&\ldots&\ldots\\0&0&\ldots&\ldots&\ldots&0&1&\ldots&1\\a_1&\ldots&a_{k_0}&b_1&\ldots&b_{k_1}&c_1&\ldots&\ldots\\a_1^2&\ldots&a_{k_0}^2&b_1^2&\ldots&b_{k_1}^2&c_1^2&\ldots&\ldots\\\ldots&\ldots&\ldots&\ldots&\ldots&\ldots&\ldots&\ldots&\ldots\\a_1^r&\ldots&a_{k_0}^r&b_1^r&\ldots&b_{k_1}^r&c_1^r&\ldots&\ldots\end{pmatrix}. \quad (1)$$

Now, taking the UFP-LRC(2+4, 2, 2) as example, we illustrate how to choose the specific coding coefficients for the generator matrix. For UFP-LRC(2+4, 2, 2), the coding equation is $G\cdot(x_0,x_1,y_0,y_1,y_2,y_3)^T=(p_x,p_y,p_0,p_1)^T$, then the generator matrix G is

$$G=\begin{pmatrix}1&1&0&0&0&0\\0&0&1&1&1&1\\a_1&a_2&b_1&b_2&b_3&b_4\\a_1^2&a_2^2&b_1^2&b_2^2&b_3^2&b_4^2\end{pmatrix}. \quad (2)$$

Afterwards, we need to determine the value of a and b so as to decode all the information-theoretically decodable 4 failures. For each failure pattern, we can derive a *reduced encoding matrix* $\hat{G}$ based on the *reduced decoding Tanner Graph (TG) of* G [29], which is used to judge if the failure is information-theoretically decodable. Fig. 6 depicts a reduced decoding TG for a decodable four data block (x_0,x_1,y_0,y_1) failures case. Specifically, given this failure case, the reduced decoding TG is derived by removing all available data blocks and their edges and failed parity blocks from the original TG of G. The bold edges show maximum matchings, implying this failure case is information-theoretically decodable. *In this way, we can derive the* $\hat{G}$ *from* G *by removing all the columns that correspond to the available data blocks in the reduced TG and the rows that correspond to the failed parities in the reduced TG from* G.

Below we only present some typical cases to explain the construction of $\hat{G}$ for UFP-LRC(2+4, 2, 2).

Four data blocks fail: Assume the 4 failures are equally divided between two groups, i.e., x_0,x_1,y_0 and y_1, we need 4 equations to decode them, and the reduced encoding matrix $\hat{G}$ is derived as follows:

$$\hat{G}=\begin{pmatrix}1&1&0&0\\0&0&1&1\\a_1&a_2&b_1&b_2\\a_1^2&a_2^2&b_1^2&b_2^2\end{pmatrix}. \quad (3)$$

TABLE 1
A Summary of Performance Metrics

Metrics \ Coding	LRC(k,l,r)	UFP-LRC(k_0,l,r)
Minimal Reconstruction Cost	$\lceil k/l \rceil$	k_0
Storage Overhead	$\frac{k+l+r}{k}$	$\frac{\sum_{i=0}^{l-1}(k_0+i+1)+r}{\sum_{i=0}^{l-1}(k_0+i)}$
Any (r+1) Failures Tolerance	YES	YES
Any (r+1) Data Block and One Parity Block Failures Tolerance	NO	the most $(r-k_0+1)$ failure-prone groups
Any (r+2) Data Block Failures Tolerance	NO	the most $(r-k_0+1)$ failure-prone groups
Maximal Failure Tolerance	$l+r$	$l+r$

Thus, the determinant should be non-singular, $|\hat{G}| = (a_2-a_1)(b_1-b_2)(b_1+b_2-a_1-a_2)=0$.

Three data blocks and one local parity fail: If whole group x plus one data block of group y are lost. This failure pattern is a worst case for decoding. Assuming y_0 fails, consider p_x is lost, we remove the 4th, 5th, 6th columns and the 1st row from G, which results in the following $\hat{G}$:

$$\hat{G} = \begin{pmatrix} 0 & 0 & 1 \\ a_1 & a_2 & b_1 \\ a_1^2 & a_2^2 & b_1^2 \end{pmatrix}. \tag{4}$$

$|\hat{G}| = a_1a_2(a_2-a_1)=0$.

Similarly, if p_y, x_0, x_1 and y_0 fail, we have

$$\hat{G} = \begin{pmatrix} 1 & 1 & 0 \\ a_1 & a_2 & b_1 \\ a_1^2 & a_2^2 & b_1^2 \end{pmatrix}. \tag{5}$$

$|\hat{G}| = b_1(a_2-a_1)(b_1-a_1-a_2)=0$.

The remaining cases are relative simple. To ensure they are decodable, the $\hat{G}$ should be non-singular, and thus have its inverse matrix. By summarizing all cases, we have the following conditions:

$$\begin{aligned} a_i, a_j, b_s, b_t &\neq 0 \\ a_i \neq a_j, b_s \neq b_t; a_i, a_j &\neq b_s, b_t \\ a_i + a_j &\neq b_s + b_t \\ a_i \neq b_s + b_t; b_s &\neq a_i + a_j \end{aligned} \tag{6}$$

It is easy to meet these conditions by choosing the a and b from a finite field GF(2^5). For instance, let $a_1 = 00011, a_2 = 00010, b_1 = 11000, b_2 = 10000, b_3 = 01000$ and $b_4 = 01100$, then the above conditions are satisfied. By selecting the higher/lower order bits of a_i and b_i in a staggered manner within a large finite field space, we can construct long UFP-LRC. Empirically, within the range of our interest, the GF $(2^{\sum_{i=0}^{l-1}(k_0+i)})$ suffices for determining the coding coefficients for UFP-LRC$(k_0,l,2)$.

4.4 Summary

A UFP-LRC(k_0,l,r) consists of $\sum_{i=0}^{l-1}(k_0+i)$ data blocks within l unequal-sized groups, l local parities, one for each group, and r shared global parities. The lower bound of read cost for reconstructing single failure is k_o. We summarize the characteristics of UFP-LRC via a quantitative comparison with LRC. Table 1 lists a summary of performance metrics of the two coding strategies. The UFP-LRC not only possesses all the advantages of LRC, but achieves better reliability and reconstruction performance as we demonstrate shortly.

5 Code Selection

We need to select the ideal coding parameters k_0, l and r from a wealth of choices in practice. In this section we describe and evaluate the tradeoff for the code selection.

5.1 Availability Model and Threshold

Considering the significant demands of reliability in failure-prone large scale cloud storage systems, we first introduce the availability analysis for the maximal recoverable codes. *Instead of using the traditional MTTF, which spans centuries or millennium far longer than disks' actual lifetimes [1], [28], we employ the availability as the reliability metric so as to reflect the impact of disk failures in real world.*

For UFP-LRC(k_0,l,r), the availability of a block represents a statistical probability that a block is available over a large number of failure-prone disks/nodes. It is equal to the number of ways in which we can distribute i unavailable blocks (up to $(l+r)$) on failed disks/nodes multiplied by the number of ways in which we can distribute available blocks (up to $n = l+r+\sum_{i=0}^{l-1}(k_0+i)$) on reachable disks/nodes, given the percentage of decodable i-failure cases, divided by the total number of ways in which we can distribute all of the n blocks on all of the disks/nodes.

Assume we randomly deploy n blocks over N $(\gg n)$ disks/nodes and place each block on a different disk/node so as to avoid correlated failures. Meanwhile assume there are M currently unavailable disks/nodes. Note that UFP-LRC(k_0,l,r) can tolerate any $(r+1)$ failures, and is able to decode up to $(l+r)$ failures probabilistically. Let p_i be the percentage of decodable i-failure cases. Therefore, the availability of a block, denoted by P_a, can be computed as follows:

$$P_a = \sum_{i=0}^{l+r} p_i \frac{\binom{M}{i}\binom{N-M}{n-i}}{\binom{N}{n}}, \tag{7}$$

where $p_i = 1$, if $i \leq (r+1)$, because all i-failure cases are decodable; otherwise, $0 < p_i < 1$. By enumerating all decodable 4-failure cases for UFP-LRC(2+4, 2, 2) and LRC(6, 2, 2), the p_4 is equal to 81.5% and 85.7%, respectively. The reason why UFP-LRC(2+4, 2, 2) results in lower availability than LRC lies in the extra undecodable 4-failure patterns incurred by the group y. For example, the erasure of y_0, y_1, y_2 and p_y is undecodable since there are only two useful global parities.

Nevertheless, the occurrence of undecodable 4-failure pattern in practice is very rare because the failures mostly come from the more failure-prone yet more failure-tolerant small group x. Our design goal is to let UFP-LRC

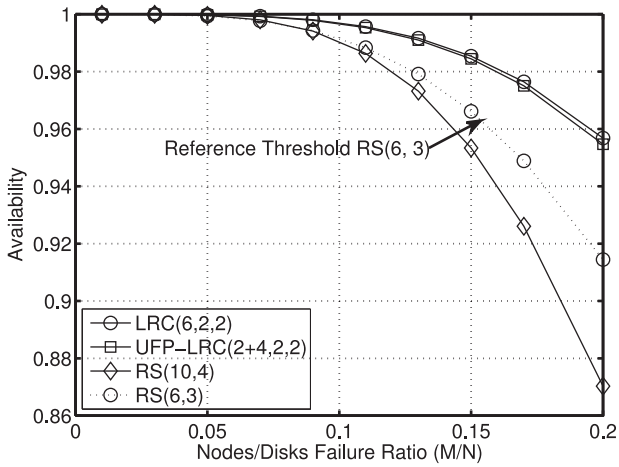


Fig. 7. Comparisons of availability under varying nodes/disks failure ratio.

outperform RS(6, 3) in terms of availability. We will take the block availability of RS(6, 3) as the reference threshold because it not only provides the standard 3-failure tolerant capability but also obtains lower redundancy than traditional 3-replication. In Fig. 7, let $N = 10,000$ and the failure ratio $\frac{M}{N}$ varies from 0.01 to 0.2. UFP-LRC(2+4, 2, 2) can achieve six nines (0.9999996) of availability at the highest points, while RS(6, 3) only provides five nines (0.999998) of availability due to the lack of 4-failure tolerant ability. On the other hand, RS(10, 4) is able to tolerate any 4 failures and thus offers higher availability than RS(6, 3) at the low failure ratio (when $\frac{M}{N}$ is less than 0.01), but its availability falls rapidly along with $\frac{M}{N}$ owing to its low redundancy.

5.2 Tradeoff and Lower Bound

We obtain numerous sets of availability by adjusting the coding parameters. Using the availability of RS(6, 3) as the reference threshold, we only keep the coding parameters sets that yield equal or higher availability than RS(6, 3). Since each block has to be arranged on a different disk/node, the number of disks/nodes in a cluster of storage system limits the total number of blocks in the code. Furthermore, I/O consumption is another constraint to the number of blocks in the face of the data-intensive cloud computing environment. To compare with LRC under the same

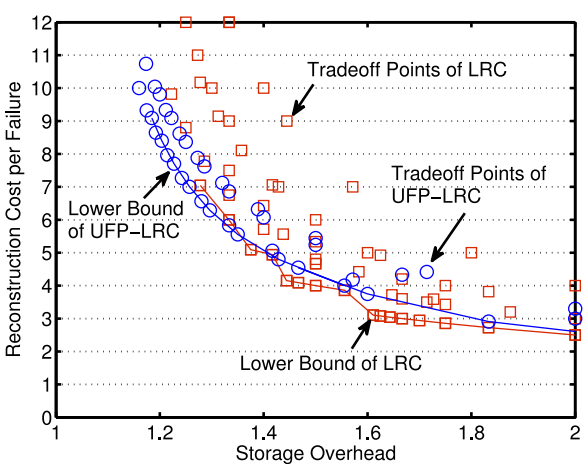


Fig. 8. Lower Bound.

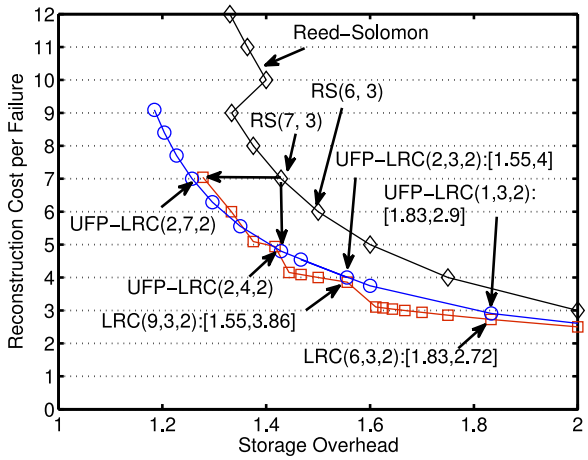


Fig. 9. Comparisons.

condition, we plot the averaged reconstruction cost per single failure in Figs. 8 and 9, given the same failure rate for all blocks (ρ=1). We only plot the storage overhead and the reconstruction cost of non-trival sets in Fig. 8. The coding parameter of each individual point represents certain tradeoff between storage overhead and reconstruction cost. Different coding parameters can lead to the same storage overhead but remarkably different reconstruction cost. For instance, the cost of UFP-LRC(2, 2, 2) is $R = \frac{3\times2+4\times3+2\times5}{2+3+2+2}$=3.1, while the cost of LRC(6, 2, 2) is $R = \frac{8\times3+2\times6}{6+2+2}$=3.6, with nearly the same storage overhead.

Thus it is only sensible to choose those coding parameters yielding lower reconstruction cost as the lower bound. The lower bound curve characterizes some candidates of UFP-LRC for our prototype storage system. Thereby, in Fig. 9, there are only twelve candidates, UFP-LRC(1, l, 2) (l=2 to 5) and UFP-LRC(2, l, 2) (l=3 to 10). Note that the point UFP-LRC(1, 2, 2) is not shown as its storage overhead exceeds 2.

5.3 Lower Bound Curve Comparisons

For (k, r) Reed-Solomon code, we vary the parameters k and r and also obtain a lower bound curve as shown in Fig. 9. Comparisons in Fig. 9 show that both UFP-LRC and LRC achieve better tradeoff points than Reed-Solomon across the range of coding parameters. In particular, compared with RS(7, 3), the UFP-LRC(2, 7, 2) is able to maintain the same reconstruct cost (i.e., R=7) while reducing the storage overhead from 1.42x to 1.25x. This is shown by the horizontal move in Fig. 9. Alternatively, as depicted by the vertical move in Fig. 9, the UFP-LRC(2, 4, 2) can keep the same storage overhead as RS(7, 3) (at 1.42x) while gaining a reduction of 31 percent for reconstruction cost (from R =7 to 4.8).

For UFP-LRC, the point UFP-LRC(2, 4, 2) is a watershed compared to LRC with regard to the reconstruction cost. The tradeoff point of UFP-LRC with less storage overhead than UFP-LRC(2, 4, 2) outperforms the corresponding point of LRC. On the contrary, the point with more storage overhead incurs more reconstruction cost than LRC. For example, with the same reconstruction cost (R=7), UFP(2, 7, 2) uses less storage overhead than LRC(18, 3, 2). In Fig. 9, the values in brackets denote the coordinates of the tradeoff points. UFP-LRC(2, 3, 2) keeps the same storage overhead (at 1.55x) while leading to slightly higher reconstruction cost (R=4) than

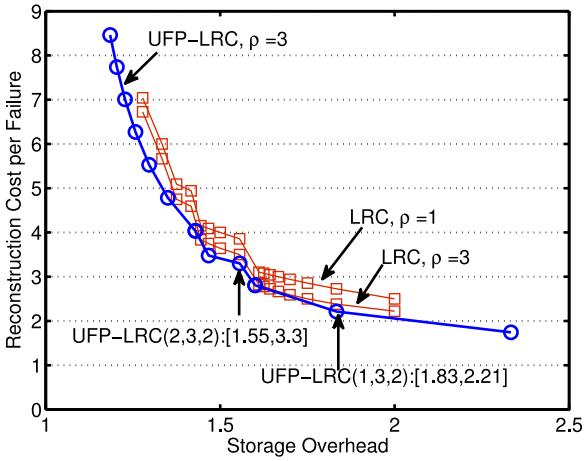


Fig. 10. Impact of failure rate ratio ρ.

LRC(9, 3, 2) (R=3.86). Similarly, the UFP(1, 3, 2) undergoes a slightly higher reconstruction cost than LRC(6, 3, 2), i.e., R=2.9 versus R=2.72, under the same storage overhead (at 1.83x). In fact, this inferior of UFP-LRC disappears under the unequal failure rates as we will detail next.

5.4 Impact of Unequal Failure Rate

In practice, the failure rate may vary across all blocks. To simplify the comparisons, for UFP-LRC(k_0, l, r) on the lower bound curve, we assume that the blocks within the most $(r - k_0 + 1)$ failure-prone groups have relative higher failure rate than others. *According to Definition 2, ρ is the failure rate ratio of the more failure-prone blocks (including the parities) to the other blocks*. Thus the averaged reconstruction cost per single failure R can be computed as follows:

$$R = \frac{\sum_{i=0}^{r-k_0} \rho(k_i+1)k_i + \sum_{i=r-k_0+1}^{l-1}(k_i+1)k_i + \sum_{i=0}^{l-1} rk_i}{\sum_{i=0}^{r-k_0} \rho(k_i+1) + \sum_{i=r-k_0+1}^{l-1}(k_i+1) + r}, \tag{8}$$

where R equals to *the total reconstruction cost divided by the number of failures during certain time period*, the $\sum_{i=0}^{r-k_0} \rho(k_i+1)k_i$ is the overall reconstruction cost of the most $(r - k_0 + 1)$ failure-prone groups, and the $\sum_{i=0}^{r-k_0} \rho(k_i+1)$ denotes the number of failures occurring within the most $(r - k_0 + 1)$ failure-prone groups during certain time period. According to above equation, for the specific lower bound points of UFP-LRC, only 3 or 5 blocks (including the local parities) have higher failure rate than others. To compare with LRC, the number of the more failure-prone blocks is set to 4 (the mean) for each LRC point. Therefore, for LRC(k, l, r), twe have $R = \frac{4\rho\lceil k/l\rceil + (k+l-4)\rho\lceil k/l\rceil + kr}{4\rho + k + l + r - 4}$.

As shown in Fig. 10, UFP-LRC outperforms LRC across all points on lower bound curve when ρ exceeds 3. If we keep the storage overhead the same, reconstruction cost in UFP-LRC is lower than that in LRC. On the other hand, if we keep reconstruction cost the same, UFP-LRC can save storage overhead compared to LRC. For example, when ρ=3.5, the UFP-LRC(2, 3, 2) turns out to obtain lower reconstruction cost (R=3.3) than LRC(9, 3, 2) at the same storage overhead, and UFP-LRC(1, 3, 2) also reduce the reconstruction cost from 2.9 to 2.21, compared to the value in Fig. 9. It is worth noting that, the ρ=3 is within the reasonable range of failure rate ratio as shown in Fig. 1 (up to 5-6). When ρ varies within the range of our interest, we can choose an appropriate point along the trade-off curve, which can reduce storage overhead and reconstruction cost at the same time.

TABLE 2
Comparisons of Decodability Ratio (p_t)

Codes \ t	4	5	6
LRC(6,3,2)	0.9545	0.8377	0
UFP-LRC(1,3,2)	0.9364	0.8312	0
LRC(9,3,2)	0.9491	0.4635	0
UFP-LRC(2,3,2)	0.945	0.4341	0
LRC(12,4,2)	0.98	0.3669	0.2926
UFP-LRC(3,3,3)	**1.0**	**0.976**	**0.6563**

6 Numerical Analysis and Comparisons

In this section, we perform a comprehensive comparison of typical lower bound points (within the range of our interest) of UFP-LRC and LRC through numerical analysis.

6.1 Decodability Ratio

Properties (2) and (3) in Theorem 1 imply that the UFP-LRC has better failure tolerance in many cases. However, UFP-LRC also introduces some extra undecodable failure patterns due to the groups of size larger than r. In Table 2, we compute the decodability ratio for several lower bound tradeoff points by enumerating all t-failure ($r \le t \le (l + r)$) patterns. Table 2 shows that some points of UFP-LRC on lower bound curve lead to a slightly lower decodability ratio than the corresponding points of LRC at each t value with the same redundancy. Particularly, compare to LRC(9, 3, 2), the UFP-LRC(2, 3, 2) achieves a little lower p_4 and p_5 when t=4 and 5. For example, for UFP-LRC(2, 3, 2), the p_5 is equal to 0.4341 (i.e., 869 decodable cases out of C_{14}^5).

Nonetheless, the key question is whether UFP-LRC can outperform LRC in terms of availability. The answer is positive. At the same storage overhead, UFP-LRC(3, 3, 3) outperforms the lower bound point LRC(12, 4, 2) at all t values. When t=5, the lower bound point LRC(12, 4, 2) merely obtain a p_5=0.3669. In contrast, the UFP-LRC(3, 3, 3) provides a much higher p_5=0.976 due to the arbitrary five data blocks failure tolerance of the first group. Undoubtedly, we can find more candidates if we make a tradeoff between availability and other performance.

6.2 Availability

Indeed, the UFP-LRC obtains the unequal failure protections for various blocks by compromising minor availability. Fortunately, this slight loss of availability generally is not important in practice, as UFP-LRC can achieve desired higher availability than our reference threshold RS(6, 3), as shown in Fig. 11. As we can see, each lower bound point of UFP-LRC achieves our desired availability, which is always higher than that of our reference threshold RS(6, 3). There is only negligible difference between the curve of UFP-LRC and the corresponding LRC. For example, the availability of UFP-LRC(2, 3, 2) is about 4.00E-08 lower than that of the

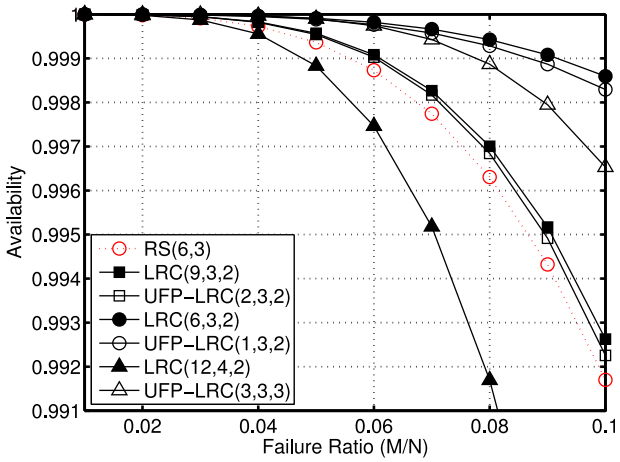


Fig. 11. Comparisons of availability under varying failure ratio.

LRC(9, 3, 2) at $\frac{M}{N}$=0.02. Actually, the reliability of UFP-LRC is better than that of LRC, as we will show in our system evaluation shortly. According to the occurrence of failure patterns in practice, mostly failure cases will take place within the more failure-prone yet more failure-tolerant small groups. As a result, UFP-LRC will yield equal or even higher reliability than LRC in general.

6.3 Reconstruction Cost for Single Failure

The examples in Fig. 4 indicate that UFP-LRC will gradually overtake LRC for reconstruction performance as ρ grows. Compared with the typical lower bound points of LRC, Fig. 12 further demonstrates this characteristic of UFP-LRC. We can clearly see the two intersection points (ρ=1.7 and 2) where the UFP-LRC(2, 3, 2) and the UFP-LRC(1, 3, 2) begins to achieve lower reconstruction cost. In addition, both UFP-LRC schemes decline more rapidly than corresponding LRC schemes, their advantages will be more obvious when the failure rate ratio exceeds 4.

7 Implementation in Storage System

To confirm the analytical savings provided by the UFP-LRC in terms of bandwidth utilization and I/Os, we deployed the UFP-LRC over a Hadoop based prototype cloud storage system. We now describe the implementation of the system and reconstruction operations.

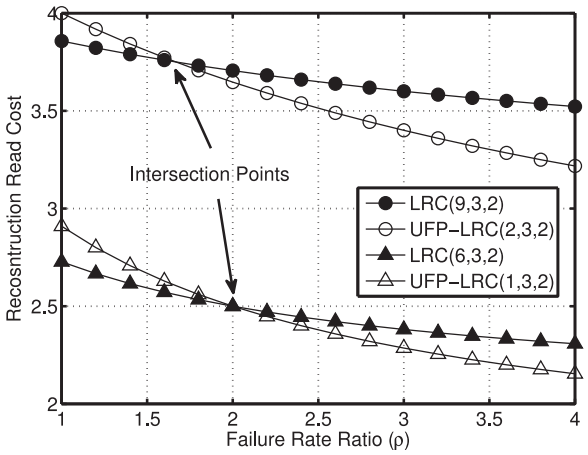


Fig. 12. Comparisons of reconstruction cost for the typical lower bound points.

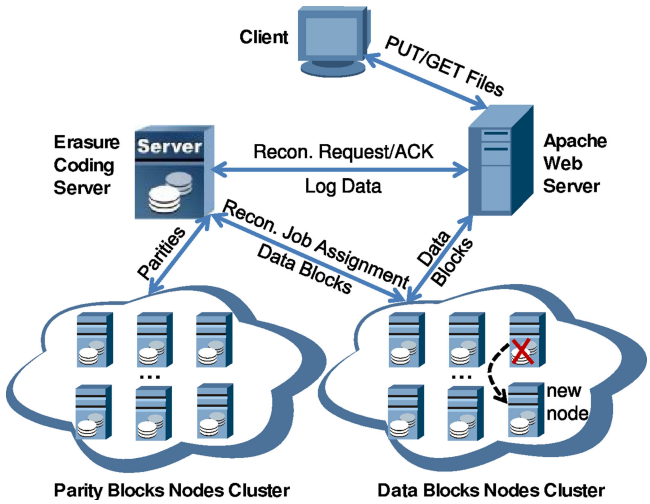


Fig. 13. Overview of our prototype storage system architecture. Our storage system erasure codes the cold data lazily in the background and deletes their additional copies.

7.1 Overview of Our Storage System Architecture

Here we provide an overview of the our prototype system with erasure coding. As the Fig. 13 depicts, the architecture of our prototype has three layers: client, servers (web server and erasure coding server) and clusters (parity blocks nodes and data blocks nodes). The workload mainly runs on clusters of inexpensive commodity hardware. Each cluster is composed of a name node and six data nodes, and each node has at least 2 disks. The name node is the centerpiece of a cluster. It keeps the directory tree of all files, and tracks where across the cluster the data is kept. While a data node stores data and then responds to requests from the name node for files operations.

Clients can PUT/GET the files directly to/from the storage clusters, after having obtained the addresses of the host nodes via Apache web server. Based on the replica mechanism of HDFS, the files are originally written to 3 copies to keep data available. Then the erasure coding server erasure codes the *cold data* that not be requested during certain time period lazily in the background and then deletes their original 3 copies. We omit the details of scanning and management for cold data. To improve the throughput, the parity blocks will be separately stored over the relative reliable (low failure rate) parity blocks nodes cluster which processes low workload in the background. The encoded data blocks are deployed over the data blocks nodes cluster suffering foreground data-intensive workload.

The erasure coding server is able to locate the failed disks/nodes. In case of a data block node failure, the erasure coding server designates a new node to perform reconstruction operation. Especially, the GET operation that happens to request currently unavailable data will trigger a reconstruction on-the-fly.

7.2 UFP-LRC Implementation

Considering the tradeoff between various metrics, we choose to implement the UFP-LRC(2, 3, 2) in our prototype system as a complementary technique to the built-in 3-way data replication of Hadoop. Each file to be stored is fragmented into blocks of 64 MB. The unit symbol for encoding/decoding operation is 16 bits in default, which benefits

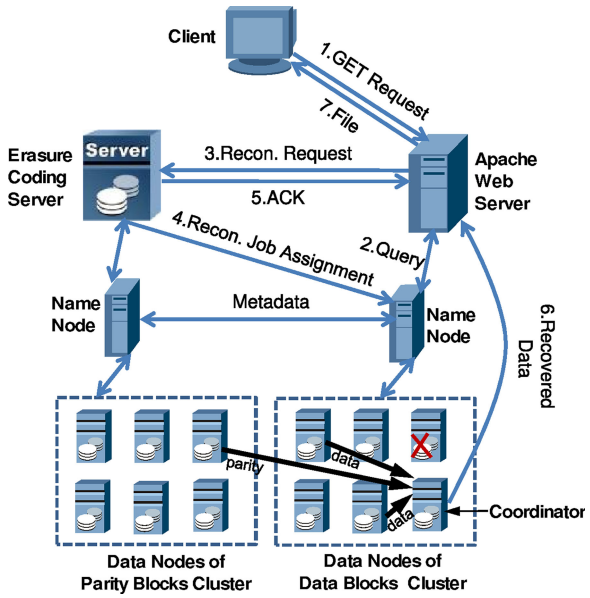


Fig. 14. Reconstruction procedure for degraded read.

the multiplication operating over a desirable small Galois Field. Nevertheless, the GF(2^{16}) has sufficient elements space to find the required coefficients.

Striping Policy: One solution to use UFP-LRC is to encode only blocks within a single file together so as to form a stripe (including data and parity blocks). That works for big files, however, in case of small files, there are not enough blocks within each file to be encoded efficiently. Thus we design the striping policy for encoding based on the statistics from several clusters of cloud storage vendors. We found that more than 70 percent of space is occupied by the small files whose size is less than 2 GB (64×32 MB blocks) while more than 85 percent of space is used to store directory whose size is more than 2 GB. Therefore, striping blocks within a single directory seems to be our favorable candidate. We employ a MapReduce job to compute parity blocks for each stripe in a parallel pattern. During a map phase, each mapper outputs a stripe by reading the original nine data blocks and calculating the parity blocks. In the reduce phase, a single reducer generates metadata gathered from each mapper.

Stream Operations: We exploit all related optimization techniques to streamline the process. In case of encoding, the unit symbols to be encoded will be delivered to a FSDataInputStream object. Also, we need a FSDataInputStream object to search the address of blocks to be encoded from metadata, and then use a FSOutputStream object to update the block-id list of metadata on the name nodes after generating the parity blocks.

7.3 Encoding Procedure

In case of finding out the cold data to be encoded via a periodically scanning over the data blocks nodes cluster, the erasure coding server initiates a 4-step encoding procedure as follows.

The first step is to arrange the cold data with the striping policy mentioned above, after which the erasure coding server maps the nine data blocks to specific disks with a random hash algorithm. The mapping mechanism ensures each data block to be placed on *distinct target disk* with different failure rate. The target disks' metadata (addresses and IDs) are cached for later use. According to the failure rates of these target disks which are indicated by the parameters of disk health (like SMART [34]), the erasure coding server can determine which data block should be put into the group x, y and z of UFP-LRC(2, 3, 2), respectively. In general, it is hard to derive an accurate failure rate, in the prototype system we use 2-bit indicator scheme to indicate the possibility of disk failure. Specifically, let the three data blocks groups (x, y and z) of UFP-LRC(2, 3, 2) correspond to three state, i.e., high, low, and very low, which are denoted by 11, 10 and 01.

Upon grouping the data blocks, then in step 3, the erasure coding server hosts a MapReduce job which performs the encoding based on the streaming technique. To avoid interfering the foreground workload processing, all reading and encoding operations are carried out by the erasure coding server in the background in case of light workload. As soon as the erasure coding server successfully executes the encoding operations, all additional 3 replicas can be deleted to save storage space. In the final step, all generated parity blocks are deployed to distinct disks of parity blocks cluster via a hash mapping, meanwhile the data blocks are sent back to the target disks of data blocks cluster according to the cached location metadata. Afterwards, the erasure coding server appends an entry to its encoding inventory which consists of all metadata.

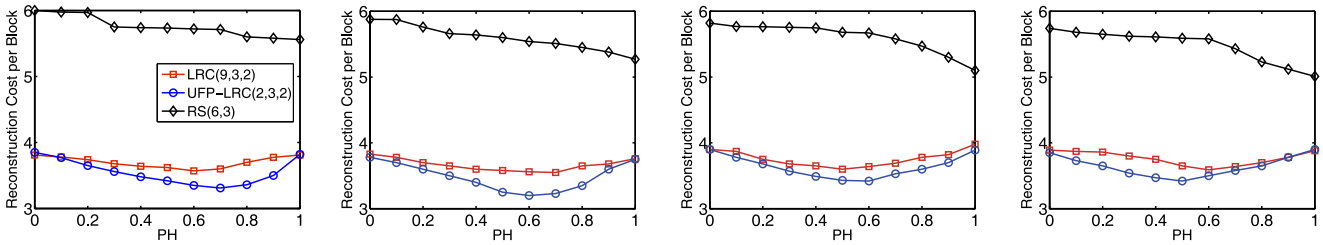
7.4 Reconstruction

There are two kinds of reconstruction in our system. Usually, the reconstruction is an active repair. If losing heartbeats from a specific disk/node for a while, the erasure coding server takes for granted that the disk/node fails and thus designates a new node (named coordinator) to perform an active yet lazy reconstruction, as shown in Fig. 13.

Unlike the time-consuming active repair responsible for recovering all missing data blocks, the passive reconstruction only is triggered by on-demand read. As pictured in Fig. 14, the read triggers a passive reconstruction on-the-fly in case that the GET operation requests for some currently unavailable data. The web server sends a RECONS_REQUEST_MSG to erasure coding server, which contains the IP addresses, as well as IDs of the lost data blocks. Upon the receipt of the RECONS_REQUEST_MSG, the erasure coding server sends a RECONS_JOB_ASSIGNMENT_MSG to the name node of data blocks cluster, and respond to web server with a ACK. The RECONS_JOB_ASSIGNMENT_MSG contains the metadata of the parity and data blocks need to be read for reconstruction, as well as the address of the coordinator who performs the reconstruction. To further save the reconstruction cost, the selected coordinator should have at least one of the data blocks involved in RECONS_JOB_ASSIGNMENT_MSG. The RECONS_JOB_ASSIGNMENT_MSG is forwarded to the coordinator to initiate the reconstruction operations. The coordinator recovers the lost data blocks with its local cached coefficients. Eventually, the recovered data blocks are built up into a file by the web server and sent back to client.

8 Performance Evaluation

We perform extensive experiments on 12 data nodes with 28 disks (which is composed of SSDs and hard drives, 16 disks



(a) FD=5 (b) FD=6 (c) FD=7 (d) FD=8

Fig. 15. Average reconstruction cost per lost block for various *FD* with varying *PH*.

in data blocks cluster and 12 disks in parity blocks cluster), 2 name nodes (quad-core Intel E3-1225v5 3.3 GHz processor, 8 GB RAM), 1 web server, 1 erasure coding server (quad-core Intel i5-4460 3.2 GHz, 4 GB RAM) and 3 client nodes that all are connected through a 1 Gbps network.

To simulate the unequal failure rate pattern, we randomly chose some disks as failed ones by holding their heartbeats periodically, based on the traces of SETI@home Desktop Clouds in failure trace archive [35]. Let *FD* denote the maximal number of failed disks. Thus some disks of the *FD* selected disks fail at high frequencies (i.e., the hourly failure rates ranging from 20 to 40 percent, indicating a MTTF varying from 5 to 2.5 hours or so), while the remaining disks fail at relative low frequencies (i.e., the hourly failure rates ranging from 10 to 20 percent, implying a MTTF varying from 10 to 5 hours). Note that the failure rate of a disk is variable due to the random selection of the failure traces. Per our scheme, data blocks deployed on these disks with high failure rates are put into the small group of individual stripe. Moreover, we let *PH* be the percentage of disks failing at high rates. For example, when *PH*=0.5, roughly half of the failed disks randomly fail at high rate according to the traces.

8.1 Average Reconstruction Cost per Lost Block

In this experiment, we measure the average reconstruction cost for a lost block (data or parity block), which is equal to the total read/downloading cost for reconstruction, divided by the sum of the number of lost blocks within an individual stripe. Since the *FD* exceeds the erasure tolerance of our codes, there are a few undecodable cases which are not taken into account. We will detail the percentage of undecodable cases shortly.

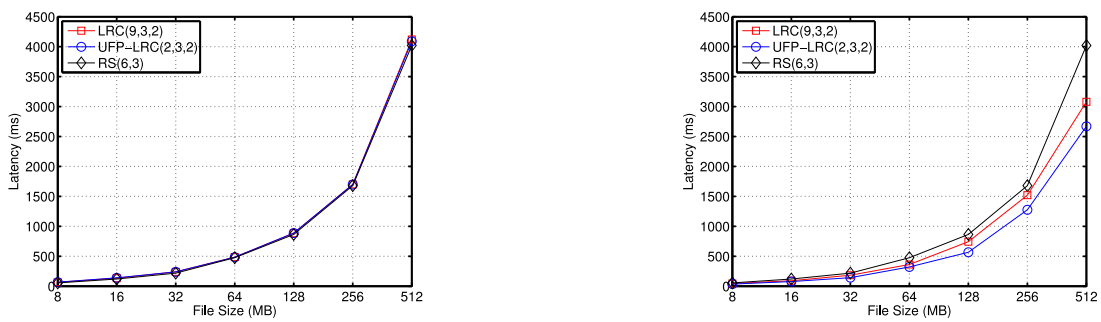
The results are depicted in Fig. 15. For the three coding schemes that have nearly the same storage overhead, we performed the same number of reconstructions for them and calculated the average value. The reconstruction cost of UFP-LRC(2,3,2) is slightly higher than LRC(9,3,2) when *PH* is small. However, the cost of UFP-LRC(2,3,2) decreases faster with the increase of *PH*, and eventually reaches a lower limit when *PH* reaches certain points (e.g., 0.5, 0.6 or 0.7). For example, with *FD*=6, the reconstruction cost per block of LRC(9,3,2) is between 3.55 to 3.83, while the faster UFP-LRC(2,3,2) recoveries a block at a cost of 3.2 to 3.78, obtaining about 10 percent cost savings. The advantage of UFP-LRC(2,3,2) owes to the enlarged variance of failure rates among failed disks. This is consistent with our numeric analysis in Fig. 12. Afterwards, UFP-LRC(2,3,2) and LRC(9,3,2) gradually converge when the *PH* approaches 1 because of the shrinking difference of failure rates of disks.

The reconstruction cost of RS(6,3) is close to be linear with the change to the *PH* because of its fixed reconstruction cost for multiple lost blocks. With the increase of *FD*, this characteristic of RS(6,3) is more apparent, i.e., still reading six surviving blocks to recover two or three lost blocks. That is why the RS codes generally employ a delaying recovery strategy, which can reduce the average reconstruction bandwidth but not facilitate the reconstruction on-the-fly.

8.2 Encoding/Decoding Latency

We compare the real-time latency of encoding and decoding operations for UFP-LRC and other two codes. We employed a variety of sizes of unit symbol for encoding/decoding operations, i.e., 16 bits, 32 bits, 64 bits and 128 bits, while the 16-bit unit symbol obtained the optimal arithmetic performance in our system (Intel i5-4460 3.2 GHz, 4 GB memory).

Fig. 16 plots the encoding/decoding latency with varying file size. We can see that encoding latencies of the three codes are fairly similar, while UFP-LRC and LRC being faster than RS in terms of decoding owing to their locality.



(a) Latency of encoding (b) Latency of decoding

Fig. 16. Latency of arithmetic operations.

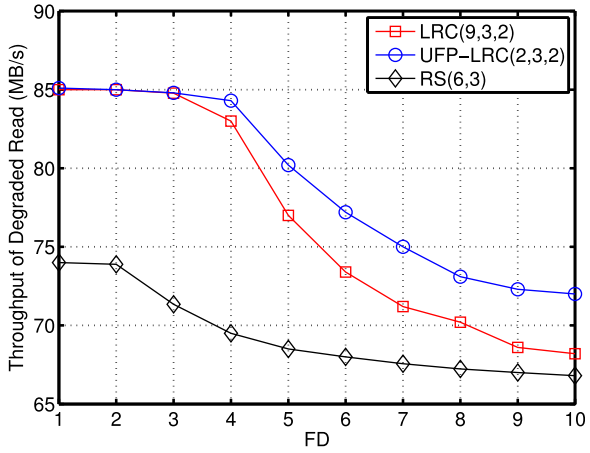


Fig. 17. Throughput of reconstruction on-the-fly.

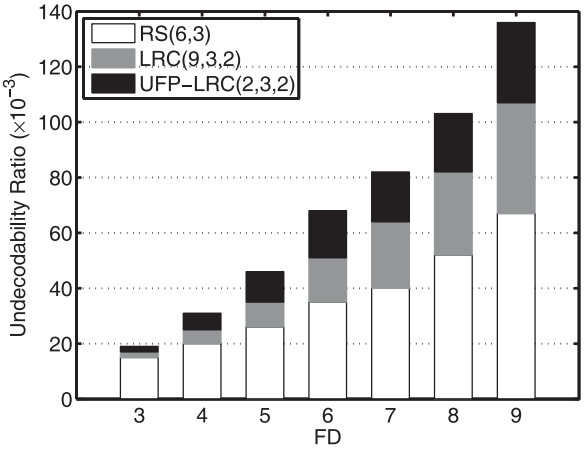


Fig. 18. Decodability ratio with varying *FD*.

For UFP-LRC and LRC, the decoding is a little faster than encoding in that fewer blocks are involved in arithmetic operations. With regard to decoding latency, our experimental results show a 8 to 12 percent improvement of UPF-LRC over LRC, because a smaller number of blocks are involved in reconstruction. For example, the average latency of decoding 64 MB and 512 MB file is 322 ms and 2670 ms for UFP-LRC(2,3,2), 363 ms and 3080 ms for LRC (9,3,2), and 476 ms and 4020 ms for RS(6,3), respectively. Nevertheless, the latencies of arithmetic operations are typically in microseconds and becoming neglectable compare to the latency of I/Os and data transferring during the reconstructions.

8.3 Throughput of Reconstruction On-The-Fly

To evaluate the performance of reconstruction on-the-fly, We carried out a set of experiments concurrently running on 3 client nodes, with a program periodically submitting GET requests for files to the web server when the system suffers from disk failures. The size of requested files ranges from 1 MB to 1 GB. On receipt of the RECONS_REQUEST_MSG from web server, the erasure coding server then initializes a reconstruction procedure for the reconstruction on-the-fly.

From Fig. 17, we observe that the average throughput for the three coding schemes does not apparently degrade until the *FD* reaches 2 or 3. This is because of the fact that the client requests do not saturate the capacity of the remaining disks/nodes. When *FD*=2, 14 out of 16 disks are still available. Indeed, when the *FD* exceeds 3, an obvious degradation has been observed because the probability to request a data block on failed disks increases, and accessing those currently unavailable data blocks requires performing a reconstruction on-the-fly. Without parity blocks loss in the nodes cluster, UFP-LRC(2,3,2) and LRC(9,3,2) can avoid costly reconstruction of parity blocks, so as to improve the throughput compared to RS(6,3). UFP-LRC(2,3,2) is able to obtain about 10 to 15 percent higher throughput than LRC (9,3,2) owing to the high failure rate ratio across the failed disks. As *FD* increases, the throughput of UFP-LRC or LRC drops more rapidly compared with RS(6,3). However, there is a slight throughput drop for RS(6,3) due to increasing number of parity blocks to be fetched for repair by the coordinator.

8.4 Undecodability

We conclude the evaluation by comparing the three codes' undecodability ratio which is equal to the number of undecodable cases divided by the total number of reconstructions. Since the undecodable cases in practice are rare, we used a large number of failed disks to increase their occurrences. The statistic undecodable cases mainly come from the disks with high failure rate running above experiments, as the *FD* exceeds the maximal erasure tolerance.

The results are shown in Fig. 18. We can see that the RS (6,3) has the highest undecodability ratio due to its relative weak 3-failure tolerance. Additionally, for undecodability, the UFP-LRC(2,3,2) and LRC(9,3,2) are comparable as analyzed in Table 2. Actually, the undecodability ratio of UFP-LRC(2,3,2) is slightly lower than LRC(9,3,2) as *FD* grows beyond 7, as more failure cases take place within its more failure-tolerant small groups. These results confirm our analysis in Section 6.2.

9 Conclusion

In this paper, we make the first attempt to provide unequal failure protection for all blocks in cloud storage systems. Our proposed technique, UFP-LRC, divides data blocks into several unequal-sized groups, assigning more failure-prone blocks into smaller groups. This way, UFP-LRC achieves stronger erasure failure protection as well as significant reconstruction cost savings for more failure-prone blocks. This leads to a substantial improvement in reliability and reconstruction performance. Our analytical results show that UFP-LRC gradually outperforms LRC along the increase of failure rate ratio. Extensive evaluations on our prototype storage system also validate that, compared to LRC, UFP-LRC can achieve a 10 to 15 percent improvement in throughput while retaining a comparable overall reliability.

Acknowledgments

This work is partially supported by the National Science Foundation of China under Grant No. 61572181, 61472131, 61772191, 61433019 and U1611261; National Security Project under Grant No.BMK2017B02; Science and Technology Key Projects of Hunan Province under Grant No. 2015TP1004, 2016JC2013 and 2016JC2012; Huxiang Excellent Youth Project under Grant No.2017RS3018.

REFERENCES

[1] J. Meza, Q. Wu, S. Kumar, and O. Mutlu, "A large-scale study of flash memory failures in the field," in *Proc. ACM SIGMETRICS/ Int. Conf. Measure. Modeling Comput. Syst.*, 2015, pp. 177–190.

[2] S. Sankar, M. Shaw, K. Vaid, and S. Gurumurthi, "Datacenter scale evaluation of the impact of temperature on hard disk drive failures," *ACM Trans. Storage*, vol. 9, no. 2, Jul. 2013, Art. no. 6.

[3] D. Ford, et al., "Availability in globally distributed storage systems," in *Proc. 9th USENIX Symp. Operating Syst. Des. Implementation*, 2010, pp. 1–7.

[4] Q. Liu, D. Feng, H. Jiang, and Y. Hu, "Z codes: General systematic erasure codes with optimal repair bandwidth and storage for distributed storage systems," in *Proc. IEEE 34th Symp. Reliable Distrib. Syst.*, 2015, pp. 212–217.

[5] H. Chen, Y. Hu, P. Lee, and Y. Tang, "NCCloud: A network-coding-based storage system in a cloud-of-clouds," *IEEE Trans. Comput.*, vol. 63, no. 1, pp. 31–44, Jan. 2014.

[6] B. Calder, et al., "Windows azure storage: A highly available cloud storage service with strong consistency," in *Proc. ACM Symp. Operating Syst. Principles*, 2011, pp. 143–157.

[7] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The hadoop distributed file system," in *Proc. IEEE Symp. Mass Storage Syst. Technol.*, 2010, pp. 1–10.

[8] S. Ghemawat, H. Gobioff, and S. Leung, "The google file system," in *Proc. 9th ACM Symp. Operating Syst. Principles*, Dec. 2003, vol. 37, no. 5, pp. 29–43.

[9] H. Weatherspoon and J. D. Kubiatowicz, "Erasure coding versus replication: A quantitative comparison," in *Proc. Int. Workshop Peer-to-Peer Syst.*, 2002, pp. 328–337.

[10] Z. Zhang, A. Deshpande, E. T. X. Ma, and D. Narayanan, "Does erasure coding have a role to play in my data center," Microsoft, Redmond, Washington, USA, Tech. Rep. MSR-TR-2010–52, 2010.

[11] C. Huang, et al., "Erasure coding in windows azure storage," in *Proc. USENIX Annu. Techn. Conf.*, Jun. 2012, pp. 2–2.

[12] (2016). [Online]. *AWS Fault Tolerance and High Availability*, https://media.amazonwebservices.com/architecturecenter/ AWS ac ra ftha 04.pdf

[13] J. C. Corbett, et al., "Spanner: Google's globally-distributed database," in *Proc. USENIX Conf. Operating Sys. Des. Implementation*, 2012, pp. 251–264.

[14] M. Sathiamoorthy, et al., "XORing elephants: Novel erasure codes for big data," *Proc. VLDB Endowment*, vol. 6, no. 5, 2013, pp. 325–336.

[15] M. Subramanian, et al., "f4: Facebook's warm BLOB storage system," in *Proc. USENIX Conf. Operating Sys. Des. Implementation*, 2014, pp. 383–398.

[16] B. Fan, W. Tantisiriroj, L. Xiao, and G. Gibson, "DiskReduce: Replication as a prelude to erasure coding in data-intensive scalable computing," Carnegie Mellon University, Pittsburgh, PA, USA, Tech. Rep. CMU-PDL-11–112, 2011.

[17] M. Whele. (2013). [Online], *EMC ATMOS Cloud Storage Architecture*, https://poland.emc.com/collateral/software/white-papers/ h9505-emc-atmos-archit-wp.pdf

[18] D. Borthakur, *HDFS and Erasure Codes (HDFS-RAID)*, Aug. 2009. [Online]. Available: http://hadoopblog.blogspot.com/2009/08/ hdfs-and-erasure-codes-hdfs-raid % .html

[19] M. Xia, et al., "A tale of two erasure codes in HDFS," in *Proc. 13th USENIX Conf. File Storage Technol.*, 2015, pp. 213–226.

[20] K. V. 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.*, 2015, pp. 81–94.

[21] J. Li and B. Li, "Cooperative repair with minimum-storage regenerating codes for distributed storage," in *Proc. IEEE Conf. Inf. Comput. Commun.*, 2014, pp. 316–324.

[22] A. Dimakis, K. Ramchandran, Y. Wu, and S. Changho, "A survey on network codes for distributed storage," *Proc. IEEE*, vol. 99, no. 3, pp. 476–489, Mar. 2011.

[23] F. Andr, A. Kermarrec, E. L. Merrer, G. Straub, N. L. Scouarnec, and A. van Kempen, "Archiving cold data in warehouses with clustered network coding," in *Proc. 9th ACM Eur. Conf. Comput. Syst.*, 2014, Art. no. 21.

[24] D. Papailiopoulos, J. Luo, A. Dimakis, and C. Huang, "Simple regenerating codes: Network coding for cloud storage," in *Proc. IEEE Conf. Inf. Comput. Commun.*, 2012, pp. 2801–2805.

[25] Y. Han, P. Hung-Ta, R. Zheng, and H. M. Wai, "Efficient exact regenerating codes for byzantine fault tolerance in distributed networked storage," *IEEE Trans. Commun.*, vol. 62, no. 2, pp. 385–397, Feb. 2014.

[26] K. Rashmi, N. Shah, and P. Kumar, "Optimal exact-regenerating codes for distributed storage at the MSR and MBR points via a product-matrix construction," *IEEE Trans. Inform. Theory*, vol. 57, no. 8, pp. 5227–5239, Aug. 2011.

[27] Locally Repairable Erasure Code Plugin. (2016). [Online]. Available: http://docs.ceph.com/docs/hammer/rados/operations/erasure-code-lrc

[28] B. Schroeder and G. Gibson, "Disk failures in the real world: What does an MTTF of 1,000,000 hours mean to you?" in *Proc. 5th USENIX Conf. File Storage Technol.*, 2007, Art. no. 1.

[29] 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, pp. 1–28, Mar. 2013.

[30] C. Huang, "LRC Erasure Coding in Windows Storage Spaces," in *Proc. Storage Developer Conf.*, 2013.

[31] D. Papailiopoulos and A. G. Dimakis, "Locally repairable codes," *IEEE Trans. Inform. Theory*, vol. 60, no. 10, pp. 5843–5855, Oct. 2014.

[32] V. Cadambe and A. Mazumdar, "Bounds on the size of locally recoverable codes," *IEEE Trans. Inform. Theory*, vol. 61, no. 11, pp. 5787–5794, Nov. 2015.

[33] X. Song, X. Peng, J. Xu, G. Shi, and F. Wu, "Unequal error protection for scalable video storage in the cloud," in *Proc. IEEE Int. Conf. Multimedia Expo*, 2015, pp. 1–6.

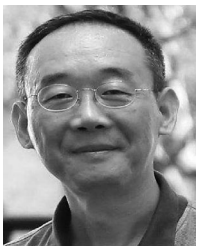
[34] E. Pinheiro, W. Weber, and L. A. Barroso, "Failure trends in a large disk drive population," in *Proc. 5th USENIX Conf. File Storage Technol.*, 2007, pp. 17–29.

[35] D. Kondo, B. Javadi, A. Iosup, and D. Epema, "The failure trace archive: Enabling comparative analysis of failures in diverse distributed systems," in *Proc. 10th IEEE/ACM Int. Conf. Cluster Cloud Grid Comput.*, 2010, pp. 398–407.

Yupeng Hu received the MS and PhD degrees in computer science from Hunan University, China, in 2005 and 2008, respectively. He is currently an associate professor in the College of Computer Science and Electronic Engineering, Hunan University. He was in the Department of Computer Science and Engineering, UT-Arlington as a visiting scholar from 2015 to 2016. He was also with IBM China Development Laboratory as an academic visitor in 2012. His research interests include storage systems, erasure coding, and security. He is a senior member of the IEEE and ACM.

Yonghe Liu received the BS and MS degrees from Tsinghua University, and the PhD degree from Rice University, in 1998, 1999, and 2004 respectively. He is currently an associate professor in the Department of Computer Science and Engineering, UT-Arlington. Before joining CSE@UTA in January 2004, he worked in the Texas Instruments for about two and half years. His research focuses on networking, wireless systems, and their system prototyping. He is a member of the IEEE.

Wenjia Li (M'11) received the Ph.D. degree in computer science from the University of Maryland Baltimore County, Baltimore, MD, USA, in 2011. In 2014, he joined the Department of Computer Science, New York Institute of Technology, New York, NY, USA, where he is currently an Assistant Professor. He was an Assistant Professor of computer science with Georgia Southern University, Statesboro, GA, USA, from 2011 to 2014. He has authored or co-authored over 50 peer-reviewed publications in various journals and conference proceedings. His current research interest include cyber security, mobile computing, and wireless networking, particularly security, trust, and policy issues for wireless networks, cyber-physical systems, Internet of Things, and intelligent transportation systems. His research has been supported by the National Institute of Health (NIH) and the U.S. Department of Transportation Region 2 University Transportation Research Center (UTRC).



Keqin Li is a SUNY distinguished professor of computer science. His current research interests include parallel computing and high-performance computing, distributed computing, energy-efficient computing and communication, heterogeneous computing systems, cloud computing, big data computing, CPU-GPU hybrid and cooperative computing, multicore computing, storage and file systems, wireless communication networks, sensor networks, peer-to-peer file sharing systems, mobile computing, service computing, Internet of things, and cyber-physical systems. He has published more than 460 journal articles, book chapters, and refereed conference papers, and has received several best paper awards. He is currently or has served on the editorial boards of the *IEEE Transactions on Parallel and Distributed Systems*, the *IEEE Transactions on Computers*, the *IEEE Transactions on Cloud Computing*, the *IEEE Transactions on Services Computing*, and the *IEEE Transactions on Sustainable Computing*. He is a fellow of the IEEE.

Kenli Li received the PhD degree in computer science from the Huazhong University of Science and Technology, China, in 2003. He was a visiting scholar with the University of Illinois at Urbana Champaign, from 2004 to 2005. He is currently the Dean of College of Computer Science and Electronic Engineering, Hunan University. His major research interests include parallel computing, cloud computing, and big data processing. He is currently served on the editorial boards of the *IEEE Trans. on Computers*, the *International Journal of Pattern Recognition,* and the *Artificial Intelligence*. He is a senior member of the IEEE.

Nong Xiao received the BS and PhD degrees in computer science from the National University of Defense Technology, China, in 1990 and 1996, respectively. He is currently a professor in the School of Data and Computer Science, Sun Yet-Sen University. His research interests mainly include grid computing, storage, and architecture. He is a member of the IEEE.

Zheng Qin received the PhD degree in computer science from Chongqing University, P. R. China, in 2001. He is a professor in the College of Computer Science and Electronic Engineering, Hunan University, China. His main interests include the computer networking, network and information security, big data and cloud computing. He is a member of the IEEE.