Optimal Codes for Distributed Storage

Jian Ren[1], Jian Li[2] and Tongtong Li[1]
[1] Dept of ECE, Michigan State University, Email: {renjian, tongli}@msu.edu
[2] School of Electronic and Information Engineering, Beijing Jiaotong University, E-mail: lijian@bjtu.edu.cn

2023 International Conference on Computing, Networking and Communications (ICNC) | 978-1-6654-5719-4/23/$31.00 ©2023 IEEE | DOI: 10.1109/ICNC57223.2023.10074312

***Abstract*—Regenerating codes are a class of distributed storage codes that allow for efficient repair of failed nodes, as compared to traditional erasure codes, which enables it to achieve high data reliability, security, and cost-efficiency and a critical infrastructure of the computing system. Existing data storage largely depends on a centralized cloud, which is not only costly but also vulnerable to single points of failure and other types of security attacks. To provide data security, data encryption has to be used, which requires extensive computing power and cumbersome key management. Distributed storage system (DSS) is being widely viewed as a natural solution to future online data storage due to improved access time and lower storage cost. However, the existing DSS also has the limitations of low storage efficiency and lack of data security. In this paper, we investigate multi-layer code-based distributed data storage systems that can achieve inherit content confidentiality and optimal storage efficiency. Our comprehensive shows that the optimal code can improve the reliable data storage by nearly 50% comparing to the existing state-of-art research.**

***Index Terms*—Distributed data storage, regeneration code, optimal storage, adversarial networks**

I. Introduction

In just a few years, the Internet of Things (IoT) has evolved into almost every aspect of our daily life. As a result, a gigantic amount of data is being generated every day, which makes data storage a critical infrastructure. Currently, the majority of various data is stored in just a few large central cloud data providers such as Amazon AWS. These providers can get full access to user data and dictate the pricing of their services. Users have to trust the central cloud providers for their data availability and security. To limit the cloud providers from accessing their data, data encryption is the only feasible option, which is not only very costly but also vulnerable to key loss.

To address these issues, distributed storage system (DSS), such as IPFS/FileCoin [1], [2], has been introduced. As the name suggests, distributed data storage works by splitting the data to be stored into multiple blocks and distributing the individual blocks across a decentralized network of storage peer nodes. To ensure data *availability*, the IPFS storage systems generally stores multiple copies of each block, which makes distributed storage much more reliable than the centralized storage. The importance of this architecture is that significant storage space can be allocated without requiring any additional hardware, which can greatly reduce the storage cost.

Unfortunately, the DSS schemes adopted by IPFS/FileCoin have some major limitations. First, the storage efficiency of IPFS is relatively low in that the exact copy of each block has to be collected in order to fully recover the original file. To ensure data availability, each block has to be stored multiple times. Second, IPFS splits large files into blocks through file fragmentation. The individual blocks carry information directly related to the original file. If not protected, the blocks could leak significant information about the whole file.

To address the aforementioned issues of IPFS while also maintaining the major advantages, we propose a new multi-layer code based distributed storage system. The major contribution of this paper can be summarized as follows:

1) We analyzed the limitations of the existing DSS in storage efficiency and security.
2) We introduced multi-layer code based reliable data storage schemes and compared our approach with the existing DSS schemes in storage efficiency and security.

The remainder of this paper is organized as follows: In Section II, various issues related to distributed data storage are discussed. A comprehensive discussion of the multi-layer code is presented in Section III. In Section IV, we provide simulation results and the conclusion in Section V.

II. Discussion of Distributed Data Storage

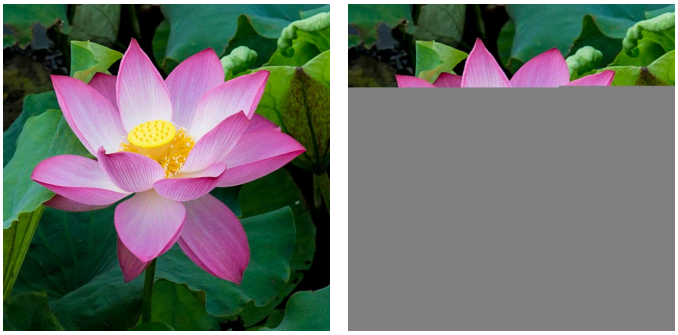
A. Security Issues of Centralized Storage

Centralized cloud provides data storage as a service. Users do not need to build a hardware infrastructure and invest a formidable amount of money to store their data and ensure data fault-tolerance or redundancy. However, building a successful cloud data storage business requires a global network of data-centers and robust user interfaces that satisfy many user demands. As a result, only a few massive corporations owning nearly all of the global cloud data storage market.

Since its inception cloud data storage has evolved to be functional but leaves many security concerns and performance issues unaddressed. First, centralized databases are highly dependent on network connectivity and the Internet speed. For data stored far away from the end-user, the database access time can be very long especially when the Internet speed is slow. In fact, the centralized databases could also become a bottleneck as a result of high traffic. Second, users must rely on these large corporations to ensure their data availability and data security. However, due to economic reasons, storage becomes a commodity or a utility for the data providers. The service providers are incentivized to lock in their customers and extract a premium. The nature of these issues could potentially conflict with user security needs.

B. Limitations of IPFS

The decentralized storage for IPFS system has major security and efficiency issues. We use the following example to illustrate the security limitation.



(a) Original picture (b) First segment of the picture

Fig. 1: File storage in IPFS.

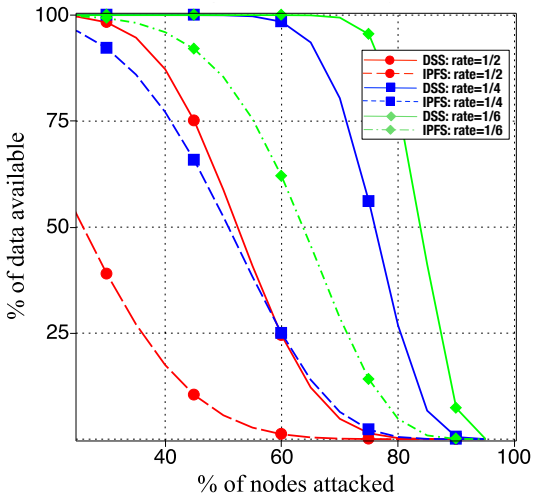


Fig. 2: Data availability comparison between DSS and IPFS.

Example 1. *To store a picture of a lotus flower in IPFS. The file size is 1,034,141 bytes with hash value*

`PQmaNMtwsCJKVmUEe9eaSesKAKhigDzrHf3J4fbAE3VFsAh.`

IPFS splits the file into blocks through file fragmentation. The default IPFS block size is $2^{18} + 14 = 262158$ *bytes. In this case, the hash values of the 4 blocks are:*

```
QmYmpjVoCXD5hiVz8PTDXvnqpBPNiiBQeDP3XfXh1iri9b 262158
QmZYgQBkctTKNRWQoUEzJ7NxwJjoGvYim9Zj9D61jKrHgU 262158
QmXbTiNaNL5AtsG8iRtBQgzUwSGfX4JkuoWAnhyBLfDXYV 262158
QmQuErMLNnXvt7SUwXkgRzQun6PYJGDGEiv2ag342L1Tdq 247723
```

Fig. 1 shows the original lotus flower and the first segment of the flower stored in IPFS. From this figure, we can see that the first segment of the lotus flower clearly reveals a significant amount of information about the original flower. This example makes it clear that the current IPFS file storage system cannot ensure content confidentiality.

In addition to security limitations, the data storage efficiency of IPFS is also quite low. As described before, the data blocks in IPFS are generated through file fragmentation based approaches. While the individual blocks are stored in a decentralized way, it is fundamentally different from the general distributed storage system (DSS) in that the blocks of DSS are generated through algebraic encoding. For IPFS based decentralized storage, to recover the original file, at least one copy of each block must be collected. To increase data availability, IPFS has to store multiple copies of each block. While for the encoding-based (n, k) DSS, the file to be stored is first *algebraically* split into n blocks so that the original file can be recovered from *any* $k \leq n$ blocks. However, the original file remains *information-theoretically secure* for anyone who can access even *up to any* $k-1$ *blocks and with unlimited computing power*. Therefore, no encryption or key management is required to ensure data confidentiality of the file stored. To increase data availability, we only need to increase the number of n, which makes is much more efficient than the IPFS based decentralized data storage. Fig. 2 shows the dramatic data availability differences for three different storage efficiencies of these two approaches.

C. Error and Erasure Correcting Codes

Reed–Solomon (RS) code is a class of maximum distance separable (MDS) code with a set of parameters $[n, k, \delta = n-k+1]$ that operates on a block of n data symbols over a finite-field $\mathbb{F}_q$ (i.e., $\Sigma = \mathbb{F}_q$), where q is a prime number or some power of a prime number. A Reed–Solomon code can locate and correct up to and including $\lfloor(\delta-1)/2\rfloor$ erroneous symbols at unknown locations. As an erasure code, it can correct twice as many erasures as errors at locations that are known and any combination of errors and erasures as long as the relation $2t + s \leq \delta - 1$ is satisfied, where t is the number of errors and s is the number of erasures in the block.

D. Distributed Storage System

In DSSs, the file to be stored is split into blocks and distributed across multiple network peer nodes. Unlike traditional hardware-based data storage, DSS outsources the data storage to a number of P2P storage servers that act as a single storage unit while the data is distributed amid the specific number of servers. DSS is being viewed as a more advanced form of the concept of software-defined storage (SDS).

Compared to the cloud based centralized storage system, the DSS can achieve great security inherit from the DSS design since the data being stored are decomposed into blocks and then spread over many different peer nodes.

In DSS, data replication and erasure coding are two widely employed data redundancy techniques. While replication has the advantage of simplicity and low access overhead, it imposes higher repair traffic and storage overheads. Moreover, the data stored in this system is essentially similar to the IPFS system described in Section II-B. The data components directly contain information of the original files. Therefore, to conceal the contents of the files, data encryption has to be applied to all the data components, which will impose not only the computational cost but also the overhead for secure key management. Conversely, to reduce storage overhead, erasure code avoids data replication. Even though it requires some very limited computational costs due to coding/decoding

operations, compared to the gain in data availability and inherit data content security, it can be fully justified.

The best known DSS schemes are designed based on the conventional (n, k) Reed-Solomon (RS) error-correction codes (such as OceanStore [3] and Total Recall [4]). When a node fails, a replacement node can be regenerated by connecting to any k benign nodes, which will first recover the whole file, and then regenerate the failure node. This approach is a waste of bandwidth because the whole file has to be downloaded to recover even a small fraction of it.

E. Regenerating Code Based DSS

To improve bandwidth efficiency in repairing node failure, Dimakis *et al*. [5] introduced the concept of $\{n, k, d, \alpha, \beta, B\}$ linear regenerating code over the finite field based on network coding. A file of size B is stored in n storage nodes, each of which stores α symbols. A data collector (DC) can reconstruct the whole original file stored in the network by downloading α symbols from each of k randomly selected storage nodes. In the context of regenerating code, a replacement node of a failed node can be regenerated by downloading $\beta \leq \alpha$ symbols from each of $d \geq k$ randomly selected storage nodes. Therefore, the total bandwidth required to regenerate a failed node is $\gamma = d\beta$, which could be far less than the whole file B.

For n storage peer nodes, the size of the file to be stored in the distributed storage network is B (symbols). Based on a cut-set bound on network-coding, the following theoretical bound was derived in [5]:

$$B \leq \sum_{i=0}^{k-1} \min\{\alpha, (d-i)\beta\}. \tag{1}$$

In equation (1), there is a trade-off between the choices of the parameters α and β, which corresponds to the *minimum storage regeneration (MSR)*, where the storage parameter α is minimized as follows:

$$(\alpha_{msr}, \gamma_{msr}) = \left(\frac{B}{k}, \frac{Bd}{k(d-k+1)}\right), \tag{2}$$

and the *minimum bandwidth regeneration (MBR)*, where the regeneration bandwidth γ is minimized as follows:

$$(\alpha_{mbr}, \gamma_{mbr}) = \left(\frac{2Bd}{2kd-k^2+k}, \frac{2Bd}{2kd-k^2+k}\right). \tag{3}$$

Existing DSS was largely constructed based on the maximum distance separable (MDS) code [5], such as RS code. The error correction and node regenerating capability is *limited to* the theoretical error correction bound of the MDS codes.

Compared to MDS code-based schemes, *regenerating code* can reach the *optimal* theoretical trade-off between the minimum *storage* regenerating and the minimum *bandwidth* regenerating. Moreover, in regeneration code, it is no longer needed to decode the original file in order to regenerate a failed peer node.

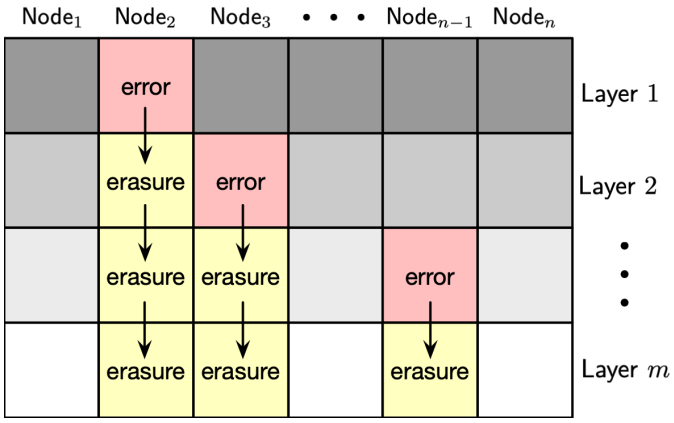


Fig. 3: Multi-layer RS code design.

F. Adversarial Model

In P2P networks, it is possible that the storage nodes may be attacked by the adversarial security attacks. The affected nodes may provide incorrect response to disrupt data reconstruction and regeneration, or becomes totally unavailable.

We assume that the attacker has knowledge of the data allocation scheme. It observes the randomized allocation with an intention to disrupt or manipulate as many peer nodes as possible for any given budget to thwart data reconstruction and availability. Based on this information, the defender first generates n data components from B symbols to be stored in the P2P network using the regenerating code-based approach. It then randomly allocates the n data components across the entire P2P network so that the original B symbols can be recovered when any k untackled components are collected. Meanwhile, any spurious node can be repaired by collecting β symbols from d storage nodes.

III. Multi-Layer Code Design

In this section, we introduce our proposed multi-layer regenerating code for distributed storage system and derive the *optimal* code construction. Since MSR and MBR are similar in description, we will only present the MSR code scenario.

A. Multi-layer Codes and DSS

Multi-layer codes, or m-layer codes, split the data to be stored into m layers and n blocks and then distribute the blocks to $n_0 \leq n$ different storage peer nodes, as shown in Fig. 3. The m layers correspond to m RS codes. The code rate of the m-layer code increases from layer 1 to layer m. On the other hand, the error-correction capability decreases from layer 1 to layer m. Because of this structure, the error correction and node repairing of the m-layer code will proceed from the 1^{st} layer and then move towards the m^{th} layer, layer by layer. Since each node stores data corresponds to all m layers, the nodes with errors detected in one layer, all the subsequent insiders layers stored in that node also becomes unreliable. Therefore, for the sake of computational efficiency, they will be treated as erasures, as illustrated in Fig. 3.

Let the parameter for the i^{th} layer MSR code be viewed as an $(n-1, d_i, n-d_i)$ MDS code, $i = 1, 2, \cdots, m$, where

each component contains β symbols as described in the regeneration code setting, and $d_i \leq d_j, \forall 1 \leq i \leq j \leq m$. Then the first layer code can detect and repair t_1 failure nodes:

$$t_1 = \lfloor (n - d_1 - 1)/2 \rfloor = (n - d_1 - \varepsilon_1 - 1)/2, \quad (4)$$

where $\varepsilon_1 = 0$ if $n - d_1 - 1$ is even and $\varepsilon_1 = 1$ otherwise.

For $i = 2, \ldots, m$, by treating the symbols from the t_{i-1} nodes where errors have been found as erasures, then based on discussion in Section II-C, the i^{th} layer code can detect and repair t_i failure nodes:

$$\begin{aligned} t_i &= \lfloor (n - d_i - 1 - t_{i-1})/2 \rfloor + t_{i-1} \\ &= \left(\sum_{j=1}^{i} 2^{j-1}(n - d_j - \varepsilon_j - 1) \right) /2^i, \end{aligned} \quad (5)$$

assuming that $n - d_i - 1 \geq t_{i-1}$, where $\varepsilon_i = 0$ if $n - d_i - 1 - t_{i-1}$ is even and $\varepsilon_i = 1$ otherwise.

B. Multi-layer Code Construction

The multi-layer DSS was first explored from Hermitian code [6]–[8]. A Hermitian curve

$$\mathcal{H}(q) \colon y^q + y = x^{q+1}$$

is defined over $\mathbb{F}_{q^2}$. There are q^3 points on this curve, denoted as: $P_{i,j} = (\alpha^{i-1}, \alpha^{(i-1)(q+1)+1} + \beta_j)$ $(i = 0, 1, ..., q^2 - 1, j = 0, ..., q-1)$, where α is the primitive element over $\mathbb{F}_{q^2}$ with $\alpha^{-1} = 0$ and β_j is the solution to $y^q + y = 0$.

Let $\mathcal{G}_j = \{(y^j f_j)(P_{0,0}), \cdots, (y^j f_j)(P_{0,q-1}), \cdots, (y^j f_j)(P_{q^2-1,0}), \cdots, (y^j f_j)(P_{q^2-1,q-1})\}$, $j = 0, \ldots, q-1$, be an RS-code, where f_j is a polynomial satisfying

$$\deg f_j < \kappa(j) = \max\{t \mid tq + j(q-1) \leq m\} + 1.$$

According to [6], we have

$$\mathcal{H}_m = \mathcal{G}_0 \oplus \mathcal{G}_1 \oplus \cdots \oplus \mathcal{G}_{q-1}.$$

Alternatively, it can be viewed as a multiple layer RS code, with code parameters:

$$(q^2 - 1, \kappa(i), q^2 - \kappa(i)), \ i = 0, 1, \cdots, q-1. \quad (6)$$

For a Hermitian code $\mathcal{H}_m$ over $\mathbb{F}_{q^2}$, a message matrix $M_{\dim(\mathcal{H}_m) \times A} = [M_1, \cdots, M_A]$ is encoded columnwise. The codeword matrix is

$$\mathcal{H}_m(M) = [\mathcal{H}_m(M_1), \mathcal{H}_m(M_2), \cdots, \mathcal{H}_m(M_A)],$$

where $\mathcal{H}_m(M_i)$ has the following form ($\varrho \in L(mQ)$):

$$[\varrho(P_{0,0}), \cdots, \varrho(P_{0,q-1}), \cdots, \varrho(P_{q^2-1,0}), \cdots, \varrho(P_{q^2-1,q-1})]^T.$$

For Hermitian code based MSR (H-MSR) construction, let $\alpha_0, \ldots, \alpha_{q-1}$ be a strictly decreasing integer sequence satisfying $0 < \alpha_i \leq k(i)$, $0 \leq i \leq q-1$, and $A = \mathsf{lcm}(\alpha_0, \cdots, \alpha_{q-1})$. Arrange $B = A \cdot \sum_{i=0}^{q-1} (\alpha_i + 1)$ symbols into two matrices S, T as follows:

$$S = \begin{bmatrix} S_0 \\ \vdots \\ S_{q-1} \end{bmatrix}, \quad T = \begin{bmatrix} T_0 \\ \vdots \\ T_{q-1} \end{bmatrix},$$

where

$$\begin{aligned} S_i &= [S_{\alpha_i,1} \ \ S_{\alpha_i,2} \ \ ...S_{\alpha_i,A/\alpha_i}], \\ T_i &= [T_{\alpha_i,1} \ \ T_{\alpha_i,2} \ \ ...T_{\alpha_i,A/\alpha_i}], \end{aligned}$$

$S_{\alpha_i,j}, T_{\alpha_i,j}$ $(0 \leq i \leq q-1, 1 \leq j \leq A/\alpha_i)$ are symmetric matrices of size $\alpha_i \times \alpha_i$. Then by processing the data symbols as described below to achieve the MSR point in the distributed storage [7], [8]:

1) Encode the data matrices S, T defined above using a Hermitian code $\mathcal{H}_m$ over $GF(q^2)$ with parameters $\kappa(j)$ $(0 \leq j \leq q-1))$ and m $(m \geq q^2 - 1)$.
2) Calculate the $q^3 \times A$ codeword matrix
$$Y = \mathcal{H}_m(S) + \Gamma \mathcal{H}_m(T),$$
where $\Gamma = \mathsf{diag}(\Lambda_0, \Lambda_1, \cdots, \Lambda_{q^2-1})$ and the q diagonal elements in Λ_i are all identical to $\lambda_i \in \mathbb{F}_{q^2}$.
3) Divide Y into q^2 submatrices $Y_0, \cdots, Y_{q^2-1}$ of the size $q \times A$. Store the submatrix in (up to) q^2 storage nodes.

Since H-MSR code can be viewed as q layers of RS codes with parameters

$$(q^2 - 1, d_i, q^2 - d_i), \ i = 0, 1, \cdots, q-1, \alpha_l \leq \kappa(l), \quad (7)$$

where $d_i = 2\alpha_i, \alpha \leq \kappa(i)$, we can choose the sequence α_i to be strictly decreasing so that d_i is also strictly decreasing. For the q RS codes, the minimum distance of the $(q^2 - 1, d_{q-1}, q^2 - d_{q-1})$ code is the largest. It can correct up to τ_{q-1} errors:

$$\tau_{q-1} = \lfloor (q^2 - d_{q-1} - 1)/2 \rfloor,$$

where $\lfloor x \rfloor$ is the floor function of x.

Next, the code $(q^2 - 1, d_l, q_2 - d_l), l = q-2, \cdots, 0$ will be decoded sequentially, which can correct at least $\tau_l = \tau_{q-1}$ errors when $q^2 - d_0 - 1 \geq \tau_{q-1}$. Therefore, the total number of errors that the the H-MSR can *detect* and *repair* is

$$\tau_{H-MSR} = q\tau_{q-1} = q \left\lfloor (q^2 - d_{q-1} - 1)/2 \right\rfloor.$$

In comparison, for RS-MSR code with the same rate can repair

$$\tau_{RS-MSR} = \left\lfloor \left(q^3 - q - \sum_{l=0}^{q-1} d_l \right) /2 \right\rfloor. \quad (8)$$

Moreover, the proposed H-MSR code has complexity $\mathcal{O}(n^{5/3})$ for node regeneration and reconstruction, which is lower than the RS-MSR complexity $\mathcal{O}(n^2)$ in both scenarios.

C. Optimal Regenerating Code

Motivated by the Hermitian code structure, a natural research task is to find out how to select the layers that can optimize the overall performance [9].

Similar to message encoding algorithm, to encode a file with size B, we select $d = 2\alpha$ and divide the file into θ blocks of data with size $\widetilde{B}$, where $\theta = \lceil B/\widetilde{B} \rceil$. Then the θ blocks of data will be encoded into codeword matrices $\mathsf{F}_1, \ldots, \mathsf{F}_\theta$ and form the final $n \times \alpha\theta$ codeword matrix: $\mathcal{C} = [\mathsf{F}_1, \ldots, \mathsf{F}_\theta]$. Each row $\mathbf{c}_i = [\mathbf{F}_{1,i}, \ldots, \mathbf{F}_{\theta,i}]$, $1 \leq i \leq n$, of the codeword

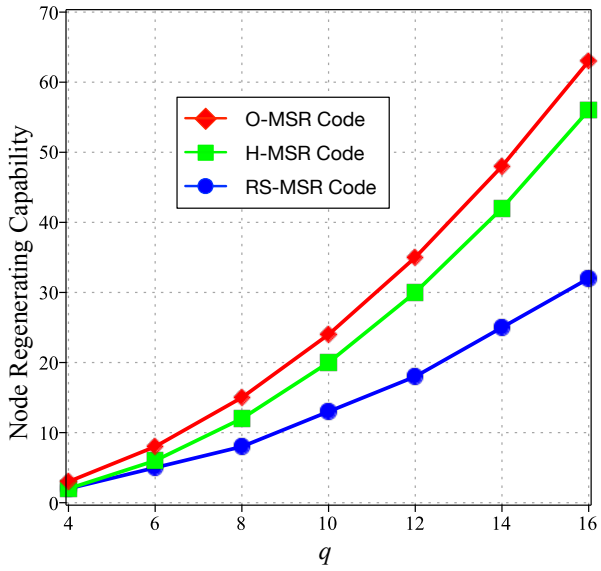


Fig. 4: Comparison of node regenerating capability of the m-layer code with the H-MSR and RS-MSR codes with code rate=3/4 and $m = q$.

matrix $\mathcal{C}$ will be stored in storage node i, where $\mathbf{f}_{j,i}$ is the i^{th} row of F_j, $1 \le j \le \theta$.

The design of optimal multi-layer code is equivalent to maximizing the number of failure nodes that can be regenerated, which is determined by t_m. The problem can be formulated as an optimization problem given the following algorithm [9].

$$\begin{aligned}
\text{maximize: } & t_m, \text{ where } t_i = \left\lfloor \frac{n - d_i - t_{i-1}}{2} \right\rfloor + t_{i-1}, \\
& i = 2, \ldots, m, t_0 = 0 \\
\text{constraints: } & \sum_{i=1}^{m} d_i = d. \\
& d_{i-1} \le d_i, \ 2 \le i \le m, \\
& n - d_i - t_{i-1} - 1 \ge 0, \ i = 2, \ldots, m \\
& n + d_i - 2d_{i+1}, \ge 0, \ i = 1, \ldots, m-1,
\end{aligned}$$

This is a linear optimization. By introducing slack variables ε_i, $i = 1, 2, \cdots, m$, we can easily find that the *optimal regenerating code* can be achieved when $d_i = \mathsf{round}(d/m) = \tilde{d}$ as shown in [9]. In this case, as shown in equation (5), the maximum number of failure nodes that the m-layer optimal regenerating code (O-MSR) can repair is at most

$$\begin{aligned}
t_m &= \left(\sum_{i=1}^{m} 2^{i-1} (n - \tilde{d} - 1 - \varepsilon_i) \right) / 2^m, \\
&= \left(1 - \frac{1}{2^m} \right) (n - \tilde{d} - 1) - \frac{1}{2^m} \sum_{i=1}^{m} 2^{i-1} \varepsilon_i.
\end{aligned} \tag{9}$$

By carefully selecting the d so that $n - \tilde{d} - 1$ is even, then we have $\varepsilon_i = 0$ for $i = 1, \ldots, m$, which can maximize the error correction capability for the given storage efficiency. In this case, from equation (9), we have

$$\begin{aligned}
\lim_{m \to \infty} t_m &= \lim_{m \to \infty} \left(\sum_{i=1}^{m} 2^{i-1} (n - \tilde{d} - \varepsilon_i - 1) \right) / 2^m \\
&= n - \tilde{d} - 1.
\end{aligned} \tag{10}$$

In this case, for the single layer RS code, we have

$$\begin{aligned}
t_{\mathsf{RS-MSR}} &= \lfloor (n - \tilde{d} - 1)/2 \rfloor \\
&= (n - \tilde{d} - 1)/2.
\end{aligned} \tag{11}$$

Therefore, we have the following theorem.

Theorem 1. *Let $t^{(m)}_{\mathsf{O-MSR}}$ be the number of nodes that can be regenerated by an m-layer optimal regenerating code and $t_{\mathsf{RS-MSR}}$ be the number of nodes that can be regenerated from a single layer RS-MSR code with comparable parameters, then*

$$\lim_{m \to \infty} t^{(m)}_{\mathsf{O-MSR}} = 2 t_{\mathsf{RS-MSR}}. \tag{12}$$

Theorem 1 proves that the overhead required to correct random node failure for the m-layer O-MSR code approaches one half of the RS-MSR code under comparable security parameters, which is the overhead required to correct erasures.

Fig. 4 compares the performance of our proposed multi-layer codes O-MSR and H-MSR codes with the RS-MSR code in which we select $m = q$ since H-MSR is only defined for this case.

IV. Conclusion

We first present limitations of the existing storage system, especially the decentralized storage system. We then analyze our previous work in multi-layer code design and applications to distributed storage systems. We also derive the theoretical performance bound of the optimal multi-layer regenerating code. Finally, we present simulation results to compare the performance of the proposed m-layer O-MSR code and the RS-MSR code.

References

[1] J. Benet. IPFS - content addressed, versioned, P2P file system (draft 3). [Online]. Available: https://github.com/ipfs/ipfs/blob/master/papers/ipfs-cap2pfs/ipfs-p2p-file-system.pdf

[2] Protocol Labs. (August 14, 2017) Filecoin: A decentralized storage network. [Online]. Available: https://filecoin.io/filecoin.pdf

[3] S. Rhea, C. Wells, P. Eaton, D. Geels, B. Zhao, H. Weatherspoon, and J. Kubiatowicz, "Maintenance-free global data storage," *IEEE Internet Computing*, vol. 5, pp. 40 – 49, 2001.

[4] R. Bhagwan, K. Tati, Y.-C. Cheng, S. Savage, and G. M. Voelker, "Total recall: System support for automated availability management," in *roc. Symp. Netw. Syst. Design Implementation*, 2004, pp. 337–350.

[5] A. Dimakis, P. Godfrey, Y. Wu, M. Wainwright, and K. Ramchandran, "Network coding for distributed storage systems," *IEEE Transactions on Information Theory*, vol. 56, pp. 4539 – 4551, 2010.

[6] J. Ren, "On the structure of Hermitian codes and decoding for burst errors," *IEEE Transactions on Information Theory*, vol. 50, no. 11, pp. 2850–2854, November 2004.

[7] J. Li, T. Li, and J. Ren, "Beyond the mds bound in distributed cloud storage," in *IEEE INFOCOM 2014*, Toronto, CA., April 27-May 2 2014, (Acceptance Rate: 320/1645=19.4%).

[8] ——, "Beyond the mds bound in distributed storage," *IEEE Transactions on Information Theory*, vol. 66, no. 7, pp. 3957–3975, 2020.

[9] ——, "Optimal construction of regenerating code through rate-matching in hostile networks," *IEEE Transactions on Information Theory*, vol. 63, no. 7, pp. 4414–4429, July 2017.