A survey on regenerating codes

V. Anto Vins [*], S.Umamageswari [**], P.Saranya [**]

[*] P.G Scholar, Department of CSE, Prathyusha Institute of Technology and Management, Tiruvallur- 602025
[**] Associate Professor, Department of CSE, Prathyusha Institute of Technology and Management, Tiruvallur- 602025

Abstract- Cloud computing refers to the various remote systems interconnected through the internet. We can outsource data, software, infrastructure from the cloud environments. Once we outsource storage there is a possibility for the data for the data to get corrupted or be hacked. The general solution is to stripe the data across multiple servers. In the setting the repair traffic will be high and the fault tolerance will be low. In order to improve the storage efficiency and the repair traffic regenerating codes have been use. In this paper we will discuss about the various regenerating codes that have evolved from time to time.

Index Terms- Erasure coding, regenerating codes, cloud storage, I/O overhead.

I. Introduction

Due to the decreased storage pricing and the decreasing bandwidth, leading IT firms implement software as a Service(SaaS), Infrastructure as a service(IaaS) inside their data centers. This new paradigm of providing services and infrastructure is called cloud computing. The volume of data generated and stored by the cloud users is rapidly increasing exponentially everyday. In order to maintain the massive volume of data being stored, cloud storage systems must be scalable. In this case, it becomes a major challenge for cloud storage systems to maintain data integrity due to large volume of data and the large number of disks used. Even though the number of disk failures is small within a data center, it may increase due to the addition of the large volumes of disks. Moreover other systems in the data center may cause outages of data, making data unavailable or even get lost.

To guarantee the integrity of data stored in the cloud, it is natural to store replicas in multiple cases, such that data loses can be managed when there is atleast one replica running. However, replicas can significantly reduce the storage efficiency. For example, if data are stored with 3-way replication, the effective storage space can be no better than 1/3 of the total consumed storage space. If the design objective is to maximize the storage efficiency without sacrificing the ability to tolerate disk failures, the storage system should store data encoded by erasure coding. Before the emergence of cloud computing, erasure coding has long been proposed to detect or correct errors in storage or communication systems. For example, RAID 6 can compensate for at most 2 disk failures by parity coding, whose storage efficiency is at most 1 – (2/n) where n is the total number of disks. Reed-Solomon codes can provide even more flexibility such that any number of disk failures under a certain threshold can be tolerated. Wual [1], an online secure storage service, uses Reed-Solomon codes to ensure data integrity, by encoding data with Reed-Solomon codes after encryption.

However, two main drawbacks conventionally prevent erasure coding from being practical and popular in cloud storage. First, to read or write data, the system needs to encode or decode data, leading to a high access latency and a low access throughput due to the CPU limit. Second, though erasure coding stores data as multiple coded blocks, when one coded block gets lost, the system must access multiple coded blocks that are sufficient to recover all the data. This overhead makes the repair with erasure coding very expensive in terms of both bandwidth and disk I/O overhead. Unfortunately, applications hosted in the cloud are sensitive to the disk I/O performance, and bandwidth is always a limited resource inside the data center since most data centers introduce link oversubscription[2].

In this paper, we examine recent advances of these coding techniques. Specifically, we introduce coding techniques that optimize the overhead of data repair in the objective of minimizing the bandwidth consumption and increasing the storage efficiency. Due to the high frequency of data repairs and the high repair overhead of conventional erasure coding, optimizations in both of these two axes fall into the considerations of data repair.

For bandwidth consumption, Dimakis et al[3] proposed a family of regenerating codes. Based on a model inspired by network coding, the repairs of coded data can be modeled into an information flow graph, with a constraint that maintains the ability to tolerate disk failures. Based on this model, an optimal tradeoff between storage and bandwidth can be derived, such that given the amount of data stored on the disk, we can obtain an optimal lower bound of the amount of data that should be transferred during the repair. In the tradeoff curve, two extreme points attract much more attention than interior points, corresponding to the minimum storage cost and the minimum bandwidth cost, respectively. To achieve these two extreme points, many papers proposed instances of regenerating codes that either construct randomized codes that maintain the tolerance against disk failures with a high probability, or construct deterministic codes using interference alignment. With the help of deterministic coding, lost data can be repaired exactly, suggesting that we can construct systematic codes to achieve an access latency as low as replicas. Moreover, cooperative regenerating codes make it possible to repair from multiple disk failures with an even lower bound of bandwidth consumption, with the help of cooperation among participating servers. In this paper, we examine the various available regenerating codes and their significance.

II. Erasure Coding and Its Performance Metrics

A. *Erasure codes in Storage Systems*

In a storage system where data are stored on a large number of commodity disks, it is clear that disk failures cannot be considered as just exceptions, but as a rule. Therefore, the storage system has to store redundancy such that when a certain number of disks lose data, data can still be accessible from other disks. For example, the N -way replication, which stores N replicas in N different disks, is able to tolerate at most N - 1 disk failures. Figure 1a illustrates an example of 3-way replication, where the original data are disseminated into 3 different disks and any one disk is able to repair or access the original data. However, N -way replication can only achieve a storage efficiency of 1\N at best. Erasure coding, on the other hand, is able to tolerate the same number of disk failures, yet with a much better storage efficiency. Among erasure coding, Maximum Distance Separable (MDS) codes (e.g., Reed-Solomon codes[4]) achieve the optimal storage efficiency. Suppose that in the storage system, data are organized in the unit of data object, which may correspond to a file or a fix-sized block in different storage systems. Assume that a data object can be stored onto n disks. Given an arbitrary number of k, where k < n, (n, k) MDS codes can guarantee to

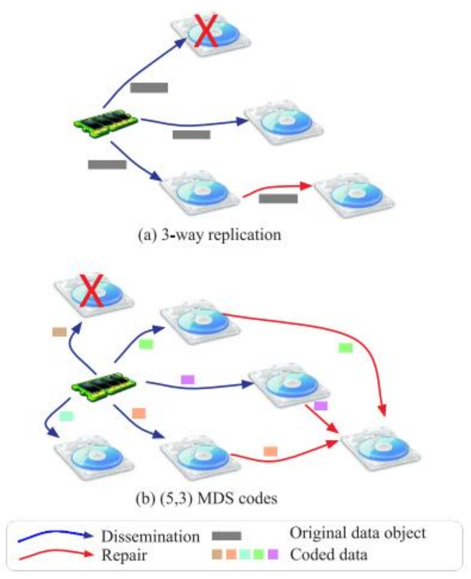

Fig. 1 Comparison of 3-way replication and (5,3) MDS codes. In 3-way replication, the original data object is replicated into 3 different disks, such that any bit of data can be obtained from any one disk. In (5,3) MDS codes, data are encoded into smaller blocks in which any three can be decoded into the original data.

tolerate at most n - k disk failures, i.e., k disks are sufficient to access any bit of the original data. Specifically, the data object is encoded into n coded blocks and are uniformly disseminated into the n disks. Suppose the size of the data object is M bits, the size of each coded block should be M\k bits, if we do not consider the metadata of the data object. In this sense, the storage efficiency of MDS codes is at best k\n. Compared to the 3-way replication as shown in Fig. 1b, (5; 3)MDS codes can still tolerate at most 2 disk failures, while improving the storage efficiency by 80%. To access the data object, the system needs to access k different coded blocks (from k different disks) and recover the original data object by the decoding algorithm of the corresponding MDS codes. Apparently, the decoding algorithm incurs an additional access latency. To improve the access latency, the storage system can use a cache to store one replica of the original data object as well[5]. Recovering the whole data object for data access may sound reasonable in most cases. However, from the point of view of the storage system, it is totally unnecessary to recover the whole data object if we only need to repair one lost coded block, as what we need is just a very small portion of the data object. Unfortunately, before the appearance of regenerating codes, all MDS codes, to our best knowledge, require to access at least k disks to repair even only one disk, while in the case of replication, to repair one replica we only need to transfer one replica. This requirement can dramatically increase bandwidth overhead in a data center, and significantly affect the performance of the storage system and other applications hosted in the cloud.

B. *Performance Metrics*

It has been made clear that it is not enough to consider only the storage efficiency and the tolerance against disk failures in cloud storage systems. The performance metrics that should be considered when designing erasure coding for cloud storage should also include:

- *Repair Bandwidth:* To repair failed disk, data stored on that disk should be repaired in a replacement disk. The server with the replacement disk, called a newcomer, needs to retrieve data from some existing disks. If the servers that host these existing disks, called providers, send out the raw data of their coded blocks, the bandwidth used to transmit the existing coded blocks equals the size of these coded blocks and then the newcomer encodes the received data by itself to generate the lost data. However, if encoding operations can be performed both on the newcomer and providers rather than on the newcomer only, a much smaller amount of data can be transferred.

In the pioneering paper of Dimakis et al[3], a surprising and promising result was shown that the bandwidth used in the repair can be approximately as low as the size of the repaired block by encoding operations of providers and the family of erasure codes that achieves the optimal bandwidth consumption during the repair was called regenerating codes.

- *Repair I/O*. Besides bandwidth, another performance metric in the repair for erasure coding is disk I/O at the participating servers. Specifically, the writing operations are performed only at the newcomer, and the amount of data written should equal the size of the coded block. As the writing operations are unavoidable, what we really care is actually the amount of data read from disks of providers. Similar to the bandwidth consumption, conventional erasure codes will ask for providers to read k blocks in total just to repair one block.
- *Access latency*. Due to the decoding operations, the access latency of data encoded by erasure coding has to be much larger than replicas. However, systematic codes, in which the original data can be embedded into code blocks, are able to maintain a higher storage efficiency than replicas while achieving a low access latency, since we now only need to access the systematic part with no decoding operations. This

attractive property of systematic codes comes with a high price that makes the repair much more complex than non-systematic codes, because the repaired data should be exactly the same as the lost data, at least for the embedded original data. Therefore, the deterministic construction of codes must be considered when designing systematic erasure codes for cloud storage.

- *Storage efficiency*. The storage efficiency denotes the ratio of the amount of the original data to the actual amount of data stored on disks. In other words, given the same amount of data, with a higher storage efficiency we can use a smaller amount of storage space to tolerate the same number of disk failures. MDS codes achieve the optimal storage efficiency, i.e., given n and k, MDS codes can be constructed such that any k among all n coded blocks are sufficient to recover the original data.)

III. Varieties of Regenerating Codes

A. *Information Flow Graph*

To investigate the bandwidth consumption in repairs with erasure coding, Dimakis et al.[3] proposed to use the information flow graph, which is a tool used in the analysis of network coding, as a model to characterize the tradeoff between storage and bandwidth.

As shown in Fig. 2, in the information flow graph, all servers can be categorized as the source, storage nodes, and the data collector. The source denotes the server where the data object is originated. Suppose that the size of the data object is M bits. After encoding, coded blocks of α bits are disseminated into n storage nodes. Particularly, the source is represented by a vertex and a storage node is represented by two vertices in the information flow graph. The weight of the edge corresponds to the amount of data stored in the storage node. Thus, after the dissemination, all n storage nodes store α bits and any k of them suffice to recover the original data object, suggesting that k$\alpha \leq$ M A virtual vertex called data collector is able to connect to any k storage nodes to recover the original data object. When a storage node fails, a newcomer does not just connect to k available storage nodes, but d storage nodes as providers (d $\leq$ k). Different from

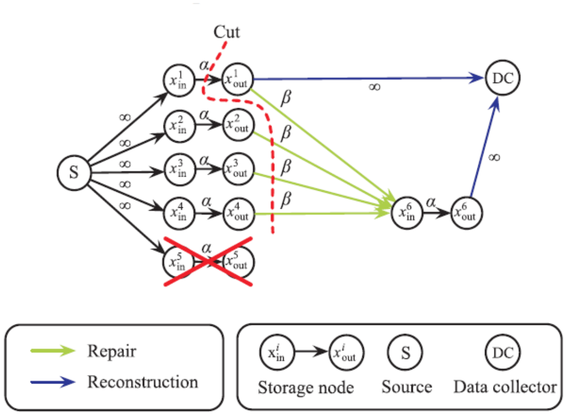

Fig 2: An illustration of the information flow graph that corresponds to (4,2) MDS codes. This figure was firstly shown in Ref. [9].

conventional MDS codes, the newcomer can receive β bits from each provider, $\beta \leq \alpha$ represented by the weight of the edge between the provider and the newcomer. After receiving a total of $r = d\beta$ bits from providers, the newcomer stores α bits and becomes a new storage node. It is required that after any rounds of repairs, the MDS property that any k coded blocks suffice to recover the original data object always holds. In other words, the data collector can connect to not only the storage nodes that receive data directly from the source, but the storage nodes repaired from newcomers in repairs as well. Besides, the weights of edges attached to the source or the data collector are all set to be infinity.

Therefore, the repair problem in the storage system can be interpreted as a multicast problem in the information flow graph, where the source is multicasting data to all possible data collectors. It is well known in the multicast problem that the maximum multicast rate equals the minimum cut separating the source from any receivers, and this rate can be achieved through network coding[6]. Thus, it can be proved that the MDS property holds after any rounds of repairs, if the min-cut in the information flow graph is no less that the size of the original data object. By calculating the minimum min-cut in the information flow graph, given d and β we can derive the minimum value of α, and then we obtain the tradeoff curve between α and the total bandwidth consumption r.

B. *Minimum-storage Regenerating codes and minimum-bandwidth regenerating codes*

The codes that achieve that the min-cut in the information flow graph equals the amount of the data object is called regenerating codes. Given the tradeoff between storage α and bandwidth r , two special cases of regenerating codes interest us most, which correspond to the minimum storage space required at storage nodes and the minimum total bandwidth consumption in the repair, respectively.

The regenerating codes that achieve the minimum storage in storage nodes is called Minimum-Storage Regenerating (MSR) codes, where $(\alpha_{MSR}, r_{MSR}) = (\frac{M}{k}, \frac{Md}{k(d-k+1)})$. Notice that now α equals to the size of coded blocks of conventional MDS codes, and thus MSR codes can be regarded as MDS codes. However, since r=$\frac{Md}{k(d-k+1)} \rightarrow \frac{M}{k}$ as $d \rightarrow \infty$, MSR codes consume bandwidth in the repair approximately the same as the amount of one coded block, while conventional MDS codes consume bandwidth that equals the size of k coded blocks. In this sense, MSR codes save a significant amount of bandwidth in the repair.

The other extreme points in the tradeoff between storage and bandwidth is called Minimum-Bandwidth Regenerating (MBR) codes, where $(\alpha_{MBR}, r_{MBR}) = (\frac{2Md}{k(2d-k+1)}, \frac{2Md}{k(2d-k+1)})$MSR codes achieve the minimum bandwidth consumption among regenerating codes. Though MBR codes require more storage than MSR codes, the newcomer only needs to receive exactly the amount of data it needs to repair, and the storage and bandwidth both converge to $\frac{M}{k}$ when d is large enough.

To implement regenerating codes, the simplest way is to use random linear coding[7] which is inspired by network coding. Dividing the original data object into k blocks, a coded block produced by random linear coding is a random linear combination of the k blocks. The encoding operations are performed on a Galois field GF(2^q). As a Galois field size of 2^8 corresponds to a byte in the computer, q is usually set to be an integral times of 8 to make encoding operations convenient. Given any k coded blocks and their coding coefficients, the decoding operations are just solving the corresponding linear system with k unknowns (i.e., original blocks) and k equations. When q is large enough, such as 16 or 32, any k coded blocks can be decoded with a very high probability.

On the other hand, the randomized construction of regenerating codes can suffer from a significant computational complexity, especially when parameters are not properly selected[8]. In additional, by no means the randomized regenerating codes can guarantee to repair data exactly as the lost data. Even worse, even a very large Galois field cannot ensure that any k code blocks are decodable, but only with a very high probability, due to the randomized coefficients. The number of repairs, however, cannot be easily limited, suggesting that after a large number of repairs data integrity cannot be maintained, as the randomness accumulates gradually in coded blocks.

Therefore, it is necessary to find explicit construction of regenerating codes, especially for MSR and MBR codes. Further, it is desirable that the lost block can be repaired exactly given the explicit construction.

C. *Exact Regenerating codes*

The most important tool used to construct exact regenerating codes is interference alignment, which is initially proposed for wireless communication. The basic idea of interference alignment is that the undesired vectors can be eliminated by aligning them onto the same linear subspace. Figure 3 illustrates how interference alignment helps to achieve exact regenerating codes.

We suppose that in Fig. 3 data are encoded by (n=4, k=2, d =3) MSR codes, i.e., any two of the four nodes can recover the original file. In each storage node, each coded block contains two coded segments, such as (A_1,A_2) in the failed storage node. To recover A_1 and A_2 , the newcomer contacts 3 storage nodes as providers and downloads half of a coded block, i.e., a coded segment from each provider to achieve the bandwidth consumption of MSR codes. Notice that each provider owns coded segments containing components of B_1 and B_2 , which are undesirable to the newcomer. To eliminate B_1 and B_2 , each provider can send a segment in which B_1 and B_2 are aligned onto the same linear subspace of $B_1 + B_2$. Clearly,$B_1 + B_2$ can be eliminated as one unknown and A_1 and A_2 can be decoded by solving three equations with three unknowns.

As for exact MBR codes, Rashmi et al.[9] proposed a Product-Matrix construction which is able to explicitly construct (n,k,d) exact MBR codes on a finite field of size n or higher with any choices of n, k, d if k $\leq$ d < n. The Product-Matrix construction produces vector MBR codes such that a coded block contains multiplecoded segments, just like the example shown in Fig. 3. As for exact MSR codes, the choices of parameters

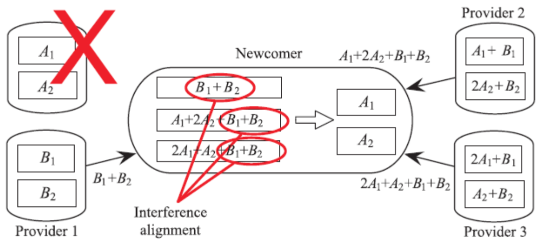

Fig 3: An example of (4,2,3) exact MSR codes. By interference alignment, the loss of (A1,A2) can be repaired exactly since B1 and B2 are eliminated by aligning them onto B1 + B2 in three providers.

become more complicated. Suh and Ramchandran[10] proposed an explicit construction of scalar exact MSR codes where $d \geq 2k-1$, over a finite field of size atleast 2(n-k). Rashmi et al[9] improved the choices of parameters such that d ≥ 2k-2, by constructing exact MSR codes using the Product-Matrix construction, with a larger finite field of size at least n.(d-k+1). In Ref [11], Shah et al. had proved that no scalar exact MSR codes exist when d <2k-3.

Cadambe et al[12]. and Suh and Ramchandran[13] showed that any choices of (n, k, d) could be achieved asymptotically when constructing vector exact MSR codes, as the field size goes to infinity. Constructing regenerating codes on a very large finite field is impractical due to its overwhelming encoding/decoding overhead. Papailiopoulos and Dimakis[14] showed that (n= k +2 , k, d = k+ 1) vector exact MSR codes can be constructed explicitly, combining interference alignment and Hadamard matrices.

Since the exact repair makes much more sense for the systemmatic part than the parity part of exact regenerating codes, we can construct hybrid regenerating codes that only support the exact repair of the systematic part while the parity part is still repaired by the functional repair. Wu[15] constructed the (n,k,d = k+1) hybrid vector MSR codes when 2k ≤ n, where the field size is greater than $2\binom{2n-1}{2n-1}$.Tamo et al[16] and Cadambe et al[17] both proposed (n,k,d = n-1) hybrid MSR codes for arbitrary n and k.

D. *Cooperative regenerating codes*

In some storage systems, such as Total Recall[18] ,in order to prevent unnecessary repairs incurred by temporary node departures, the system repairs failed nodes in batch when the number of failed nodes reaches a certain threshold. Hu et al.[19] first found that if newcomers can cooperate, there exist cooperative regenerating codes that achieve a better tradeoff curve between storage and bandwidth. Still analyzing the min-cut in the information flow graph, Hu et al.[19] showed that $(n, k, d = n - t, t)$ randomized Minimum Storage Cooperative Regenerating (MSCR) codes can achieve the bandwidth consumption of $\frac{M}{k} \cdot \frac{n-1}{n-k}$ with t providers (n-t $\geq k$). Notice that the bandwidth consumption is independent of the number of providers.

A more general result of the bound of bandwidth consumption is shown that per newcomer, $(\alpha_{MSCR}, \beta_{MSCR}) = (\frac{M}{k}, \frac{M}{k}, \frac{d+t-1}{d+t-k})$ [20,21] and $(\alpha_{MSCR}, \beta_{MSCR}) = (\frac{M}{k}, \frac{2d+t-1}{2d+t-k}, \frac{M}{k} \cdot \frac{2d+t-1}{2d+t-k})$ [21]. Shum and Hu[22] proposed an explicit construction of exact $(n, k, d = k, t = n - k)$ MBCR codes. Wang and Zhang[23] showed that as for all possible values of (n,k,d,t), there exist explicit constructions of exact MBCR on a field of size at least n. On the other hand, when $d = k \neq n - t, (n, k, d, t)$ scalar MSCR codes can be constructed[20] . Le Scouarnec[24] discussed the construction of exact MSCR codes with some other choices of parameters when $d \geq k = 2$ .The existence of exact MSCR codes with other values of parameters still remains an open problem.

E. Repair-by-transfer regenerating codes

The regenerating codes we have mentioned above require providers to encode their data to align vectors in a specific linear subspace and the newcomer to encode received data to eliminate undesired components. The arithmetic operations performed on a finite field can be expensive, if the field size is large. Thus, the cloud storage systems will favor the repair-by transfer property that in the repair no arithmetic operations are required at providers. With the repair-by-transfer property, the disk I/O overhead can be minimal since only data needed by the newcomer will be read from providers. Moreover, the storage node then can have no intelligence, such that its functionality can be implemented by hardware with a low cost. Some choices of parameters have been considered to construct the corresponding repair-by-transfer regenerating codes. Shum and Hu[25] and Hu et al.[26\ constructed $(n, k = 2, d = n - 1)$ and $(n, k = n - 2, d = n - 1)$ functional repair-by transfer MSR codes, respectively. On the other hand, $(n, k = n - 2, d = n - 1)$ exact MBR codes can be constructed over a finite field of size 2[27] . The existence of exact regenerating codes remains an open problem as for most other choices of parameters. The only known result is that if $d \geq 2$ and $t \geq 2$, any (n, k, d, t) linear exact MBCR codes cannot achieve the repair-by-transfer property[23].

Nevertheless, Rashmi et al.[28] proposed an intuitive graph-based construction of repair-by-transfer exact MBR codes, where any missing block can only be repaired from its direct neighbors in the graph. Fractional repetition codes are employed in the construction such that direct neighbors share the same segment and the newcomer only needs to receive replicas of the segments from its neighbors to repair itself, where no arithmetic operations are required not only at providers, but at the newcomer as well. It is shown that any $(n, k, d = n - 1)$ MBR codes can be constructed uniquely with this method, n > k. Though repair-by-transfer regenerating codes are able to achieve the minimum disk I/O overhead, only some specific choices of parameters have instances of corresponding regenerating codes so far. Nevertheless, cloud storage systems can have a wide spectrum of parameter choices due to their own requirements. Therefore, it is desirable that erasure coding can achieve low disk I/O overhead with a wide range of parameter choices, even if some properties, such as the MDS property or the storage efficiency, will be sacrificed.

F. FMSR (Functionally minimum storage regenerating code)

FMSR codes belong to maximum distance separable (MDS)codes. An MDS code is defined by the parameters(n,k), where k<n. It encodes a file F of size $|F|$ into n pieces of size $|F|/k$ each. An (n,k) MDS code states that the original file can be reconstructed from any k out of n pieces (i.e., the total size of data required is $|F|$). An extra feature of FMSR codes is that a specific piece can be reconstructed from data of size less than $|F|$. FMSR codes are built on regenerating codes[9], which minimize the repair traffic while preserving the MDS property.

We consider a distributed storage setting in which a file is striped over n servers using an (n,k)-FMSR code. Each server can be a storage site or even a cloud storage provider, and is assumed to be independent of other servers. An (n,k)-FMSR code splits a file of size $|F|$ evenly into k(n-k) native chunks, and encodes them into n(n-k) code chunks,

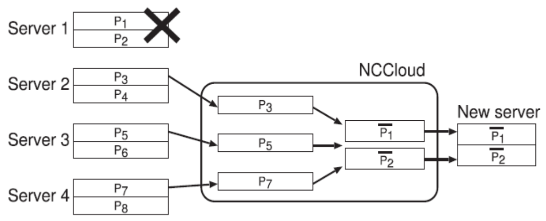

Fig 4:Example of how a file is repaired in (4,2)- FMSR codes. Each of the code chunks P1,….,, P8 is a random linear combination of the native chunks. $\bar{P}1$ and $\overline{P2}$ are distinct random linear combinations of P3,P5 and P7

where each native and code chunk has size $\frac{|F|}{k(n-k)}$. Each code chunk, denoted by Pi (where $1 \leq i \leq n(n-k)$), is constructed by a random linear combination of the native chunks, similar to the idea in [29]. The n(n-k) code chunks are stored in n servers (i.e, n-k code chunks per server), where the k(n-k) code chunks from any k servers can be decoded to reconstruct the original data. Decoding can be done by inverting the encoding matrix[30] . FMSR codes are nonsystematic codes that keep only code chunks rather than native chunks. Nevertheless, FMSR codes are mainly designed for long-term archival applications, where the read frequency is low and each read operation typically restores the entire file.

G. FMSR- DIP (Functionally Minimum Storage Regenerating Code – Data Integrity Protection)

FMSR-DIP codes are implemented atop NCCloud[31] . FMSR-DIP[32] codes operate in units of bytes. The notations for FMSR-DIP codes are given below. For an (n-k) -FMSR code, we define $\{\alpha_{ij}\} 1 \leq i \leq n(n-k), 1 \leq j \leq k(n-k)$ the set of encoding coefficients that encode k(n-k) native chunks $\{F_j\} 1 \leq j \leq k(n-k)$ into n(n-k) code chunks $\{P_i\} 1 \leq i \leq n(n-k)$ Thus, each code chunk Pi is formed by $P_i = \sum_{j=1}^{k(n-k)} \alpha_{ij} F_j$ [32].

All arithmetic operations are performed in the Galois field GF(2^8). However, byte-level operations may make the implementation inefficient in practice, especially for large files. Here, we describe how FMSR-DIP codes can be extended to operate in units of blocks (i.e., a sequence of bytes) to trade security for performance. In the following, we describe the possible tunable parameters that are supported in FMSR-DIP codes. PRP block size. Instead of permuting bytes, we can permute blocks of a tunable size (called the PRP block size). A larger PRP block size increases efficiency, but at the same time decreases security guarantees. PRF block size. In a byte-level PRF operation, we can simply take the first byte of the AES-128 output as the PRF output. In fact, we can also compute a longer PRF and apply the PRF output to a block of bytes of a tunable size (called the PRF block size). To extend the PRF beyond the AES block size (16 bytes), we can pad the nonce with a chain of input blocks of 16 bytes each, and encrypt them using CBC mode. However, setting the PRF block size to larger than 16 bytes shows minimal performance improvement, as AES is invoked once for every 16 bytes of input in CBC mode and the total number of AES invocations remains the same for a larger PRF block size. Check block size. Reading data from cloud storage is priced based on the number of GET requests. In the Check operation, downloading 1 byte per request will incur a huge monetary overhead. To reduce the number of GET requests, we can check a block of bytes of a tunable size (called the check block size). The checked blocks at the same offset of all code chunks will contain multiple rows of bytes. Although not necessary, it is recommended to set the check block size as a multiple of the PRF block size, so as to align with the PRF block operations.

AECC parameters: The AECC parameters ðn0; k0Þ control the error tolerance within a code chunk and the domain size of the PRP being used in AECC. Given the same k0, a larger n0 implies better protection, but introduces a higher computational overhead.

Checking percentage. The checking percentage _ defines the percentage of a file to be checked in the Check operation. A larger _ implies more robust checking, at the expense of both higher monetary and performance overheads with more data to download and check.

H. Flat XOR Codes

Flat Xor-based[33] codes make use of Minimal Erasure List(MEL),a fault tolerance metric for flat codes[34] . Second it makes use of Fault Tolerance Vector another fault tolerance metric for flat codes. The reliability of flat xor codes is determined by High-Fidelity Reliability Simulator[35] . The HFR Simulator is used to validate the efficacy of the redundancy placement algorithms. Traditionally, the Hamming distance is used to describe the fault tolerance of an erasure code: a code tolerates all sets of erasures smaller than the Hamming distance. The Hamming distance completely describes the fault tolerance of MDS codes, since all erasures at or beyond the Hamming distance result in data loss. Flat erasure codes can be nonMDS: they tolerate some sets of erasures at and beyond the Hamming distance. We previously developed the Minimal Erasures (ME) Algorithm to efficiently analyze a flat erasure code and characterize its fault tolerance[34] .Consider the following definitions: A set of erasures is a set of erased symbols; an erasure pattern is a set of erasures that results in at least one data symbol being irrecoverable; and, a minimal erasure is an erasure pattern in which every erasure is necessary and sufficient for it to be an erasure pattern. The ME Algorithm determines the minimal erasures list (MEL) of an erasure code: the list of the code's minimal erasures, which completely describes the fault tolerance of an erasure code. The MEL can be transformed into a minimal erasures vector MEV in which the ith element is the total number of minimal erasures of size i in the MEL. The length of the shortest minimal erasure in the MEL is the Hamming distance of the code, and so the first non-zero entry in the MEV corresponds to the Hamming distance.

For instance, consider (4,4)-RAID 10 specified by the parity bitmaps (1, 2, 4, 8). The first parity symbol for this code, s_4, is simply a replica of s_0 and so the bitmap is 1 (i.e.,s_4 = s_0 because 1 = 2^0). A more complex example is (5,3)-FLAT with parity bitmaps (7, 11, 29). The first parity symbol for this code, s_5, has bitmap 7 because it is computed as the XOR of data symbols $s_0, s_1,$ and s_2(i.e., $s_5 = s_0 \oplus s_1 \oplus s_2$ because 7 = $2^0 + 2^1 + 2^2$). The (4,4)-RAID 10 code is an example of a common RAID technique that simply replicates each data symbol. RAID 10 is a flat erasure code that tolerates any single disk failure. The MEL for the code is {$(s_0, s_4), (s_1, s_5), (s_2, s_6), (s_3, s_7)$} and the MEV is (0, 4, 0, 0). The MEL for (4,4)-RAID 10 is intuitive: whenever any pair of devices that store the same replicated symbol fails, data is lost. The MEV simply summarizes the count of minimal erasures of each size up to m. The MEL for (5,3)-FLAT is {$(s_4, s_7), (s_0, s_1, s_4), (s_0, s_1, s_7), (s_0, s_2, s_6),$
$(s_1, s_2, s_3), (s_1, s_5, s_6), (s_2, s_4, s_5)$
$(s_2, s_5, s_7), (s_3, s_4, s_6), (s_3, s_6, s_7)$}, and, the MEV is (0, 1, 10). This code better illustrates the irregularity that non-MDS flat codes can exhibit. There is no obvious structure or symmetry to the sets of device failures which lead to data loss. The Fault Tolerance Vector (FTV) indicates the probability that data is lost given some number of failures. To construct the FTV, the MEL is transformed into the erasures list (EL). The erasures list consists of every erasure pattern for a code. The EL is a super set of the MEL, and every element in it is either a minimal erasure or a super set of at least one minimal erasure. The erasures vector (EV) is to the EL what the MEV is to the MEL, and is easily

determined given the EL. Finally, the EV is transformed into the FTV. Let the ith entry of the EV be e_i. For a code with n symbols, the ith entry of the FTV is $e_i/\binom{n}{i}$. The FTV is the complement of the conditional probabilities vector described by Hafner and Rao[36]

The High-Fidelity Reliability (HFR) Simulator[35] is similar to the simulator developed by Elerath and Pecht[37] . Both are Monte Carlo reliability simulators that simulate disk failure, disk recovery, sector failure, and sector scrubbing, and both can useWeibull distributions for such failure and recovery rates. However, the HFR Simulator is high-fidelity in the sense that it simulates the reliability of non- MDS erasure codes that can tolerate two or more disk failures, with regard to both disk and sector failures. It is nontrivial to extend the methods of Elerath and Pecht in this manner. The difficulty in simulating non-MDS codes is in efficiently determining if a specific set of failures leads to data loss. The HFR Simulator has two modes of bookkeeping that allow it to efficiently determine if a set of device failures leads to data loss: via the MEL, and via the FTV. Using the MEL permits the HFR Simulator to accurately determine if a specific set of failures leads to data loss. Therefore, it is the method we must use to simulate the reliability of a redundancy placement of a flat code on heterogeneous devices. The FTV is a coarse-grained metric that does not capture the details of a specific placement of symbols on heterogeneous devices; however, it describes the fault tolerance of the median placement and provides good reliability.

IV. Conclusion

Throughout this paper, we give an overview of the various advances in regenerating codes for cloud storage systems. The design objective generally transfers from data integrity to resource overhead such as bandwidth consumption and storage efficiency. The exact regenerating codes help to maintain systematic erasure codes in the storage system such that no decoding operations are required to recover the original file. The repair-by-transfer regenerating codes help to achieve a repair process without arithmetic operations on both the newcomer and the providers. The FMSR codes are used to regenerate the data across a multi-server setting and performs a two-phase checking on the new code chunks generated in the repair operation. Flat XOR-codes—erasure codes in which parity disks are calculated as the XOR of some subset of data disks. These advances reduce the bandwidth consumption of the repair operations.

References

[1] Wuala, http://www.wuala.com/en/learn/technology, 2012.

[2] M. Al-Fares, A. Loukissas, and A. Vahdat, A scalable, commodity data center network architecture, in Proc. ACM SIGCOMM Conference on Data Communication (SIGCOMM), 2008.

[3] A. G. Dimakis, P. B. Godfrey, M. J. W. Y. Wu, and K. Ramchandran, Network coding for distributed storage system, IEEE Trans. Inform. Theory, vol. 56, no. 9, pp. 4539-4551, 2010.

[4] I. Reed and G. Solomon, Polynomial codes over certain finite fields, Journal of the Society for Industrial and Applied Mathematics, vol. 8, no. 2, pp. 300-304, 1960.

[5] C. Huang, H. Simitci, Y. Xu, A. Ogus, B. Calder, P. Gopalan, J. Li, and S. Yekhanin, Erasure coding in windows azure storage, in Proc. USENIX Annual Technical Conference (USENIX ATC), 2012.

[6] S.-Y. R. Li, R. W. Yeung, and N. Cai, Linear network coding, IEEE Trans. on Inform. Theory, vol. 49, no. 2, pp. 371-381, 2003.

[7] T. Ho, R. Koetter, M. Medard, D. Karger, and M. Effros, The benefits of coding over routing in a randomized setting, in Proc. IEEE International Symp. Inform. Theory, 2003, pp. 442-442.

[8] A. Duminuco and E. Biersack, A practical study of regenerating codes for peer-to-peer backup systems, in Proc. IEEE International Conference on Distributed Computing Systems (ICDCS), 2009, pp. 376-384.

[9] K. Rashmi, N. Shah, and P. Kumar, Optimal exactregenerating codes for distributed storage at the MSR and MBR points via a product-matrix construction, IEEE Trans. on Inform. Theory, vol. 57, no. 8, pp. 5227-5239, 2011.

[10] C. Suh and K. Ramchandran, Exact-repair MDS code construction using interference alignment, IEEE Trans. on Inform. Theory, vol. 57, no. 3, pp. 1425-1442, 2011.

[11] N. B. Shah, K. V. Rashmi, P. V. Kumar, and K. Ramchandran, Interference alignment in regenerating codes for distributed storage: Necessity and code constructions, IEEE Trans. on Inform. Theory, vol. 58, no. 4, pp. 2134-2158, 2012.

[12] V. R. Cadambe, S. A. Jafar, and H. Maleki, Distributed data storage with minimum storage regenerating codes – Exact and functional repair are asymptotically equally efficient, http://arxiv.org/abs/1004.4299, 2010.

[13] C. Suh and K. Ramchandran, On the existence of optimal exact-repair MDS codes for distributed storage, http://arxiv.org/abs/1004.4663, 2010.

[14] D. S. Papailiopoulos and A. G. Dimakis, Distributed storage codes through hadamard designs, in Proc. IEEE International Symposium on Information Theory (ISIT), 2011, pp. 1230-1234.

[15] Y. Wu, A construction of systematic MDS codes with minimum repair bandwidth, IEEE Trans. on Inform. Theory, vol. 57, no. 6, pp. 3738-3741, 2011.

[16] I. Tamo, Z. Wang, and J. Bruck, MDS array codes with optimal rebuilding, in Proc. IEEE International Symposium on Information Theory (ISIT), 2011, pp. 1240-1244

[17] V. R. Cadambe, C. Huang, S. A. Jafar, and J. Li, Optimal repair of MDS codes in distributed storage via subspace interference alignment, http://arxiv.org/abs/1106.1250,2011

[18] R. Bhagwan, K. Tati, Y.-C. Cheng, S. Savage, and G. M. Voelker, Total recall: System support for automated availability management, in Proc. USENIX Symposium on Networked Systems Design and Implementation (NSDI), 2004.

[19] Y. Hu, Y. Xu, X. Wang, C. Zhan, and P. Li, Cooperative recovery of distributed storage systems from multiple losses with network coding, IEEE Journal on Selected Areas in Communications, vol. 28, no. 2, pp. 268-276, 2010.

[20] K. Shum, Cooperative regenerating codes for distributed storage systems, in IEEE International Conference on

Communications (ICC), 2011.

[21] A. Kermarrec and N. Le, Repairing multiple failures with coordinated and adaptive regenerating codes, in IEEE International Symposium on Network Coding (NetCod), 2011.

[22] K. Shum and Y. Hu, Exact minimum-repair-bandwidth cooperative regenerating codes for distributed storage systems, in Proc. IEEE International Symposium on Information Theory (ISIT), 2011.

[23] A. Wang and Z. Zhang, Exact cooperative regenerating codes with minimum-repair-bandwidth for distributed storage, http://arxiv.org/abs/1207.0879, 2012.

[24] N. Le Scouarnec, Exact scalar minimum storage coordinated regenerating codes, in Proc. IEEE International Symposium on Information Theory (ISIT), 2012.

[25] K. Shum and Y. Hu, Functional-repair-by-transfer regenerating codes, in Proc. IEEE International Symposium on Information Theory (ISIT), 2012.

[26] Y. Hu, P. P. C. Lee, and K. W. Shum, Analysis and construction of functional regenerating codes with uncoded repair for distributed storage systems, http://arxiv.org/abs/1208.2787v1, 2012.

[27] N. B. Shah, K. V. Rashmi, P. V. Kumar, and K. Ramchandran, Distributed storage codes with repair-by-transfer and nonachievability of interior points on the storage-bandwidth tradeoff, IEEE Trans. on Inform. Theory, vol. 58, no. 3, pp. 1837-1852, 2012

[28] K. Rashmi, N. Shah, and P. Kumar, Explicit construction of optimal exact regenerating codes for distributed storage, in Forty-Seventh Annual Allerton Conference on Communication, Control, and Computing, 2009.

[29] M.O. Rabin, "Efficient Dispersal of Information for Security, Load Balancing, and Fault Tolerance,"J. ACM, vol. 36, no. 2, pp. 335-348, Apr. 1989.

[30] J.S. Plank, "A Tutorial on Reed-Solomon Coding for FaultTolerance in RAID-Like Systems,"Software - Practice & Experience,vol. 27, no. 9, pp. 995-1012, Sept. 1997.

[31] Y. Hu, H. Chen, P. Lee, and Y. Tang, "NCCloud: Applying Network Coding for the Storage Repair in a Cloud-of-Clouds," Proc. 10th USENIX Conf. File and Storage Technologies (FAST '12), 2012

[32] Henry C.H. Chen and Patrick P.C. Lee, Enabling Data Integrity Protection in Regenerating-Coding-Based Cloud Storage: Theory and Implementation, IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, VOL. 25, NO. 2, FEBRUARY 2014

[33] Kevin M. Greenan∗†, Ethan L. Miller†, and Jay J. Wylie, Reliability of flat XOR-based erasure codes on heterogeneous devices, International Conference on Dependable Systems & Networks: Anchorage, Alaska, June 24-27 2008.

[34] J. J. Wylie and R. Swaminathan. Determining fault tolerance of XOR-based erasure codes efficiently. In DSN-2007, pages 206–215. IEEE, June 2007.

[35] K. M. Greenan and J. J. Wylie. High-fidelity reliability simulation of erasure-coded storage. Technical Report (to appear), Hewlett-Packard Labs.

[36] J. L. Hafner and K. Rao. Notes on reliability models for non-MDS erasure codes. Technical Report RJ–10391, IBM, October 2006.

[37] J. F. Elerath and M. Pecht. Enhanced reliability modeling of raid storage systems. In DSN-2007, pages 175–184. IEEE, June 2007

AUTHORS

First Author – V Anto Vins, (M.E)¯, Prathyusha Institue of Technology and Management, anto.vins@gmail.com .
Second Author – S. Umamageswari, M.E, Prathyusha Institute of Technology and Management, umasrinivasan72@rediffmail.com .
Third Author – P.Saranya, M.E, Prathyusha Institute of Technology and Management, saran.pitam@gmail.com .