Functional Minimum Storage Regenerating Codes For Multi-Cloud Failure

Anitha p
Assistant professor,Dept of ISE,JSSATE,BANGALORE

Abstract— This paper reduces the network repair traffic and recovery from multiple node failures in a cloud storage environment. Suppose if a cloud suffers from permanent failure and loses all its data by more than one node, then we need to regenerate the lost data with the help of the other neighboring clouds. We design a proxy based storage system for fault tolerant multiple cloud storage called NC cloud, which achieves reduction in repair traffic for a permanent multi cloud failure. Functional minimum storage regenerating [FMSR] codes is used for regenerating data in multicloud failure. we also prove that FMSR codes reduces repair traffic and incur less monitory cost during data transfer than Erasure [RAID-6] codes and EMSR [exact minimum storage regenerating]codes.

Index Terms—multicloud, regenerating codes, repair traffic, fault tolerance, recovery.

1. INTRODUCTION

Cloud storage denotes a family of increasingly popular on-line services for archiving, backup, and even primary storage of files.[11] Cloud-storage providers offer users clean and simple file system interfaces and also raises concerns such as having single or multi point of failure and vendor lock- ins[7][1][4]. A possible solution is to distribute data across different cloud Providers and by exploiting the diversity of multiple clouds as suggested in [10][11][1].Using this we can improve the fault tolerance of cloud storage.

Cloud storage denotes a family of increasingly popular on-line services for archiving, backup, and even primary storage of files.[11] Cloud-storage providers offer users clean and simple file-system interfaces and also raises concerns such as having single or multi point of failure and vendor lock- ins[7][1][4]. A possible solution is to distribute data across different cloud providers and by exploiting the diversity of multiple clouds as suggested in [10][11][4].Using this we can improve the fault tolerance of cloud storage.

From disk arrays through clouds to archival systems, storage systems must tolerate failures and prevent data loss. There are 2 types of cloud failures: Transient and permanent failures. Erasure coding provides the fundamental technology for storage systems to add redundancy and tolerate short-term transient failures or foreseeable permanent failures[1].Amazon S3 [1] is a well known example for a permanent failure during 2011[12].

P.Anitha, working for JSS Academy of technical education as an Asst Prof in the Dept of ISE.(panitha_80@rediffmail.com)

So this paper focuses on unexpected permanent cloud failures.

When a permanent failure of a cloud occurs, it is necessary to activate repair in order to maintain redundancy of data and fault tolerance. Since data has been striped across different cloud providers, the repair operation recovers data from existing surviving clouds over the network and restores the lost data in a new cloud. Moving an enormous amount of data across clouds can introduce significant monetary costs. So it is important to reduce the network repair traffic (i.e., the amount of data being transferred over the network during repair), storing data redundantly and hence the monetary costs due to data migration.

Redundancy can be achieved by two common methods , replication and erasure coding. Replication is the simplest redundancy scheme, where *c* identical copies of a file are kept at *c* nodes, each node with one copy. The other is an (*n*, *k*) Maximum-Distance Separable (MDS) code, where each file of size *M* bytes is divided into *k* fragments of size *M/k* bytes and the *k* fragments are encoded into *n* fragments stored at *n* nodes, each node with one fragment, where $n > k$. The key property of erasure coding is that the original file can be reconstructed from any *k* fragments. Compared with replication, erasure coding uses an order of magnitude less bandwidth and storage to provide the same system availability[12].

Regenerating Codes

Upon failure of an individual node, a self-sustaining data storage network must necessarily possess the ability to *regenerate* (i.e., repair) a failed node. An obvious means to accomplish this is to permit the replacement node to connect to any nodes, download the entire message, and extract the data that was stored in the failed node. But downloading the entire units of data in order to recover the data stored in a single node that stores only a fraction of the entire message is wasteful, and raises the question as to whether there is a better option. Such an option is indeed available and provided by the concept of what is known as a *regenerating co**de***

Regenerating codes has been used for reducing repair traffic and storing the data redundantly in a distributed storage system(a collection of interconnected storage nodes). Regenerating codes are built on the concept of network coding [2][9]. Network codes designed specifically for distributed storage systems have the potential to provide dramatically higher storage efficiency for the same availability. One main challenge in the design of such codes is the exact repair problem: if a node storing encoded information fails, in order to maintain the same level of reliability we need to create encoded information at a new node. One of the main open problems in this emerging area has been the design of simple coding schemes that allow exact and low cost repair of failed nodes and have high data rates.

In this paper, a proxy-based storage system has been proposed for providing fault-tolerant storage over multiple cloud storage providers, referred to as NCCloud. Our FMSR code implementation maintains double-fault tolerance and has the same storage cost as in traditional erasure coding schemes based on RAID-6 codes, but uses less repair traffic when recovering a single-cloud failure and somewhat more for recovering a multi cloud failure. In particular, we eliminate the need to perform encoding operations within storage nodes during repair, while preserving the benefits of network coding in reducing repair traffic.

Summary of this paper using FMSR code is a s follows:

1. In multi cloud storage, FMSR codes can save the repair cost by 25% compared to RAID 6.FMSR codes are designed in such a way that double fault tolerance is used. FMSR code can save repair traffic cost by 25% and maintain the same amount of storage overhead as compared to RAID 6 codes. Note that FMSR codes can be deployed in a thin-cloud setting as they do not require storage nodes to perform encoding during repair, while still preserving the benefits of network coding in reducing repair traffic. Thus, FMSR codes can be readily deployed in today's cloud storage services.
2. Inordere to provide the implementation details of how a file object can be stored via FMSR codes, we propose a two-phase checking scheme, which ensures that double-fault tolerance is maintained in the current and next round of repair. By performing two-phase checking, we ensure that double-fault tolerance is maintained after iterative rounds of repair of node failures. We conduct simulations to validate the importance of two-phase checking.•
3. We conduct monetary cost analysis to show that FMSR codes effectively reduce the cost of repair when compared to traditional erasure codes, using the price models of today's cloud storage providers.
4. We conduct extensive experiments on both local cloud and commercial cloud settings. We show that our FMSR code implementation only adds a small encoding overhead, which can be easily masked by the file transfer time over the Internet. Thus, our work validates the practicality of FMSR codes via NCCloud, and motivates further studies of realizing regenerating codes in large-scale deployments

2. BACKGROUND

2.1 Erasure codes

Classical coding theory focuses on the tradeoff between redundancy and error tolerance. In terms of redundancy-reliability tradeoff, the Maximum Distance Separable (MDS) codes are optimal. The most well-known family on erasure coding focus on other performance metrics. decoding complexity. Another line of research for erasure coding in storage applications is parity array codes; see, e.g., [16], [17], [18], [19]. The array codes are based solely on XOR operations and they are generally designed with the objective of low encoding, decoding, and update complexities. See also the tutorial by Plank [20] on erasure coding for storage applications of MDS erasure codes is Reed-Solomon codes.

3. IMPORTANCE OF REPAIR IN CLOUD STORAGE

In this section we will brief the importance of repair in a cloud storage, especially in a permanent failures. Earlier we come across the transient and permanent failures, now we will discuss about the difference between those types. A transient failure is expected to be short-term, such that the "failed" cloud will return to normal after some time and no outsourced data is lost. Transient failures are listed in the Table 1, where the durations of such failures range from several minutes to several days.

Permanent failure. A failure in which data will be lost permanently from the failed cloud refers to a permanent cloud node failure. So data cannot be recoverable in a permanent failure compared to a transient one. Although we expect that a permanent failure is very rare to happen, there are several situations where permanent failures are still possible:

1. Loss and corruption of data: There are several cases of cloud services losing or corrupting customer data[10][4]. For example, in October 2009 a subsidiary of Microsoft, Danger Inc., lost the contacts, notes, photos, etc. of a large number of users of the Sidekick service [Sarno 2009]. The data was recovered several days later, but the users of Magnolia were not so lucky in February of the same year, when the company lost half a terabyte of data that it never managed to recover.
2. Destructive attacks. To provide security guarantees for outsourced data, one solution is to have the client application encrypt the data before putting the data on the cloud. On the other hand, if the outsourced data is corrupted (e.g., by virus or malware), then even though the content of the data is encrypted and remains confidential to outsiders, the data itself is no longer useful. AFCOM [48] found that about 65 percent of data centers have no plan or procedure to deal with cyber-criminals.
3. Data center outages in disasters. AFCOM [48] found that many data centers are ill-prepared for disasters[4]. For example, 50% of the respondents have no plans to repair damages after a disaster. It was reported[48] that the earthquake and tsunami in northeastern Japan in March 11, 2011 knocked out several data centers there.

Compared to transient failures, permanent failures will make the data no longer accessible.

Cloud service	Failure reason	Duration	Date
Google Gmail	Software bug [24]	4 days	Feb 27-Mar 2,2011
Google Search	Programming error [38]	40 mins	Jan 31,2009
Amazon S3	Gossip protocol blowup [9]	6-8 hours	July 20,2008
Microsoft Azure	Malfunction in Windows Azure [36]	22 hours	Mar 13-14,2008

TABLE 1

Examples of transient failures in different cloud services.

4. FMSR CODE DESCRIPTION

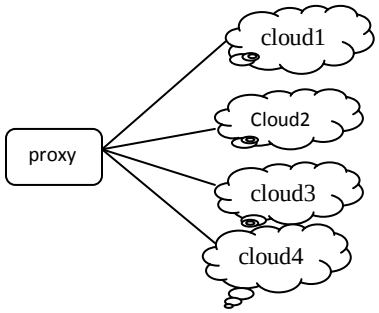
FMSR codes preserve the benefits of network coding as they minimize the repair bandwidth (e.g., the repair and width saving compared to RAID-6 codes is up to 50% [22][21]). FMSR codes use *uncoded* repair without requiring encoding of surviving nodes during repair,and this can minimize disk reads as the amount of data read from disk is the same as that being transferred. FMSR codes are designed as *non-systematic* codes as they do not keep the original uncoded data as their systematic counterparts, but instead store only linear combinations of original data called *parity chunks*. Each round of repair regenerates new parity chunks for the new node and ensures that the fault tolerance level is maintained. A trade-off of FMSR codes is that the whole encoded file must be decoded first if parts of a file are accessed. Nevertheless, FMSR codes are suited to long-term archival applications, since data backups are rarely read and it is common to restore the whole file rather than file parts.

Because of above advantages of FMSR code we consider a distributed, multiple-cloud storage setting from a client's perspective, where data is striped over multiple cloud providers. We propose a proxy-based design [1], [30] that interconnects multiple cloud repositories, as shown in Figure 1(a). The proxy serves as an interface between client applications and the clouds. If a cloud experiences a permanent failure, the proxy activates the repair operation, as shown in Figure 1(b).

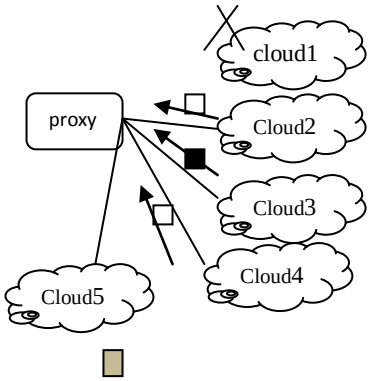
From the above diagram, proxy reads the essential data pieces from other surviving clouds, reconstructs new data pieces, and writes these new pieces to a new cloud. Note that this repair operation does not involve direct interactions among the clouds.

Now consider fault-tolerant storage based on a type of maximum distance separable (MDS) codes. Given a file object of size M , we divide it into equal-size native chunks, which are linearly combined to form code chunks. When an (n, k)-MDS code is used, the native/code chunks are then distributed over n (larger than k) nodes,each storing chunks of a total size M/k, such that the original file object may be reconstructed from the chunks contained in any k of the n nodes. Thus, it tolerates the failures of any $n - k$ nodes. We call this fault tolerance feature the MDS property. The extra feature of FMSR codes is that reconstructing the chunks stored in a failed node can be achieved by downloading less data from the surviving nodes than reconstructing the whole file. his paper considers a multiple-cloud setting with two levels of reliability: fault tolerance and recovery. First, we assume that the multiple-cloud storage is double-fault tolerant (e.g., as in conventional RAID-6 codes) and provides data availability under the transient unavailability of at most two clouds. That is, we set $k = n - 2$. Thus, clients can always access their data as long as no more than two clouds experience transient failures (see examples in Table 1) or any possible connectivity problems. We expect that such a fault tolerance level suffices in practice.

Second, we consider single-fault recovery in multiple-cloud storage, given that a permanent cloud failure is less frequent but possible. Our primary objective is to minimize the cost of storage repair (due to the migration of data over the clouds) for a permanent single-cloud failure. In this work, we focus on comparing two codes: traditional RAID-6 codes and our FMSR codes with double-fault tolerance3 .



1 (a)Normal operation



1(b) Repair operation

We define the repair traffic as the amount of outbound data being downloaded from the other surviving clouds during the single-cloud failure recovery. We seek to minimize the repair traffic for cost-effective repair. Here, we do not consider the inbound traffic (i.e., the data being written to a cloud), as it is free of charge for many cloud providers . Suppose that we store a file of size M on four clouds, each viewed as a logical storage node. As we know in conventional RAID-6 codes, which is double-fault tolerant and the implementation is based on the Reed-Solomon code [23]. According to RAID 6 a file will be divided into two native chunks (i.e., A and B) of size M /2 each and add two code chunks formed by the linear combinations of the native chunks. Suppose now that Node 1 is down. Then the proxy must download the same number of chunks as the original file from two other nodes (e.g., B and A + B from Nodes 2 and 3, respectively). It then reconstructs and stores the lost chunk A on the new node. The total storage size is 2M , while the repair traffic is M . Regenerating codes have been proposed to reduce the repair traffic. One class of regenerating codes is called the exact minimum-storage regenerating (EMSR) codes[24].

EMSR codes keep the same storage size as in RAID-6 codes, while having the storage nodes send encoded chunks to the proxy so as to reduce the repair traffic. In EMSR coding a file will be divided into 4 chunks, and allocate the native and code chunks. Suppose Node 1 is down. To repair it, each surviving node sends the XOR summation of the data chunks to the proxy, which then reconstructs the lost chunks.

We can see that in EMSR codes, the storage size is 2M (same as RAID-6 codes), while the repair traffic is 0.75M , which is 25% of saving (compared with RAID-6 codes).

We now propose the double-fault tolerant implementation of FMSR codes as shown in Figure 2(a). We divide the file into four native chunks, and construct eight distinct code chunks P1 , · · · , P8 formed by different linear combinations of the native chunks. Each code chunk has the same size M /4 as a native chunk. Any two nodes can be used to recover the original four native chunks. Suppose Node 1 is down. The proxy collects one code chunk from each surviving node, though 2 nodes are enough to reconstruct the failed node, and downloads three code chunks of size M /4 each. Then the proxy regenerates two code chunks P1 and P2 formed by different linear combinations of the three code chunks as shown in the fig 2(a).

Suppose 2 nodes fail at a time as in fig 2(b), in which from the remaining 2 surviving nodes any one failed node will be recovered and using that recovered node along with node 3 & 4 another failed node can be recovered as shown in the fig2(b) &(c) . Note that P1 and P2 are still linear combinations of the native chunks. The proxy then writes P1 and P2 to the new node. In FMSR codes, the storage size is 2M (as in RAID-6 codes), yet the repair traffic is 0.75M , which is the same as in EMSR codes. A key property of our FMSR codes is that nodes do not perform encoding during repair.

To generalize double-fault tolerant FMSR codes for n storage nodes, we divide a file of size M into 2(n − 2) native chunks, and use them to generate 2n code chunks M Then each node will store two code chunks of size 2(n−2) Mn each. Thus, the total storage size is n−2 . To repair a failed node, we download one chunk from each of the other n−1 for one node failure and n-2 for 2 node failures. so the repair traffic is M (n−1) . In contrast, for 2(n−2) MnRAID-6 codes, the total storage size is also n−2 , while the repair traffic is M . When n is large, FMSR codes can save the repair traffic by close to 50%.

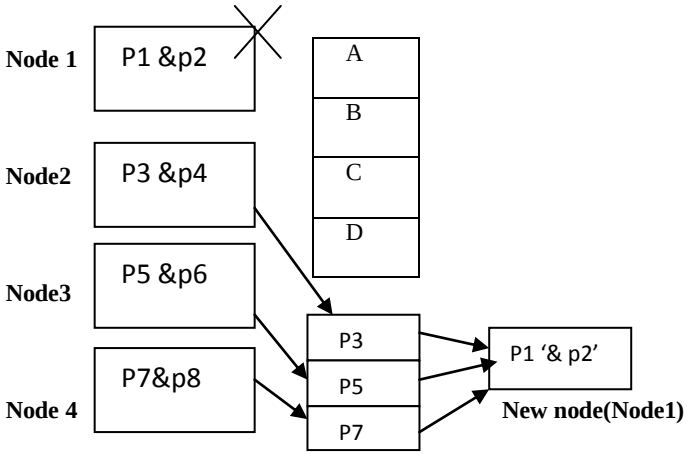
Note that FMSR codes are non-systematic, as they keep only code chunks but not native chunks. To access a single chunk of a file, we need to download and decode the entire file for that particular chunk. This is opposed to systematic codes (as in traditional RAID storage), in which native chunks are kept. Nevertheless, FMSR codes are acceptable for long-term archival applications, where the read frequency is typically low. Also, to restore backups, it is natural to retrieve the entire file rather than a particular chunk [25].

5.IMPLEMENTATION

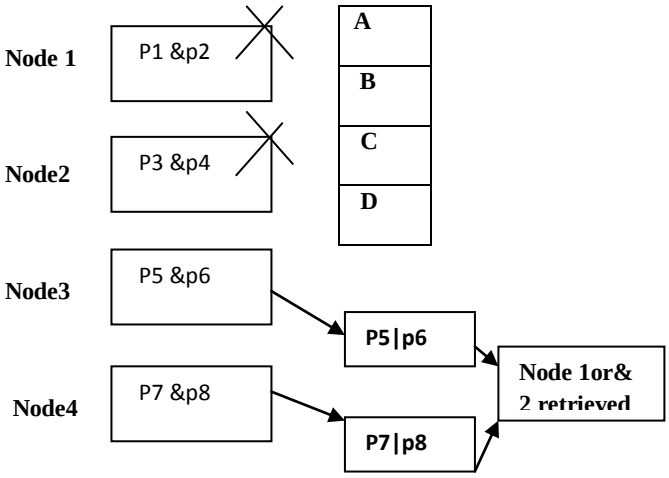
The proposed FMSR code for multiple cloud storage has three operations on a particular file object:
(1) file upload;
(2) file download;
(3) repair.

Each cloud repository is viewed as a logical storage node. The implementation assumes a thin-cloud interface , such that the storage nodes (i.e., cloud repositories) only need to support basic read/write operations. Thus, we expect that our FMSR code implementation is compatible with today's cloud storage services.

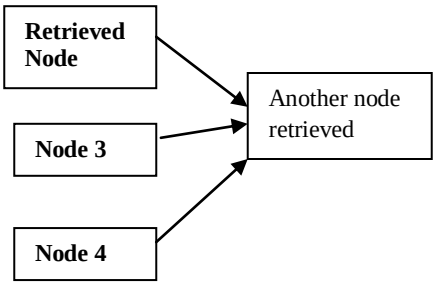
One property of FMSR codes is that we do not require lost chunks to be exactly reconstructed, but instead in each repair, we regenerate code chunks that are not necessarily identical to those originally stored in the failed node, as long as the MDS property holds. We propose a two-phase checking scheme, which ensures that the code chunks on all nodes always satisfy the MDS property, and hence data availability, even after iterative repairs. In this section, we analyze the importance of the two-phase checking scheme.



2 a) FMSR codes for one node failure



2b) FMSR codes for two node failures



2c) Retrieval of second node

4.1 Uploading a file in a multi cloud environment

First thing is, divide a file F into k(n − k) equal- size native chunks, denoted by (Fi)i=1,2,⋯,k(n−k) . then encode these k(n − k) native chunks into n(n − k) code chunks, denoted by (Pi)i=1,2,⋯,n(n−k) . Each Pi is formed by a linear combination of the k(n − k) native chunks. Specifically, we let EM = [αi,j] be an n(n − k) × k(n−k) encoding matrix for some coefficients αi,j (where i = 1, . . . , n(n − k) and j = 1, . . . , k(n − k)) in the Galois field GF(28). We call a row vector of EM an encoding coefficient vector (ECV), which contains k(n − k) elements. We let ECVi denote the ith row vector of EM. We compute each Pi by the product of ECVi and all the native k(n−k) chunks F1 , F2 , · · · , Fk(n−k) , i.e., Pi = j=1 αi,j Fj for i = 1, 2, · · · , n(n − k), where all arithmetic operations are performed over GF(28). The code chunks are then evenly stored in the n storage nodes, each having (n−k) chunks. Also, we store the whole EM in a metadata object that is then replicated to all storage nodes. There are many ways of constructing EM, as long as it passes our two-phase checking . Note that the implementation details of the arithmetic operations in Galois Fields are extensively discussed in [22].

4.2 File Download

To download a file, we first download the corresponding metadata object that contains the ECVs. Then we select any k of the n storage nodes, and download the k(n − k) code chunks from the k nodes. The ECVs of the k(n − k) code chunks can form a k(n−k)×k(n−k) square matrix. If the MDS property is maintained, then by definition, the inverse of the square matrix must exist. Thus, we multiply the inverse of the square matrix with the code chunks and obtain the original k(n − k) native chunks.

The idea is that we treat FMSR codes as standard Reed- Solomon codes, and our technique of creating an inverse matrix to decode the original data has been described in the tutorial [26].

4.3 REPAIR

After upload and download a file the last one is how to repair a permanent multi node failure. Given that FMSR codes regenerates different chunks in each repair, one challenge is to ensure that the MDS property still holds even after iterative repairs. This is in contrast to regenerating the exact lost chunks as in RAID-6, which guarantees the invariance of the stored chunks. Here, we propose a two-phase checking heuristic as follows. Suppose that the (r − 1)th repair is successful, and we now consider how to operate the rth repair for a permanent multi node failure (where r ≥ 1).

.

We first check if the new set of chunks in all storage nodes satisfies the MDS property after the rth repair. In addition, we also check if another new set of chunks in all storage nodes still satisfies the MDS property after the (r + 1)th repair, should another single permanent node failure occur (we call this the repair MDS (rMDS) property). We now describe the rth repair as follows:

Step 1: From the above explanation we know that any 2 nodes are enough to reconstruct a failed node. let n be the number of nodes and m be the number of failed nodes(for example if n=6 and m<=n-2). Download the encoding matrix from a surviving node. Recall that the encoding matrix EM specifies the ECVs for constructing all code chunks via linear combinations of native chunks. We use these ECVs for our later two-phase checking. Since we embed EM in a metadata object that is replicated, we can simply download the metadata object from one of the surviving nodes.

Step 2: Select one ECV from each of the surviving nodes. Each ECV in EM corresponds to a code chunk. We pick one ECV from each of the surviving nodes. We call these ECVs to be ECVi1 , ECVi2 , · · ·, ECVin−1 .

Step 3: Generate a repair matrix. Construct a repair matrix RM = [γi,j], where each element γi,j (where i = 1, . . . , n − k and j = 1, . . . , n − 1) is randomly selected in GF(28). Note that the idea of generating a random matrix for reliable storage is consistent with that in [27].

Step 4: Compute the ECVs for the new code chunks and reproduce a new encoding matrix. We multiply RM with the ECVs selected in Step 2 to construct n−k new ECVs, n−1 denoted by ECVi = j=1 γi,j ECVij for i = 1, 2, · · · , n−k. Then we reproduce a new encoding matrix, denoted by EM , which is formed by substituting the ECVs of EM of the failed node with the corresponding new ECVs.

Step 5: Given EM , check if both the MDS and rMDS properties are satisfied. Intuitively, we verify the MDS property by enumerating all n subsets of k nodes to seek if each of their corresponding encoding matrices forms a full rank. For the rMDS property, we check that for any possible node failure (one out of n nodes), we can collect one out of n−k chunks from each of the other n−1 surviving nodes and reconstruct the chunks in the new node, such that the MDS property is maintained. The number of checks performed for the rMDS property is at most n(n − k)n−1 n . If n is small, then the enumeration complexities for both MDS and rMDS properties are manageable. If either one phase fails, then we return to Step 2 and repeat. We emphasize that Steps 1 to 5 only deal with the ECVs, so their overhead does not depend on the chunk size.

Step 6: Download the actual chunk data and regenerate new chunk data. If the two-phase checking in Step 5 succeeds, then we proceed to download the n − 1 chunks that correspond to the selected ECVs in Step 2 from the n − 1 surviving storage nodes to NCCloud. Also, using the new ECVs computed in Step 4, we regenerate new chunks and upload them from NCCloud to a new node.

Remark: We can reduce the complexity of two-phase checking with the proposed FMSR code construction in our recent work [28]. The proposed construction specifies the ECVs to be selected in Step 2 deterministically, and tests their correctness (i.e., satisfying both MDS and rMDS properties) by checking against a set of inequalities in Step 5. This reduces the complexity of each iteration as well as the number of iterations (i.e., number of times that Steps 2-5 are repeated) in generating a valid EM . Our current implementation of NCCloud includes the proposed construction.

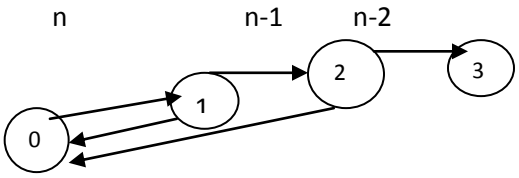


Fig 3.Markov model for double fault tolerant codes

MTTDL is solved via the Markov model. Figure 3 shows the Markov model for double-fault tolerant codes (i.e., k = n − 2), in which state i (where i = 0, 1, 2, 3) denotes the number of failed nodes in a storage system. State 3 means that there are more than two failed nodes and the data is permanently lost. We compute MTTDL as the expected time to move from state 0 (i.e., all nodes are normal) to state 3.

6. R ELATED WORK

Multiple-cloud storage. There are several systems proposed for multiple-cloud storage. NCCloud[4] provides node recovery for single node failure. NCCloud excludes the failed cloud in repair, Where as HAIL [10] provides integrity and availability guarantees for stored data. RACS [1] uses erasure coding to mitigate vendor lock-ins when switching cloud vendors. It retrieves data from the cloud that is about to fail and moves the data to the new cloud. Unlike RACS, DEPSKY [10] addresses Byzantine fault tolerance by combining encryption and erasure coding for stored data. All the above systems are built on erasure codes to provide fault tolerance, in this paper we are proposing multi node failure recovery using FMSR codes with an emphasis on both fault tolerance and storage repair.

Minimizing I/Os. Several studies propose efficient single-node failure recovery schemes that minimize the amount of data read (or I/Os) for XOR-based erasure codes. For example optimal recovery for specific RAID-6 codes reduces the amount of data read by up to around 25% (compared to conventional repair that downloads the amount of original data) for any number of nodes. Note that our FMSR codes can achieve 25% saving when the number of nodes is four, and up to 50% saving if the number of nodes increases. Authors of [35] propose an enumeration-based approach to search for an optimal recovery solution for arbitrary XOR-based erasure codes. Efficient recovery is recently addressed in commercial cloud storage systems. For example, new constructions of non-MDS erasure codes designed for efficient recovery are proposed for Azure [29]. The codes used in [31], [53] trade storage overhead for performance, and are mainly designed for data-intensive computing. Our work targets the cloud backup applications.

Regenerating codes will minimize the network traffic using network coding and it also reduces the repair traffic among storage nodes. Some studies[4] addresses the security issues for regenerating codes in case of single failure recovery. Whereas we proposed in this paper on multinode failure recovery(,which accounts for the majority of failures in cloud storage systems [31].

Some studies (e.g., [4]) address the security issues for regenerating-coded data, while the security aspect of FMSR codes is addressed in our prior work [30].

7.CONCLUSIONS

In this paper we proposed a multi node failure recovery using network coding(FMSR) method. We used a proxy-based, multiple-cloud storage system that practically addresses the reliability of today's cloud backup storage. This paper not only provides fault tolerance in storage, but also allows cost-effective repair when a cloud permanently fails. Recovery from multimode failure takes place only if the following condition is met:

M=N-2 (N:number of nodes & M:number of failure nodes) .

we proposed theoretically functional minimum storage regenerating (FMSR) codes, which regenerates new parity chunks during repair subject to the required degree of data redundancy. Our FMSR code implementation eliminates the encoding requirement of storage nodes (or cloud) during repair, while ensuring that the new set of stored chunks after each round of repair preserves the required fault tolerance. This paper shows the effectiveness of FMSR codes in the cloud backup usage, in terms of monetary costs and response times.

8. FUTURE WORK

FMSR code implementation has been proved in the recent papers[31] along with its correctness. Now we point out some of the issues of the existing design of FMSR codes, and we make them as future work.

While double-fault tolerance is the default setting of today's enterprise storage systems (e.g., 3-way replication in GFS [32]), it is unclear how to generalize FMSR codes for different (n, k) values. FMSR code used in[4] tells us about single node failure recovery. In this paper we discussed about multimode failure where M=N-2, but it is interesting to study if all the nodes in a cloud fail simultaneously. This can be posed as a future work . while single-node failures are the most common failure patterns in practical cloud storage systems [31], it is interesting to study how to generalize FMSR codes to support efficient repairs of concurrent node failures.

In FMSR codes, we always download the same amount of original data by connecting to any k nodes . while in traditional RAID-6 codes, the original amount of data is retrieved to recover the lost data. Thus, FMSR codes and traditional RAID-6 codes retrieve the same amount of data in degraded reads, while FMSR codes have higher computational overhead in decoding.

ACKNOWLEDGMENTS

This work is supported by JSS Academy of technical education. I am grateful for my institution.

REFERENCES

[1] H. Abu-Libdeh, L. Princehouse, and H. Weatherspoon. RACS: A Case for Cloud Storage Diversity. In Proc. of ACM SoCC, 2010.
[2] R. Ahlswede, N. Cai, S.-Y. R. Li, and R. W. Yeung. Network Information Flow. IEEE Trans. on Information Theory, 6(4):1204–1216, Jul 2000.
[3] K. V. Rashmi, Nihar B. Shah, and P. Vijay Kumar, Fellow, IEEE Optimal exact regenerating Storage at the MSR and MBR Points Via a Product-Matrix Construction" VOL. 57, NO. 8, AUGUST 2011.
[4] NCCloud:" A network coding based storage system in a cloud of c clouds", Henry C. H. Chen, Yuchong Hu, Patrick P. C. Lee, and Yang Tang, Jan 2014
[5] "Remote Data Checking for Network Coding-based",by Bo Chen, Reza Curtmola Department of Computer Science New Jersey Institute of Technology {bc47,crix}@njit.edu
[6] "Cooperative Recovery of Distributed Storage Systems from Multiple Losses with Network Coding", Yuchong Hu, Yinlong Xu, Xiaozhao Wang, Cheng Zhan and Pei Li.
[7] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia. "A View of Cloud Computing". Communications of the ACM, 53(4):50–58, 2010.
[8] " A survey on network codes for distributed storage," IEEE ,vol 99,no 3,mar2011
[9] Network Coding for Distributed Storage Systems Alexandros G. Dimakis, P. Brighten Godfrey, Yunnan Wu, Martin Wainwright And Kannan Ramchandran,IEEE trans,sep 2010
[10] DEPSKY: Dependable and Secure Storage in a loud-of-Clouds.In Proc. of ACM EuroSys, 2011. K. D. Bowers, A. Juels, and A. Oprea. HAIL: A High-Availability and Integrity Layer for Cloud Storage. In Proc. of ACM CCS, 2009.
[11] Business Insider. Amazon's Cloud Crash Disaster Permanently Destroyed Many Customers' 2011. data.http://www.businessinsider.com/amazon-lost-data-2011-4/, Apr.
[12] H.Blodget, "Amazon's cloud crash disaster permanently destroyed many customers data,",Apr 2011.
[13] M. Luby, M. Mitzenmacher, M. A. Shokrollahi, and D. Spielman, "Improved low-density parity check codes using irregular graphs," *IEEE Trans. Info. Theory*, vol. 47, pp. 585–598, February 2001.
[14] M. Luby, "LT codes," *Proc. IEEE Foundations of Computer Science (FOCS)*, 2002.
[15] A. Shokrollahi, "Raptor codes," *IEEE Trans. on Information Theory*, vol. 52, pp. 2551–2567, June 2006
[16] M. Blaum, J. Brady, J. Bruck, and J. Menon, "EVENODD: An efficient scheme for tolerating double disk failures," *IEEE Trans. On Computing*, vol. 44, pp. 192–202, February 1995.
[17] L. Xu and J. Bruck, "X-code: MDS array codes with optimal encoding," *IEEE Trans. on Information Theory*, vol. 45, pp. 272–276, January 1999
.[18] C. Huang and L. Xu, "STAR: An efficient coding scheme for correcting triple storage node failures," in *FAST-2005: 4th Usenix Conference on File and Storage Technologies*, (San Francisco, CA), December 2005.
19] J. L. Hafner, "WEAVER codes: Highly fault tolerant erasure codes for storage systems," in *FAST-2005: 4th Usenix Conference* on *File and Storage Technologies*, (San Francisco, CA),
[20] J. S. Plank, "Erasure codes for storage applications," in *Tutorial, FAST-2005: 4th Usenix Conference on File and Storage technologies*,(San Francisco, CA), December 2005. [online] http://www.cs.utk.edu/ plank/plank/papers/FAST-2005.html.
[21] Analysis and Construction of Functional Regenerating Codes with Uncoded Repair for Distributed Storage Systems Yuchong Hu†,Patrick P. C. Lee‡, and Kenneth W. Shum††Institute of Network Coding, The Chinese University of Hong Kong ‡Department of Computer Science and Engineering, Jan 2013.
[22] Y. Hu, H. C. H. Chen, P. P. C. Lee, and Y. Tang. NCCloud:" Applying Network Coding for the Storage Repair in a Cloud-of-Clouds". In *Proc. of FAST*, 2012.
[23] I. Reed and G. Solomon. Polynomial Codes over Certain Finite Fields. Journal of the Society for Industrial and Applied Mathematics, 8(2):300–304, 1960.
[24] C. Suh and K. Ramchandran. Exact-Repair MDS Code Theory, Construction using Interference Alignment. IEEE Trans. on Information 57(3):1425–1442, Mar 2011.
[25] H,C,H Chen and P. P. C. Lee. Enabling Data Integrity Protection in Regenerating-Coding-Based Cloud Storage. In Proc. of IEEE SRDS, 2012.
[26] J. S. Plank. A Tutorial on Reed-Solomon Coding for Fault-Tolerance in RAID-like Systems. Software-Practice&Experience, 27(9):995–1012, Sep 1997.
[27] M. O. Rabin. Efficient Dispersal of Information for Security, Load Balancing, and Fault Tolerance. Journal of the ACM, 36(2):335–348, Apr 1989.
[28] D. Ford, F. Labelle, F. I. Popovici, M. Stokely, V.-A. Truong, L. Barroso, C. Grimes, and S. Quinlan. Availability in Globally Distributed Storage Systems. In *Proc. of USENIX OSDI*, 2010.
[29] 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. of USENIX ATC*, 2012.
[30] P. B. Godfrey, Y. Wu, M. Wainwright, and K. Ram-chandran. Network Coding for Distributed Storage Systems. IEEE Trans. on Information Theory, 56(9):4539–4551, Sep 2010.