A CONTENT PROTECTION SCHEME FOR CLOUD CLIENTS THROUGH NETWORK CODING TECHNIQUE IN CLOUD

Brahmini. P[1], Ms. R. Poorva Devi [2], Dr. S. Rajalakshmi[3]

[1]PG Student, [2]Assistant Professor, Department of CSE,
SCSVMV University, Kanchipuram-631561,(India)
[3]Director, Advanced computing center (SJCAC), SCSVMV University,
Kanchipuram-631561, (India)

ABSTRACT

The real world scenario of cloud is users wish to store massive data in to the cloud database. So, Cloud must store the data without any failure. Sometimes, the data is striped across multiple cloud vendors. If a cloud suffers from a permanent failure and loses all its data, we need to repair the lost data with the help of the other surviving clouds to preserve data redundancy. We present a proxy-based storage system for fault-tolerant multiple-cloud storage with the help of network coding based storage scheme called the functional minimum-storage regenerating (FMSR) codes, which maintain the same fault tolerance and data redundancy as in traditional erasure codes (e.g., RAID-6). One key design feature of our FMSR codes is that we relax the encoding requirement of storage nodes during repair, while preserving the benefits of network coding in repair. With this approach, we may able to safeguard the user's data from the various vulnerabilities.

Keywords: Regenerating Codes, Network Coding, Fault Tolerance, Recovery.

I INTRODUCTION

Cloud storage provides an on-demand remote backup solution. However, using a single cloud storage provider raises concerns such as having a single point of failure and vendor lock-ins. By exploiting the diversity of multiple clouds, we can improve the fault-tolerance of cloud storage. While striping data with conventional erasure codes performs well when some clouds experience short-term transient failures or foreseeable permanent failures, there are real-life cases showing that permanent failures do occur and are not always foreseeable.

In view of this, this work focuses on unexpected permanent cloud failures. When a cloud fails permanently, it is necessary to activate repair to maintain data redundancy and fault tolerance. A repair operation retrieves data from existing surviving clouds over the network and reconstructs the lost data in a new cloud. To minimize repair traffic, regenerating codes have been proposed for storing data redundantly in a distributed storage system (a collection of interconnected storage nodes). Each node could refer to a simple storage device, a storage site, or a cloud storage provider.

Concept of network coding is that nodes can perform encoding operations and send encoded data. During repair, each surviving node encodes its stored data chunks and sends the encoded chunks to a new node, which then regenerates the lost data. It is shown that regenerating codes require less repair traffic than traditional erasure codes with the same fault tolerance level. One key challenge for deploying regenerating codes in practice is that most existing regenerating codes require storage nodes to be equipped with computation capabilities for performing encoding operations during repair. On the other hand, to make regenerating codes portable to any cloud storage service, it is desirable to assume only a thin-cloud interface, where storage nodes only need to support the standard read/write functionalities. This motivates us to explore, from an applied perspective, how to practically deploy regenerating codes in multiple-cloud storage, if only the thin-cloud interface is assumed.

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.

II IMPORTANCE OF REPAIR IN MULTI-CLOUD STORAGE

In this section, we discuss the importance of repair in cloud storage, especially in disastrous cloud failures that make stored data permanently unrecoverable. We consider two types of failures:

2.1 Transient Failure.

2.2 Permanent Failure.

2.1 Transient Failure

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. Table 1 show several real-life examples for the occurrences of transient failures in today's clouds, where the durations of such failures range from several minutes to several days.

Cloud Service	Failure Reason	Duration	Date
Google Gmail	Software Bug	3 days	Mar 2-Mar 4,2012
Google Search	Programming Error	28 minutes	Feb 7,2009
Amazon S3	Protocol Blowup	8-9 hours	Jan 20,2008
Microsoft Azure	Malfunction	22 hours	July 13-14,2008

Table I. Examples of transient failures in different cloud services

2.2 Permanent Failure

A permanent failure is long-term, in the sense that the outsourced data on a failed cloud will become permanently unavailable. Clearly, a permanent failure is more disastrous than a transient one. Although we expect that a permanent failure is unlikely to happen, there are several situations where permanent cloud failures are still possible:

1) Data center outages in disasters.
2) Data loss and corruption.
3) Malicious attacks.

III MOTIVATION OF FMSR CODES

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 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)

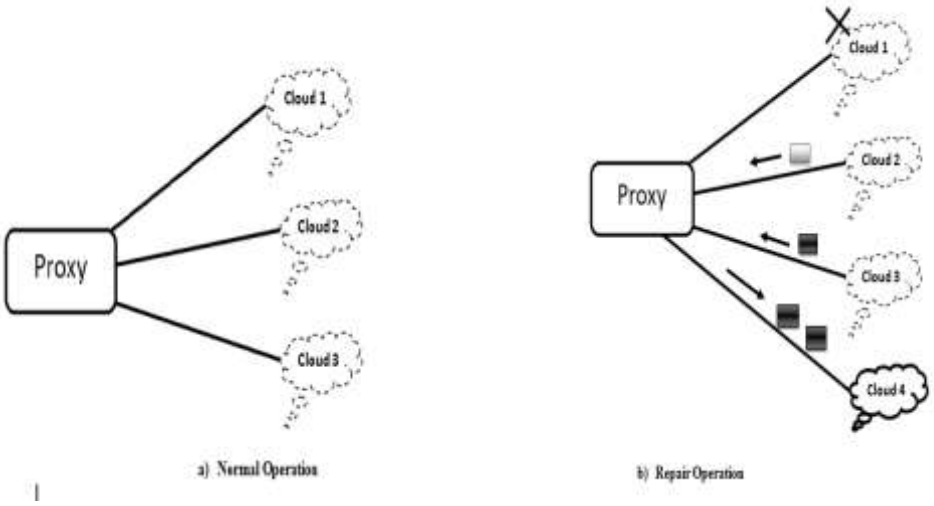

Figure I. Proxy-based design for multiple-cloud storage: (a) normal operation and (b) repair operation when Cloud node 1 fails. During repair, the proxy regenerates data for the new cloud.

That is, the proxy reads the essential data pieces from other surviving clouds, reconstructs new data pieces, and writes these new pieces to a new cloud.

We 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.

IV FMSR CODE IMPLEMENTATION

The details for implementing FMSR codes in multi cloud storage:

First, we specify three operations on file object:

1) File upload.
2) File download
3) Repair.

Each repository in cloud can be seen as a logical storage node. In our implementation, we assume a thin cloud interface, such that storage nodes need to support basic read, write operations.

One feature of FMSR codes is that lost chunks will not be reconstructed exactly, but in each repair operation, code chunks are regenerated which are not identical to those stored in the failed node.

4.1 Basic operations

The basic operations defined by FMSR codes are:

4.1.1 File Upload

To upload a file F, we first divide it into k(n−k) equal- size native chunks, denoted by (Fi)i=1,2,⋯,k(n−k). We then encode these k(n − k) native chunks into n(n − k) code chunks. We let EM = [αi,j] be an n(n−k)× k(n−k) encoding matrix

4.1.2 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.

4.1.3 Repairs

The repair of FMSR codes for a file F for a permanent single-node failure is that, given that FMSR codes regenerate's different chunks in each repair; one challenge is to ensure that the MDS property still holds even after iterative repairs. The repair operation is carried out as follows:

Step 1: Download the encoding matrix from a surviving node.

Step 2: Select one ECV from each of the n-1 surviving nodes.

Step 3: Generate a Repair matrix.

Step 4: Compute the ECVs for the new code chunks and reproduce a new encoding matrix.

Step 5: For the given EM, check if MDS property is satisfied.

Step 6: Download the actual chunk data and regenerate new chunk data.

V SIMULATION WORK

We conduct simulations to check the MDS property and can make iterative repairs. Our simulations are carried out in a 2.4 GHz CPU core. First, we consider multiple rounds of node repairs for different values of n, and argue that in addition to checking the MDS property, checking the rMDS property is essential for iterative repairs. Specifically, in each round, we randomly pick a node to fail, and then repair the failed node. We say a repair is bad if the loop of Steps 2 to 5 in our two-phase checking is repeated over a threshold number of times but no suitable encoding matrix has yet been obtained.

Figure 2 shows the number of rounds of repair that can be sustained when the rMDS property is checked or is not checked. It shows that checking the rMDS property enables us to sustain more rounds of repair before seeing a bad repair. For example, suppose that we set the threshold to be 10 loops. Then we can sustain 500 rounds of repair for different values of n (number of nodes) by checking the rMDS property, but we encounter a bad repair quickly (e.g., in 3 rounds of repair for n = 10) if we do not check the rMDS property.

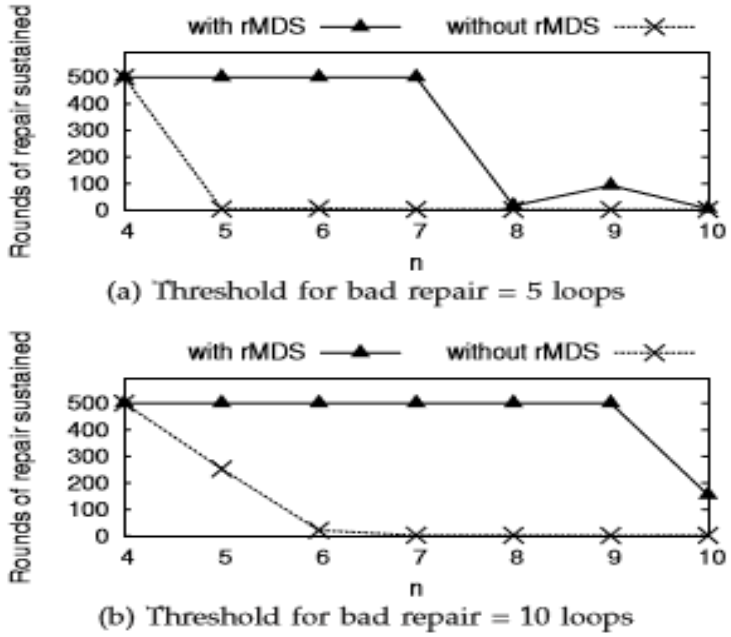

Figure 2. Number of rounds repairs sustainable without seeing bad repair.

Figure 3 plots the cumulative time of two-phase checking for 50 rounds of repair for n = 4 to n=12.It takes negligible time for the repair. For example, when n=6, it takes 0.06 seconds to carry out the repairs. Not only for n=6, but also for n=8 and n=12 it takes only 0.1seconds to carry out the repairs.

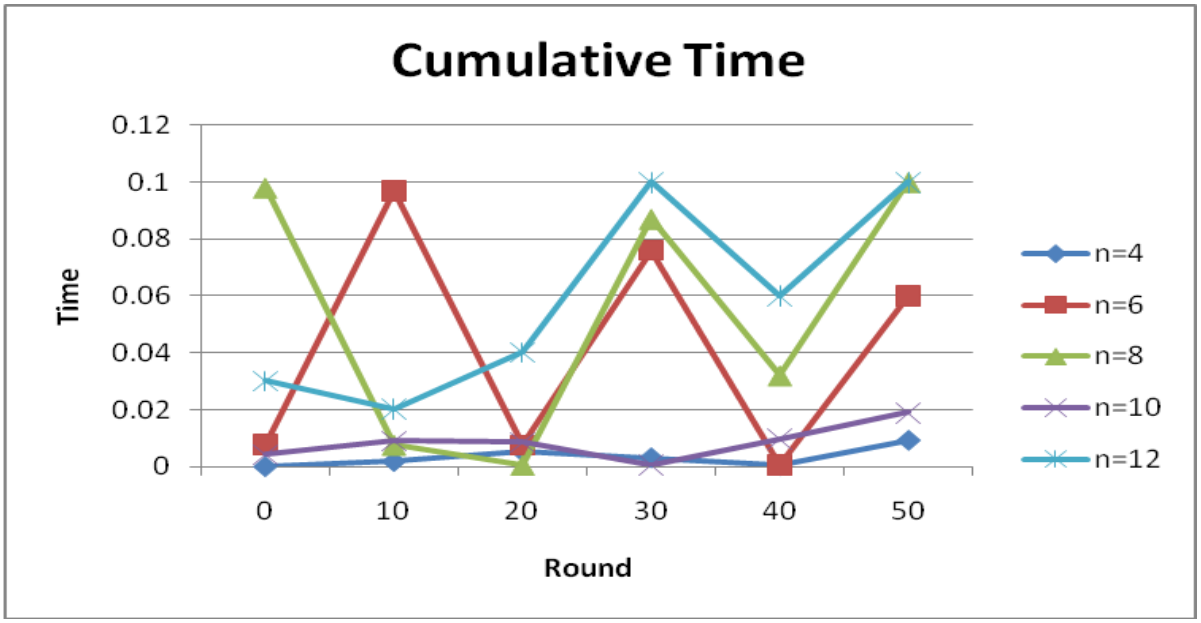

Figure 3: The cumulative time required for checking phase in 50 consecutive rounds of repair from n=4 to n=12.

VI NC CLOUD DESIGN

We now implement NCCloud as a proxy. It acts as a bridge between the user applications and the multi clouds. It has three layers. The three layers are:

1) File System Layer.
2) Coding Layer.
3) Storage Layer.

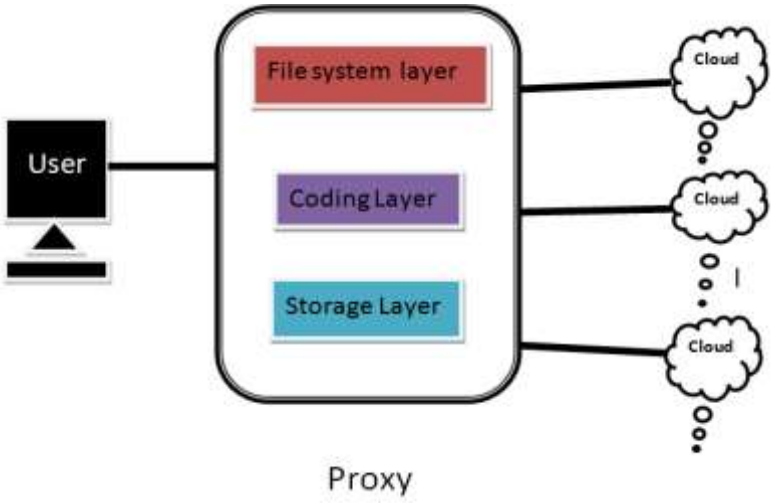

Figure 4. The role of NC Cloud as proxy.

6.1 File System Layer

It presents NCCloud as a mounted drive, which can thus be easily interfaced with the user applications.

6.2 Coding Layer

It deals with the encoding and decoding functions.

6.3 Storage Layer

It deals with the read and writes requests from different clouds.

Each file is associated with a metadata object, which is replicated at each repository. The metadata object holds the file details and the coding information i.e., encoding coefficients for FMSR codes.

VII EXPERIMENTAL RESULTS

In this section we tried to evaluate the NC Cloud implementation by using following three steps:

1) Fie Upload.
2) File Download.
3) Repair.

All the resultant set variables are calculated by using the cloud SLA constraints and need to incorporate the various security implications into the network coding phenomenon. The entire variable is specified in the two level implementation factors such as given below:

1) RAID-6
2) FMSR

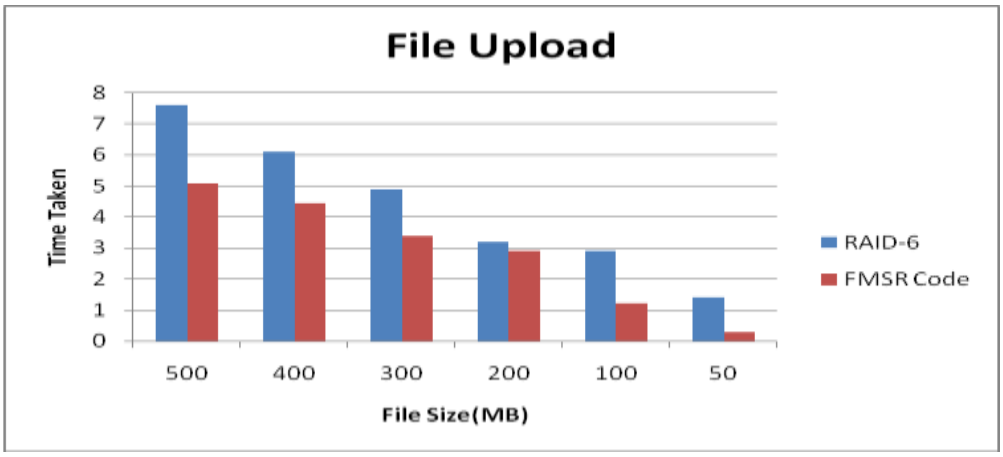

Figure 5. Time taken for File upload in RAID-6 and FMSR Codes

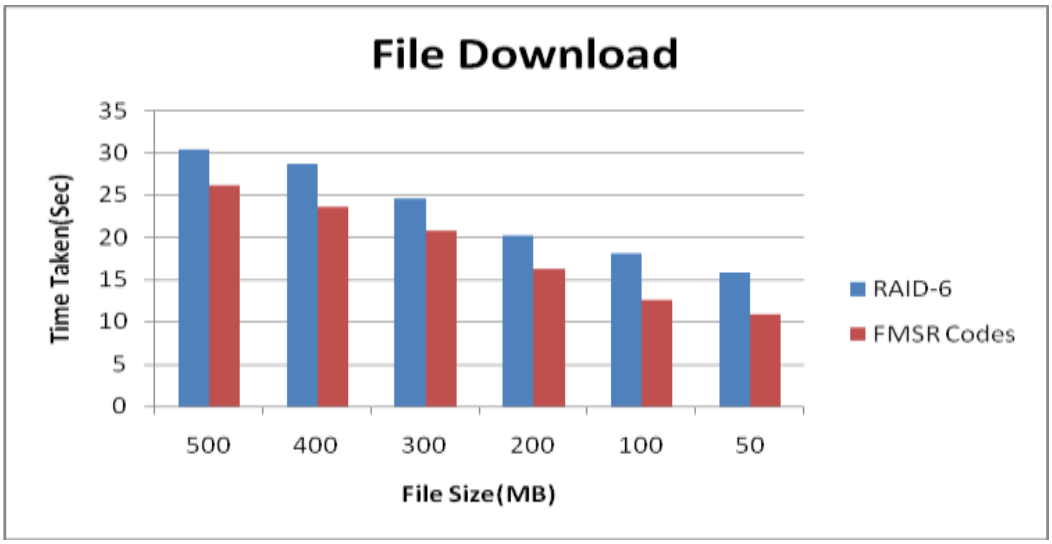

Figure 6Time taken for File download in RAID-6 and FMSR Codes

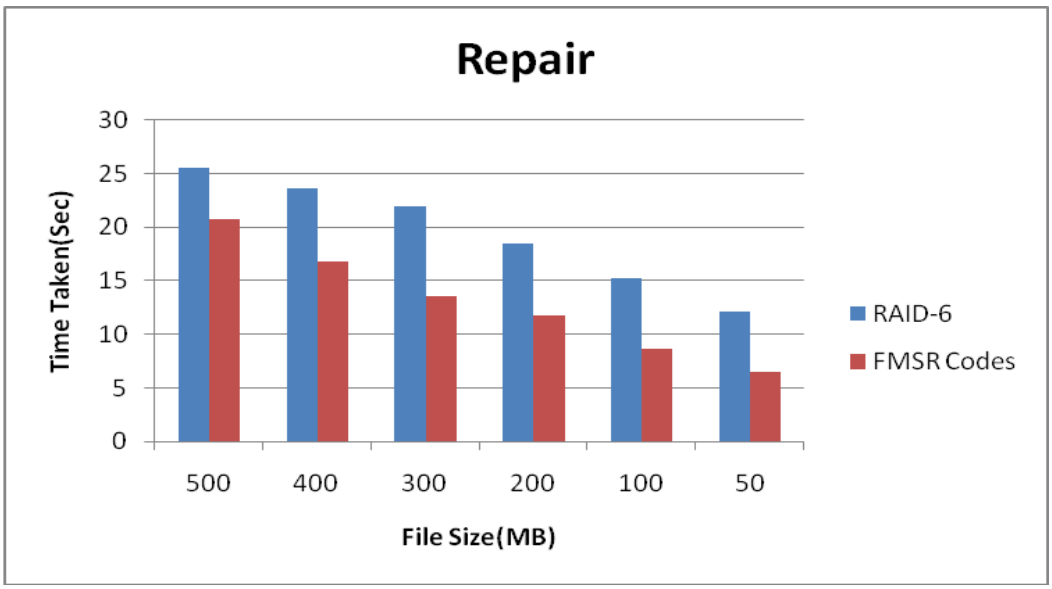

Figure 7. Time taken for Repair in RAID-6 and FMSR Codes

From, the above results it is clearly identified that FMSR is efficient than the previous erasure codes.

VIII CONCLUSION

The implementation of NCCloud, which is a proxy based storage system, addresses reliability issues. It also supports multi cloud storage for backup storage which is opt for today's cloud storage. The implementation of FMSR codes is also very efficient against the previous implementations. The double scheme of checking in steps 2 and 5 verify the MDS property. Even the response time for FMSR code is low.

IX FUTURE ENHANCEMENT

In this paper we studied the relationship between network coding and the cloud storage. In our study we have checked the fault tolerance mechanisms. But still there is scope for auditing the cloud for data integrity.

REFERENCES

[1] Truong- Huu. T A novel model for competition and cooperation among cloud providers IEEE transactions on cloud computing volume 2, issue 3, 2014.

[2] 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.

[3] A. G. Dimakis, P. B. Godfrey, Y. Wu, M. Wainwright, and K. Ramchandran. Network Coding for Distributed Storage Systems. IEEE Trans. on Information Theory, 56(9):4539–4551, Sep 2010.

[4] 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.

[5] K. Shum and Y. Hu. Exact Minimum-Repair-Bandwidth Cooperative Regenerating Codes for Distributed Storage Systems. In Proc. of IEEE Int. Symp. On Information Theory (ISIT), Jul 2011.

[6] L. Xiang, Y. Xu, J. Lui, Q. Chang, Y. Pan, and R. Li. A Hybrid Approach to Failed Disk Recovery Using RAID-6 Codes: Algorithms and Performance Evaluation. ACM Trans. on Storage, 7(3):11, 2011.

[7] B. Chen, R. Curtmola, G. Ateniese, and R. Burns. Remote Data Checking for Network Coding-Based Distributed Storage Systems. In Proc. of ACM CCSW, 2010.

[8] L. Xiang, Y. Xu, J. Lui, Q. Chang, Y. Pan, and R. Li. A Hybrid Approach to Failed Disk Recovery Using RAID-6 Codes: Algorithms and Performance Evaluation. ACM Trans. on Storage, 7(3):11, 2011.