FTCloud: Fault Tolerant Multiple Cloud Storage using Proxy based Storage System

Abhishek G, Kavyashree N,
Shreelakshmi D R, Shruthi S
Department of Computer Science and Engineering
T John Institute of Technology
Bangalore, India

Srinivasa H P
Associate Professor
Department of Computer Science and Engineering
T John Institute of Technology
Bangalore, India

***Abstract*— The recent trend is to stripe data across the multiple cloud vendors to provide fault tolerance. But when the cloud fails permanently and lose all its data we recover it with the help of the other surviving cloud which provides data redundancy. The solution is to provide a fault-tolerant multiple cloud storage using a proxy-based storage system called FTCloud, this achieves a cost effective repair for permanent double-cloud failure. FTCloud is built using network-coding based storage scheme called as FMSR(minimum-storage regenerating) codes technique, it provides more fault tolerance and data redundancy than the traditional techniques(e.g. like RAID6) and also use less repair traffic thereby incur less data transfer cost. The key feature of FMSR is that we provide encoding requirement relaxation by preserving the network coding benefits in repair. We implement this concept of FTCloud and deploy it on both local and commercial clouds. We check and validate that FMSR technique provides less cost and high performance in cloud storage operations like upload and download.**

Keywords - Regenerating Codes, Network Codes, Fault Tolerance, Recovery, Implementation, Experimentation.

I. INTRODUCTION

Cloud storage always provides the back up for data. But usage of single cloud storage gives rise to problems such as single point of failure, vendor lock in etc. The solution for this is to stripe data across different cloud providers. By making use of multiple cloud concept we can improve the fault tolerance of cloud storage.

During striping of data, the existing methods perform well when some clouds fails for shorter period of time or for permanent failures there are many real life cases which tells us the occurrence of permanent failure and are not anticipated. Here in this view our work focuses on unexpected *permanent cloud failure*. When the cloud fails permanently we need to activate a repair to maintain the data redundancy and to have a fault tolerance. The repair operation retrieves the lost data from surviving clouds over the network and it regenerates(reconstruct) lost data into a new cloud. In the recent days the cloud storage providers charge the users enormously for keeping the data backup, so moving the data across cloud require high monetary costs. It is very important to reduce repair traffic and also the monetary cost due to the data migration.

Regenerating codes concept is proposed to repair traffic for storing the data in a distributed storage system redundantly. Every node can refer to some simple storage device, a cloud storage provider or a storage site. The regenerating codes are formed using the network coding concept, wherein the nodes themselves perform the encoding operation and send the encoded data. While repair is taking place, the surviving node encode the data stored in it and send the encoded data to new node which regenerates the lost data. The advantage of regenerating code is that it require less repair traffic than the existing methods with better fault-tolerance level. The extensive study on regenerating codes are carried out in following contexts([14], [16], [29], [34], [41], [50], [51], [55]–[57]). But regenerating code's practical performance will always remain uncertain. The main key challenge for the deployment of regenerating codes in existing system require that the storage node itself perform encoding operation during repair. To make the regenerating codes portable to any cloud storage services, we need to assume only the thin-cloud interface where the storage node should only support the standard read/write functionalities. This helps us to know how practically we can deploy the regenerating codes in multiple cloud storage.

Here in our work, we present you the design and implementation of *FTCloud*, a proxy-based storage system designed for providing fault-tolerant storage over multiple cloud storage providers. FTCloud interconnect different clouds and transparently stripe data across clouds. We also propose the first implementable FMSR codes. i.e. functional minimum storage regenerating codes.

The FMSR code implementation provides double fault tolerance and also has the same cost as that of traditional RAID based schemes but uses less repair traffic while single cloud failure. We particularly eliminate the need of encoding operations within storage nodes during repair, also preserves benefit of network coding by reducing the repair traffic. According to the survey made by us, this is the first study that shows the application of regenerating codes in the storage system and evaluates it practically.

The main advantage of the FMSR codes is that it is non-systematic which means that only the encoded data is formed by the linear combinations of the original data and will not keep the original data like that of traditional schemes. The FMSR design is mainly applied in two cases;

(i) Where the data backup is maintained

(ii) Where the whole data in the file should be restored rather than the lost data.

In real-life examples there are many organizations which store an enormous amount of data like even in petabyte scale using the cloud storage the case studies is provided in [4], [8], [43], [59]. In August 2012, Amazon further introduced Glacier [5], a cloud storage offering optimized data back-up for low cost with slow and costly data retrieval. We provide a solution in form of FMSR codes that provide an alternative option for enterprises and organizations to store their data with the help of multiple-cloud storage in a fault-tolerant and cost-effective manner.

Our work is encouraged by the multiple cloud concept and is developed by keeping multiple cloud storage concept, the FMSR codes proposed would be applied in the distributed storage systems that are prone to failures. And it is also applied where network transmission bandwidth is limited. It is applicable where minimizing traffic is important which intern minimizes the overall repair time.

Our projects contribution is as below.
We are presenting a FMSR design by assuming the occurrence of double fault tolerance. Here we show that the FMSR can save 25% of repair cost when compared to RAID6, when 4 nodes are used. And it can also save up to 50% when number of storage nodes increases. The FMSR codes also maintain same amount of storage overhead as that of RAID6 codes. FMSR can also be implemented in thin cloud settings as they do not require encoding during repair. Hence FMSR codes can be deployed in today's cloud services.

- Here we let you know the implementation details of how the file objects can be stored through FMSR codes. Mainly we propose two-phase checking which conforms the concept of double-fault tolerance. This two phase checking ensures double fault tolerance through iterative repair of failure nodes.
- The monetary cost analysis is done to show the effectiveness of repair cost compared to traditional approaches.
- Here we conduct experiments on both local and commercial cloud settings. We ensure that our FMSR code implementation provides a small amount of encoding overhead that can be masked during file transfer over internet. This gives room for further research on FMSR codes in high-scale deployments.

The content of paper is as follows. Section 2 concentrates on the importance of multiple cloud storage. Section 3, concentrates on how FMSR codes reduce repair traffic through an example. Section 4, concentrates on implementable design of FMSR codes and analysis of iterative schemes of FMSR design. Section 5, concentrates on the deployment of FMSR codes. Section 6, Concentrates on evaluation of RAID-6 and FMSR codes using both private and commercial cloud settings. Section 7, Reviews related work. And Section 8 concludes the paper.

II. IMPORTANCE OF REPAIR IN MULTIPLE-CLOUD STORAGE

In this section, the discussion is on the importance of repair in cloud storage, mainly in the disastrous cloud failures which make the data to be lost permanently and is unrecoverable. Two types of failures are: 1) Transient failure 2) Permanent failure.

Transient failure: Transient failure is nothing but the cloud returns to normal after some time of failure and none of the data is lost. The table A shows some of the real time examples of occurrence of transient failure in today's clouds. It shows how the failure may occur from several minutes to even several days. It highlights that even though the cloud provider like Amazon claims that it provides service with 99.99% of availability[6], there are some raising concerns about this claim and the reliability of other cloud providers after Amazon's outage in April 2011 [12]. The transient failures are common in the clouds, but they will be eventually recovered. Thus we need to deploy multiple cloud storage with more redundancy so that we can retrieve the data from other surviving clouds during the failure.

Permanent failure: Permanent cloud failure is the one where if the cloud is failed; the data on the cloud will be lost permanently. This says that the permanent failure is more disastrous than transient failure. Though the permanent cloud failure is rare, there are many cases due to which they are still possible:

- *Data center may fail because of disasters.* AFCOM [48] found that many data centers are not prepared for disasters. For example, 50% of the cloud services have no plans regarding the damage repairs after the happening of damage. It was reported that earthquake and tsunami in northeastern Japan in March 11, 2011 knocked out several data centers there[48].

- *Data loss and data corruption.* There are many examples where a cloud may accidentally lose some data[12],[40],[58]. Example, In Magnolia [40] half terabyte of data is lost.

- *Malicious attacks.* The basic way of providing the security for data is to encrypt it before outsourced and put on cloud. If the outsourced data is attacked by virus or malware data is corrupted, which means though the data is encrypted confidential outside, the data inside is not useful. According to the study of AFCOM [48], 65 percent of data centers have no plan or procedure to deal with cyber-criminals.

Since the permanent cloud failure is not like transient failure where the cloud never returns back to normal, the data will be lost and is unrecoverable. So we need to repair it and reconstruct the lost data by making use of data available on other clouds to maintain the fault-tolerance. By the word repair, we mean to retrieve the lost data only from the surviving nodes and reconstruct the data to new cloud.

TABLE A

Example for transient failures occurring in different cloud services.

Cloud service	Failure reason	Duration	Date
Google Gmail	Software bug [24]	4 days	Feb 27-Mar 2,2011
Google Search	Programming error [38]	40 minutes	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

III. MOTIVATION OF FMSR CODES

Here in our concept, we are considering a distributed, multiple cloud storage setting from client's point of view, here the data is striped over multiple cloud providers. We are proposing a proxy-based design [1], [30] which interconnects the multiple cloud providers as shown in figure 1 (a). The proxy layer acts as an interface between client application and the clouds. If any cloud fails permanently, the proxy starts the proxy operation as shown in figure 1 (b).

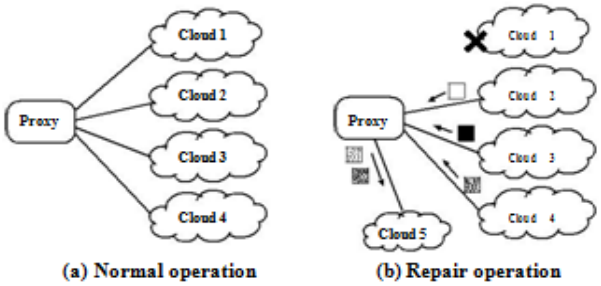

Fig.1. Proxy-based design for multiple-cloud storage: (a) normal operation, and (b) repair operation when Cloud node 1 fails. During repair, the proxy regenerates data for the new cloud.

As you can see from the figure, the proxy takes the required data pieces from other surviving clouds, reconstructs the lost data and write it to new cloud. The repair operation doesn't involve any direct interactions between the clouds.

Here we consider fault-tolerance based on the type of MDS(Maximum Distance Separable) codes. A given file of size M is divided into equal size native data chunks, this is later linearly combined to form chunks of code. When an MDS codes with (n, k) is used, the code chunks are then distributed over n (greater than k) nodes, where every storing chunks is of total M/k, such that the original file object may be reconstructed from the chunks contained in *any* k of the n nodes. This gives the opportunity to tolerate failures of any n-k nodes. This feature is the property of MDS. The main feature of FMSR codes is that the lost data chunks are reconstructed without downloading or reconstructing the whole file which means that we download very less file.

Our paper considers multiple cloud settings with two levels of reliability: 1)Fault-tolerance 2)Recovery. Let us assume that multiple cloud storage is double-fault tolerance for example., RAID-6 and it provides data availability under transient unavailability of maximum two clouds. So we set k=n-2. Hence the client can always access the data until two clouds fail transiently, or due to any connectivity problem. Secondly, we consider single-fault recovery in multiple cloud storage, which tell that permanent cloud failure is less frequent**.** The main objective of our project is to minimize the repair burden during storage during data migration over cloud for permanent single cloud failure.

The amount of outbound data being downloaded from the other surviving clouds during the single-cloud failure recovery is defined as repair traffic. We try to minimize the repair traffic for cost-effective repair .The inbound traffic is not considered (i.e., the data that is been written to a cloud), as it has no charge for many cloud providers (see Table 3 in Section 6).

Now we study the repair traffic involved in different schemes by an example. Suppose a file has to be stored of size M on four clouds, each cloud is viewed as a logical storage node. Let us now first consider conventional RAID-6 codes, which are double-fault tolerant. Based on the Reed-Solomon code [52] we consider a RAID-6 code implementation, as shown in Figure 2(a). Here, we divide the file of M 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 the Node 1 is down now, then the proxy must download same number of chunks as in the original from other two nodes (e.g., B and A+B from Nodes 2 and 3, respectively). Then the surviving nodes, reconstructs and stores the lost chunk X on the new node. Hence it can be concluded as The total storage size is 2M , while its repair traffic is M.

To reduce the repair traffic we Regenerating codes. The exact minimum-storage regenerating (EMSR) codes [57] is one among class of regenerating codes. EMSR codes maintains the storage similar in size as in RAID-6 codes, so as to reduce the repair traffic; the storage nodes send encoded chunks to the proxy. Figure 2(b) illustrates the double-fault tolerant implementation of EMSR codes. The file to be uploaded is divided into four chunks, as shown in the figure and accordingly allocation of native and code chunks is done. If suppose Node 1 is down then to repair it, each surviving node sends the XOR summation of the data chunks to the proxy, which then reconstructs the lost chunks. The storage size of EMSR codes is 2M (same as RAID-6 codes), while the repair traffic is 0.75M which is 25% of saving (compared with RAID-6 codes). As the nodes will generate encoded chunks during repair, EMSR codes leverage the notion network coding [2].

We now consider the double-fault tolerant implementation of FMSR codes as denoted in Figure 2(c). A file is divided into

four native chunks, and then constructs eight distinct code chunks P1… P8 obtained by performing different linear combinations of the chunks. Each code chunk has the same size M /4 . For recovery process , any of the two nodes can be used the original four native chunks. If Suppose Node 1 proxy retrieves one code chunk from each of the surviving node, so it

3. let us assume that single-fault tolerance (i.e., k = n − 1) and single-fault recovery, according to the theoretical results of [16], it is shown that traditional RAID-5 codes [45] have the same data redundancy and same repair traffic as FMSR codes.

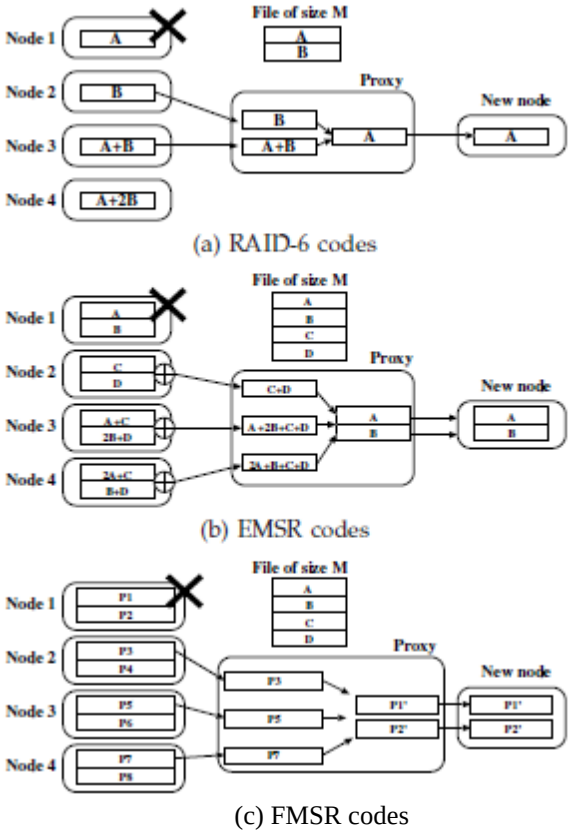

Fig. 2. Examples of repair operations in different codes with n = 4 and k = 2. All arithmetic operations are performed over the Galois Field GF(2^8).

Three code chunks each of size M/4 is 'downloaded. Then the proxy regenerates two code chunks $P1'$ and $P2'$ by performing of three code chunks. Note that $P1'$ and $P2'$ are still linear combinations of the native chunks. The proxy then writes $P1'$ and $P2'$ to the new node. The storage size in FMSR codes is 2M (it is as in RAID-6 codes), even though the repair traffic is 0.75M , which is similar to that of the EMSR codes. A FMSR codes is that encoding is not performed during repair of nodes.

In order to generalize double-fault tolerant we make use of FMSR codes for n storage nodes, a file of size M is divided into 2(n - 2) native chunks, and it is used to produce 2n code chunks. Then two code chunks of size $M/2(n-2)$ will be stored in each node. Thus, the total storage size is $Mn/(n-2)$. In order to repair a failed node, downloading of one chunk from each of the other n−1 nodes, so the repair traffic is $M(n-1)/2(n-2)$. In contrast, for $Mn/(n-2)$ RAID-6 codes, the total storage size is also , when the repair traffic is having the value as M . Whenever n is large, the FMSR codes can save the repair traffic by close to 50%.

Note To access a single chunk of the file, download and decode method is necessary. FMSR codes are non-systematic, as they keep only code chunks but not native chunks. The complete file for that particular chunk. This is opposed to systematic codes (as currently existing in the traditional RAID storage), where native chunks are placed. FMSR codes are acceptable for long-term archival applications, the read frequency is typically low and also, to restore backups, it is good to retrieve the entire file rather than a particular chunk [14].

This paper considers Reed-Solomon codes the baseline for RAID-6 implementation .This repair method involves reconstruction of complete file first, and can also be applicable for all erasure codes in general. Recent studies [35], [62], [63] proves that data reads can be reduced specifically for XOR-based erasure codes. Consider an example, reading of the data can be reduced by 25% compared to that of reconstructing the whole file [62], [63]. Although such approaches can achieve FMSR codes, which can save up to 50% of repair traffic, the use of efficient XOR operations can also be practical interest.

FMSR CODE IMPLEMENTATION

Let us now present the details for implementing FMSR codes in multiple-cloud storage. FMSR codes has three operations on a particular file object as follows:- (1) file upload; (2) file download; (3) repair. Each cloud repository is viewed as a logical storage node. In a thin-cloud interface [60], such that the storage nodes (i.e., cloud repositories) requires only to support basic read/write operations. Thus, we expect that our FMSR code implementation is compatible with today's cloud storage services.

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

4.1 Basic operations

4.1.1 File Upload:

To upload a file F , the first step is to divide the file into k(n - k) equal-size chunks, indicated 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 using a linear combination of the k(n − k) native chunks. 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(2^8). We call a row vector of EM an encoding coefficient vector (ECV) , which contains n(n − k) elements. We make use of ECVi to denote the ith row vector of EM. We compute each Pi by the product of ECVi and all the native k(n−k) αi,j Fj for chunks $F1, F2 \dots, Fk(n-k)$, i.e., Pi = $\sum_{j-1}^{k(n-k)} \alpha i, jFj$ for i=1,2,..,n(n-k).where all arithmetic operations are performed over GF(2^8).The code chunks are evenly stored in the n storage nodes, each having (n-k) chunks and we store the complete EM in a metadata object that is then replicated to all storage nodes (see Section 5). There are many ways of constructing EM, as long as it passes two-phase checking (see Section 4.1.3).Note that the implementation details of the arithmetic operations in Galois Fields have been extensively discussed in [25].where all arithmetic operations are performed over GF(2^8).

4.1.2 File Download:

To download a file, first download the corresponding metadata object that contains the ECVs. From n storage nodes chose any k nodes , and download the k nodes from code chunks. 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 satisfied, then as the according to the definition, the inverse of the square matrix should exist. Later the inverse of the square matrix along with the code chunks and obtain the original k(n − k) native chunks is multiplied. The idea obtained here is that we treat FMSR codes as standard Reed-Solomon codes, and we describe the technique of creating an inverse matrix to decode the original data , in the tutorial [46].

4.1.3 Iterative Repairs:

Let us now take an example of the repair of FMSR codes for a file F for a permanent single-node failure. Given that FMSR codes regenerates different chunks in each repair, one of the challenge is to ensure that the MDS property is achieved even after iterative repairs. In contrast to regenerating the exact lost chunks as in RAID-6, which guarantees the invariance of the stored chunks. A two-phase checking heuristic is proposed as follows. Suppose that the $(r-1)th$ repair is successful. and now let us consider how to handle the rth repair for a single permanent node failure (where r ≥ 1). We now first check if the new set in all storage nodes satisfies the MDS property after the rth repair. In addition to that we also check whether any other new set of chunks in all the existing storage nodes still achieve the MDS property after the(r + 1)th repair, should another single permanent node failure occur (we call this the repair rMDS) property). Let us now describe the rth repair as follows.

Step 1: The encoding matrix from a surviving node has to be downloaded. The encoding matrix specifies the ECVs for constructing all code chunks through linear combinations of native chunks. These ECVs are used later for two-phase checking. Since EM is embedded in a metadata object that is replicated, we can simply download from one of the surviving nodes the metadata object.

Step 2: Now select one ECV from each of the n − 1 surviving nodes. Each of ECV in EM corresponds to a code chunk. We now pick one ECV from each of the n -1 We call those selected ECVs to be ECVi1 , ECVi2 ,…., ECVin−1

Step 3: obtain a repair matrix. We construct an (n−k)×(n−1) 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(2^8). Note that the idea of generating a random matrix for reliable storage is consistent with that in [49].

Step 4: Calculate the ECVs for the new code chunks and reproduce a new encoding matrix. now multiply RM with the ECVs selected in Step 2 to construct (n-k) new ECVs, Denoted by ECVi′ = $\sum_{j-1}^{n-1} \gamma i, jECVi$ for i=1,2,…,(n-k). Then reproduce a new encoding matrix, denoted by EM', that is formed by substituting the ECVs of EM of the failed node with its corresponding new ECVs.

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

Step 6:Here Downloading of the actual chunk data and regenerating the new chunk data. If the two-phase checking that is shown in the Step 5 succeeds, then we proceed with the process to download the n − 1 chunks that will correspond to that of the selected ECVs shown in the Step 2 from the n − 1 surviving storage nodes to NCCloud. Also, we are using the new ECVs computed in Step 4, we are also regenerating the new chunks and upload them from NCCloud to a new node.

Remark: we can simplify the complexity of the two-phase checking with that of the proposed FMSR code construction that is being done in our recent work [28]. And also our proposed construction will specify the ECVs to be selected in Step 2 deterministically, and that will tests their accurty (i.e., satisfying both the MDS and rMDS properties) by checking it against that a set of inequal-ities shown in the Step 5. This will also reduces the complexity present in each of the iteration along with the number of iterations (i.e., number of times the Steps 2-5 are being repeated) in process of generating a valid EM'. According to our present implementation of the NCCloud also includes the proposed construction. We also refer the readers to [28] for more details of the proposed construction.

4.2 Analysis

The process of checking the rMDS property in each repair is very much necessary for the purpose of maintaining the MDS property after all the iterative repairs done . We now show through a counter-example that if a repair is checking only the MDS property but without checking the rMDS property, then in such cases the MDS property will be lost in the very next repair. We are also showing it through the simulations that our two-phase checking can even sustain many iterations of repairs in more than the general cases.

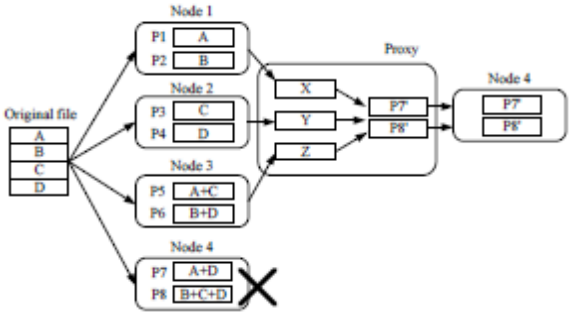

Fig. 3. Counter-example: code chunks that satisfy the MDS property but not the rMDS property

4.2.1 A Counter-Example

Here counter example shown in the figure 3is considered to show the importance of the rMDs property=4 and k=2 is same as that being the described in the figure2(c) .If suppose a linear combination is done for the code chunks P1, . . . , P8 from the native chunks A,B,C and D as shown in the figure 3,and these linear combinations are similar to that of te shortened even odd code IN[62].Verification of whether the code chunks P1, . . . , P8 satisfy the MDS properties easy i.e, reconstruction of the native chunks A,B,C and D can be done using the four chunks from any two nodes .But,we do not check whether it satisfies the RMDS property(as they do not satisfy, we shall see it later).

Now consider that node FMSR codes,if the basis of the FMSR codes fails ,ne chunk from each nodes 1,2,and 3 the repair selects(denote the chunks X,Y, and Z)and using them regenerate the new code chunks P7' AND P8',that will be stored in a new node(that will be stored in a new node 4).Totally there are 2^3 = 8 possible selections of {X,Y,Z}.

Lets consider one possible selections of{X,Y,Z}.There lets us use one possible selection{P1, P3, P5}. Now the new code chunks become E

$P7' = \gamma 1,1P1 + \gamma 1,2P3 + \gamma 1,3P5,$

$P8' = \gamma 2,2P1 + \gamma 2,2P3 + \gamma 2,3P5,$

where $\gamma i,j$ ($i = 1, \ldots, n - k$ and $j = 1, \ldots, n - 1$) are some random coefficients used to generate the new code chunks. Then we have

$P7' = (\gamma 1,1 + \gamma 1,3)A + (\gamma 1,2 + \gamma 1,3)C,$

$P8' = (\gamma 2,1 + \gamma 2,3)A + (\gamma 2,2 + \gamma 2,3)C .$

Since P1 = A and P2 = B, we cannot reconstruct the native chunk D from P1, P2, P7', P8'. The MDS property is lost because the chunks of Nodes 1 and 4 cannot be used to reconstruct the native chunks. Thus, the repair fails with this selection of chunks.

The chunks of nodes 1 and 4 cannot be sed to reconstruct the chunks hence the MDS property is lost.in the selection of the chunks the repail fails due to this reason.

Similar kind of the reasoning can be applied to the other possible selections of the chunks .The selection of the eight possible selection of the chunks along with the set chunks that cannot be used anymore to rebuild the original file is shown in the table 2.

TABLE 2

Eight possible selections of chunks from surviving nodes for generating P70 and P80, along with the corresponding set of chunks that will fail to reconstruct the file.

X, Y, Z	Set of chunks that cannot rebuild the file
P1, P3, P5	P1, P2, P70, P80 (Nodes 1 and 4)
P2, P3, P5	P1, P2, P70, P80 (Nodes 1 and 4)
P1, P4, P5	P3, P4, P70, P80 (Nodes 2 and 4)
P2, P4, P5	P5, P6, P70, P80 (Nodes 3 and 4)
P1, P3, P6	P5, P6, P70, P80 (Nodes 3 and 4)
P2, P3, P6	P3, P4, P70, P80 (Nodes 2 and 4)
P1, P4, P6	P1, P2, P70, P80 (Nodes 1 and 4)
P2, P4, P6	P1, P2, P70, P80 (Nodes 1 and 4)

MDS property after the repair. The above counter-example shows on checking the MDS property only but not on how the rMDS property can lead to a failed repair.

Simulations

Simulations is conducted to justify the rMDS property that can make an iterative repairs sustainable. Evaluation is done using simulations it is the overhead of our two -phase checking(steps from 2 to 5 of the repair).Here our simulation is done on the 2.4GHz CPU core. Firstly ,for different values of new consider the multiple rounds of repair and the argue in the addition to checking of the MDS property, and the rMDs property checking is required for the iterative repairs. Particularly, In each round ,we select a node randomly that has to be failed and then repairing the failed node . We consider a repair is bad if the loop of Steps 2 to 5 in two-phase been repeated over a threshold number of times but still no suitable encoding matrix is being obtained. In this simulation, we are varying the threshold of the number of loops for identifying a bad repair. Maximum of 500 rounds of repairs being carried out, and stop once bad repair. We are not considering the construction of [28] in this part of simulations to study the effects of the baseline

Figure 4 shows the number of rounds of repair that can be sustained when the rMDS property is whether checked or is not . It shows checking the or is not . It shows checking the rMDS property provides us to sustain more rounds of repair before seeing a bad repair. For example,if suppose that we set the thresholdas 20 loops. Then repair can be sustained for 500 rounds, for different values of n (number of nodes) by checking the rMDS property, but a bad repair quickly (e.g., in

3 rounds of repair for n = 10) if we don't check the rMDS property. Next we evaluate through simulations the time of the two-phase checking, with the proposed FMSR code construction[28] that reduces the complexity. In each round of repair, randomly pick a node to be failed and carry out the repair operation. Then carry out the two-phase checking (i.e., Steps 2 to 5), and measure the time required to generate an encoding matrix that satisfies both the MDS and rMDS properties.

Figure 5 plots the cumulative time of two-phase checking for 50 rounds of repaig in log scale) for n =4

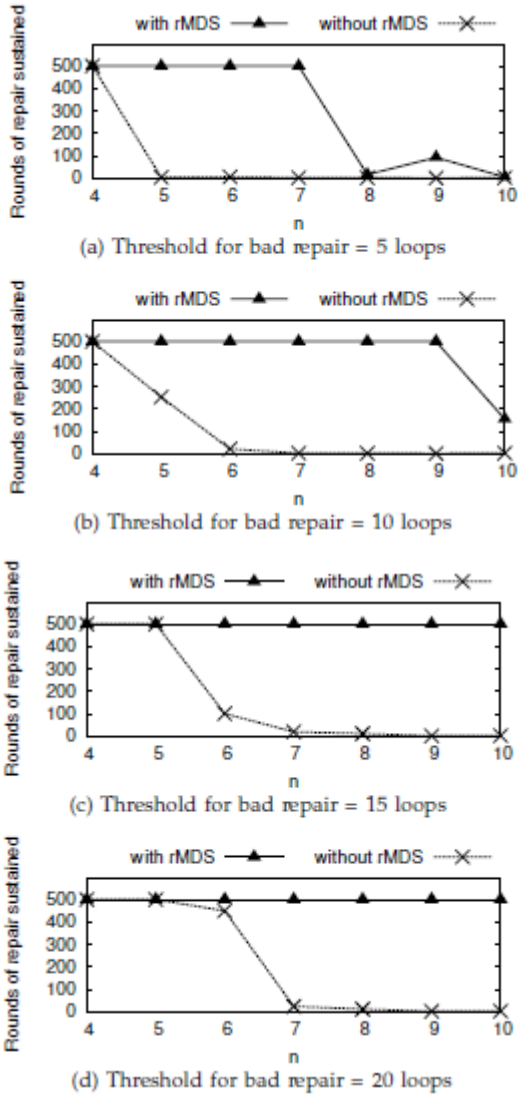

Fig. 4. Number of rounds of repair sustainable without seeing a bad repair. n = 16. The checking process done takes the negligible time compared to that of the actual repairs of even a 1MB file (see Section 6.2.2). consider for example, when n = 10, it takes only 0.02s to carry out 50 consecutive repairs (around 0.0004s per repair); even when the value of the n = 16, it takes only 0.1s to carry out 50 consecutive repairs (around 0.002s per repair). Here observe that the range of n we consider following the stripe sizes used in many practical storage systems [47]. In order to reduce it further reduce the overhead, we can pre-compute the newly encoding coefficients for any possible node failure offline when the system is running as normal, and keep the obtained results to prepare for the next repair.

4.2.3 Reliability Analysis

Following the studies that is evaluating the reliability of various erasure codes and replication (e.g., [20], [31],

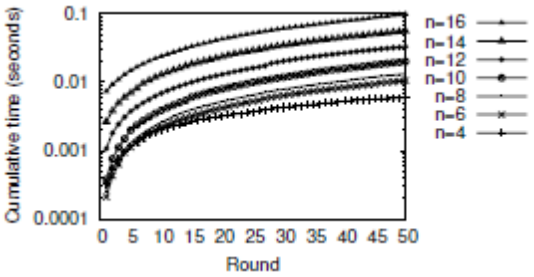

Fig. 5. The cumulative time needed by the checking phase (plotted in log scale) in 50 consecutive rounds of repairing from n = 4 to n = 16.

Fig. 6. Markov model for double-fault tolerant codes.

we are comparing the reliability of FMSR codes and traditional RAID-6 codes with respect to the different failure rates with the help of the mean-time-to-data-loss (MTTDL) metric, which is defined as the expecting time which has been elapsed till the original data will be unrecoverable. When MTTDL is not that effective to identify the qunatity the real reliability [26],in such case it remains a more adopted reliability metric of the storage community and we
Make use of it only for the comparative study of different coding schemes with different repair performance also.

MTTDL is being solved using the Markov model. Figure 6 it shows the Markov model which is suitable for the double-fault tolerant codes (i.e., k = n − 2), in which state i (where i = 0, 1, 2, 3) is denoting the number of the failed nodes in a storage system. State 3 indicates that there failed nodes are more than two in number and the data which is permanently lost. We are computing the MTTDL which is the expected time to move from state 0 (i.e., all nodes which are normal) to state 3.

Here we are making an assumptions in our analysis. For the sake of the simplicity, we are assuming that node failures and repairing the process are indepen-dent events which follow an exponential distribution. But this assumption is imperfect in general [54], but which makes our analysis tractable and which has been used in previous studies [20], [31], [53]. Let it λ be the node failure rate (i.e., 1/λ is the expected time in which failure of a node occurs). hence, the transition rate from the state i to state i +1 is (n − i)λ, where i = 0, 1, 2. And also, consider μ_1 and μ_2 be the repair rates for that of the single-node and double-node failures, respectively. We are assuming that the transfer network between the *surviving* nodes and that of the proxy is one of the major bottleneck (see the Section 3 for this formulation) and finding

the resulting repair rates. Assume S being the size of the data stored in each of the node (i.e., the total amount of original datathat is being stored is (n − 2)S) and B being the network capacity between the surviving nodes and that of the proxy. Now, considering the repair of a single-node failure. As shown in above Section 3, for FMSR codes, the repair traffic is calculated as $(n-1)S/2$ and hence $\mu_1=2B/(n-1)S$ RAID-6 codes, the repair traffic is (n − 2)S and hence $\mu_1=B/(n-1)S$, For the repair of a double-node failure, both

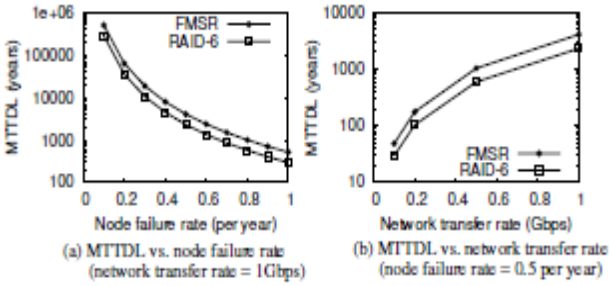

Fig. 7. MTTDLs of FMSR codes and RAID-6 codes are (plotted in log scale) when the value of n = 10 and k = 8.

Of the FMSR codes and that of the traditional RAID-6 codes will resort to the conventional approach and also the reconstruction of the lost data by downloading of the total amount of original data (i.e., (n − 2)S) from that of the remaining k = n−2 presting nodes. Both of them have $\mu 2 = B/(n-1)S$ We must now evaluate the MTTDLs of the FMSR codes and that of the traditional RAID-6 codes for some of the specific parameters. Suppose that we fix n = 10, k = 8, and S = 1TB. Figure 7(a) is showing the MTTDLs for different values of λ from 0.1 to that of the 1 (in units per year) when the value of the B = 1Gbps, while Figure 7(b)is showing the MTTDLs for different values of B from 0.1 to 1 (in units of Gbps) when the value λ = 0.5 is per year. Based on the settings that we have done, the MTTDL of FMSR codes is upto 50% to 80% lengthier than that of the traditional RAID-6 codes because of their higher repair rate for a single-node failure. Considering the example, with λ = 0.5 per year and B = 1Gbps, the MTTDL of the FMSR codes is 76% longer.

4.3 Discussions

We here consider the several open issues of the current design of FMSR codes, and then we would give them as future work.

Generalization of FMSR codes. Here we presently consider only an FMSR code implementation with the double-fault tolerance (i.e., k = n − 2). Its accuracy is being proven in our recent work [28]. When the value of the double-fault tolerance is the that of the default setting of current enterprise storage systems (e.g., 3-way replication in the GFS [22]), it is not clear on how to generalize the FMSR codes for the different (n, k) values. In ad-dition to this, in practical cloud storage systems [31] while single-node failures are the most common failure patterns , it is most interesting to study like how to generalize the FMSR codes to support the most effective repairs of concurrent node failures.

Study of different reliability metrics. In this Section 4.2.3, we are comparing the reliability of FMSR codes and the conven-tional RAID-6 codes for the different failure rating using that of the MTTDL metric. The open issue for the modeling the failure rate of a cloud repository. In the future works, we can also plan to the conduct further analysis regarding the reliabilty using the more effective metrics [26].

Degraded reads. When the process of reading the original data in failure mode is done then, we perform degraded reading, in which we are reconstructing the lost data of a failed node from the available data on the other surviving nodes. In FMSR codes, we are always downloading the same amount of original data by connecting to of the any k nodes (refer Section 4.1.2); in case of traditional RAID-6 codes, the original amount of data is retrieved in order to recover the lost data. Thus, traditional RAID-6 and FMSR codes retrieve the equal amount of data in degraded reads, when FMSR codes have higher computational overhead in decoding (refer Section 6.2.1). Recent studies [31], [35], [53] improve the degraded read performance for erasure-coded data. we do not consider degraded reads in this work since FMSR codes are designed for long-term archives that are rarely read.

IV. FTCLOUD DESIGN AND IMPLEMENTATION

FTCloud is implemented as a proxy that connects user applications and clouds. It is designed on top of three layers. Firstly, the ***File System Layer*** which makes FTCloud as a mounted drive that can be easily interfaced with user applications. Secondly, the encoding and decoding functions are taken care by ***Coding Layer.*** Lastly, the read/write requests with clouds are dealt by ***Storage Layer***.

Every file is attached to a *metadata* object that is replicated at each repository. The metadata object includes the details of file and the information related to coding.

Java is the key language to FTCloud's implementation and the coding part is implemented through C. The file system layer is constructed on FUSE [21]. Both RAID-6 and FMSR codes are implemented by coding layer. The RAID-6 code is implemented based on the Reed-Solomon code [52] (as shown in Figure 2(a)) for baseline evaluation. zfec [65] is used to implement the RAID-6 codes. For fair comparison we make use of zfec's optimization for implementation of FMSR codes.

Multiple chunks that are generated by FMSR codes are stored on same repository which causes request cost overhead. In order to reduce it, aggregation of those chunks is performed before upload .Hence FMSR codes keep only the aggregated chunks per file object on each cloud like in RAID-6 codes. While retrieving a particular chunk, its offset within the combined chunk is calculated and a range GET request is issued.

FTCloud is deployed in one or more machines. Inorder to prevent simultaneous updates on same file we use ZooKeeper[32] that implements a distributed file-based shared lock. Pre-liminary evaluations is conducted in a LAN environment andthe overhead that could be caused by ZooKeeper is observed to make sure its minimal. Our focusis on deploying FTCloud on a single machine, and is mounted as a local file system.

V. EVALUATION

FTCloud prototype is used to evaluate RAID-6 codes as well as FMSR codes in multiple cloud storage. The data retrieval with two cloud failures is allowed by focusing on settingk = n − 2 for different values of n.

TABLE 3
Monthly price plans (in US dollars) for Amazon S3 (US Standard), Rackspace CloudFiles and Windows Azure Storage (North America and Europe), as of May, 2013.

	S3	RS	Azure
Storage (per GB)	$0.095	$0.10	$0.095
Data transfer in (per GB)	free	Free	free
Data transfer out (per GB)	$0.12	$0.12	$0.12
PUT,POST (per 10K requests)	$0.05	Free	$0.001
GET (per 10K requests)	$0.004	Free	$0.001

Our goal is to discover the practical usage of FMSR codes in multiple cloud storage. There are two parts in evaluation. Firstly, the comparison of the monetary costs when RAID-6 and FMSR codes are used is performed. This is based on the price plans of today's cloud providers. Secondly, response time performance of FTCloud prototype is evaluated on both local and commercial cloud provider.

Summary of evaluation results.
Our summary goes as below. Main importance is given to the monetary cost advantage of using FMSR codes over RAID-6 codes, on the other hand maintaining good response time performance. In case of monetary costs in normal operations, both RAID-6 and FMSR codes costs almost the same in operation of storage, and in the operation of repair, FMSR codes is ahead of RAID-6 codes because it saves a good amount of transfer comparatively.When it comes to response time, both FMSR and RAID-6 codes have comparable response time performance (within 5%) when it is deployed on a commercial cloud (Azure).The transmission performance of the Internet determines theresulting response time.

6.1 Cost Analysis

6.1.1 Repair Cost Saving

Let us first analyze saving the costs due to repair. Table 3 includes the price plans in each month for three major providers as of May 2013. We analyse the cost based for more than 1GB/month data transfer within a limit of 1TB/month of data usage.

Looking at the analysis in Section 3, we could save 25-50% of the traffic of download during storage repair. The size of the storage and the number of chunks generated per file object is same in both RAID-6 and FMSR codes. In the analysis, we have neglected two considerations in practicality: One, the size of metadata (Section 5). Two, the number of requests that are issued while repair. We prove that our argument of neglecting these consideration, also argue that the optimized calculations based only on file size are sufficient for real-time applications.

Metadata size: According to ourimplementation, the size metadata for FMSR codes is within 160 bytes when n = 4 and k = 2, no matter what the file size is. When n is greater, example when n = 12 and k = 10, the metadata size is

TABLE 4
Tiered monthly price plans (in US dollars) for both Amazon S3 (US Standard) and Windows Azure Storage (North America and Europe), as of May 2013.

Storage (per GB)	Data transfer out (per GB)
$0.095 (First 1TB/month)	$0.12 (First 10TB/month)
$0.08 (Next 49TB/month)	$0.09 (Next 40TB/month)
$0.07 (Next 450TB/month)	$0.07 (Next 100TB/month)
$0.065 (Next 500TB/month)	$0.05 (Over 150TB/month)
$0.06 (Next 4000TB/month)	

still inside the range of 900 bytes. Main aim of FTCloud is to provide backups for long time(see Section 3), and to integrate with other applications used for backup. In order to save the overhead of processing, the backup applications that are existing (e.g., [19], [60]) combines small files into a larger data chunk.For instance, 4MB is the chunk size that is created by the default setting for Cumulus [60]. Hence, the overhead of metadata size is made negligible. Since the amount of file data stored by both RAID-6 and FMSR codes is same, they have much similar costs.
of storage in normal usage.

Number of requests: Observations of Table 3 says that, it is being charged for requests by some cloudproviders. The number of requests when retrieving data during repair is different for RAID-6 and FMSR codes. Suppose a file object of size 4MB is stored with n = 4 and k = 2. While repairing, RAID-6 code retrieves two chunks and FMSR code retrieves three chunks(see Figure 2). The overhead of cost due to the issue of GET requests for RAID-6 is equal to 0.171 percent and for FMSR codes is 0.341%. Hence it is an insignificant 0.17% increase.

6.1.2 Case Study

We now provide the conclusions for our analysis of cost using an enterprise use case. Our analysis is built on the case of Backupify, who is a cloud backup solution provider, founded in 2008 and used to store backups of amount that is in the range of terabytes to petabytes o S3 and Glacier . To make our analysis simple, let's assume that Backupify stores backups of worth 1PB in the cloud. And also the data is replicated over 10 clouds, with n = 10 and k = 8, it causes a

redundancy overhead of 25%. As we have been arguing above, both RAID-6 and FMSR codes causes same storage cost and data transfer cost, but FMSR code causes less repair cost comparatively to RAID-6 codes. Precisely, FMSR codes saves the cost by a percentage of 1–((n-1)/2(n-2))(see Section 3), equal to 43.75%. In the following, considering two cost models.

Regular-cost storage model.

If terabytes of data needs to be stored, the pricing scheme used by cloud storage providers is a tiered scheme, which allows higher usage at lower rates. Table 4 poses a simplified tiered pricing scheme and used by both Amazon S3 and Windows Azure. This tiered scheme is used for most of our cost calculation.

Coming to our case, the amount of data that is stored is 1.25 petabytes, and $86,851 is the storage cost to be paid monthly for both RAID-6 and FMSR codes. Suppose a cloud repository fails permanently and then we run the operation of repair, then the amount of data downloaded by RAID-6 code and FMSR code is 1PB and 0.5625PB respectively. Hence, the repair cost for RAID-6 codes is $56,832, and that of FMSR codes is $33,894. Showing that FMSR code saves cost amount of $22,938.

Low-cost storage model.

We say that the monthly storage cost can be exceeded by the repair cost if a storage model of low cost is used alternatively. For example, Amazon Glacier [5] which uses the same data price of S3, charges a flat rate of $0.01 per GB of stored data, referring to the table 4, this is much cheaper than S3. But the drawback of using Amazon instead of S3 is that it consumes a longer time for the restore operation and is also more expensive. And also, if more than 5% of stored data is to be restored, Amazon charges a restore fee of $0.01 per GB on monthly basis.

According to this cost model, the monthly storage cost is reduced to only $13,107 for both RAID-6 and FMSR codes. However, the cost of repair for RAID-6 codes is $66,662, and that of FMSR codes is $39,137. Hence FMSR codes save cost by $27,525.

We cannot say that the annual saving that is brought by the reduction in repair cost to be purely measured by the failure rate of a cloud storage repository, we note that in the last few years ,permanent data loss of varying degrees has occurred in cloud storage since its adopted by the masses popularly (e.g., [12], [40], [58], [64]). If we calculate that if complete repairs have to be made for every two year(average), this results over $10,000 of saving annually in our case.

Concluding, we observe that in spite of cloud failures being rare, the monetary benefit gained by usage of FMSR codes in events of repair that is unexpected is important. We haven't showed another consideration in practicality, which is data accumulation. According to our case study we assume that the amount of data stored is constant. But during times like, when customers are producing new data daily or when the number of customers using the storage service is increased, the amount of data is no more constant but grows along with the time in reality. As time passes, this larger data accumulation results in archive of larger size, thus making our monetary advantage in repair cost more emphasized.

6.2 Response Time Analysis

Our FTCloud prototype is deployed in real environments. The three basic operations that stands as a basis for us to evaluate the response time performance are, file upload, file download and repair, in two scenarios. In the first part, the time taken by the different FTCloud operations is analyzed in detail. In order to reduce the effects caused by network fluctuations, it is performed on a local cloud storage test bed. In the second part, we evaluate how actually FTCloud performs when deployed on a commercial cloud. Forty runs is the average of all results. Since our assumption, that the coefficients for repair are offline generated (see Section 4.2.2), we do not take the time taken by two-phase checking into account for in the repair operation. Since the time consumed for checks is less comparatively to the overall operation of repair, it has limited impact on our results as shown in Section 4.2.2

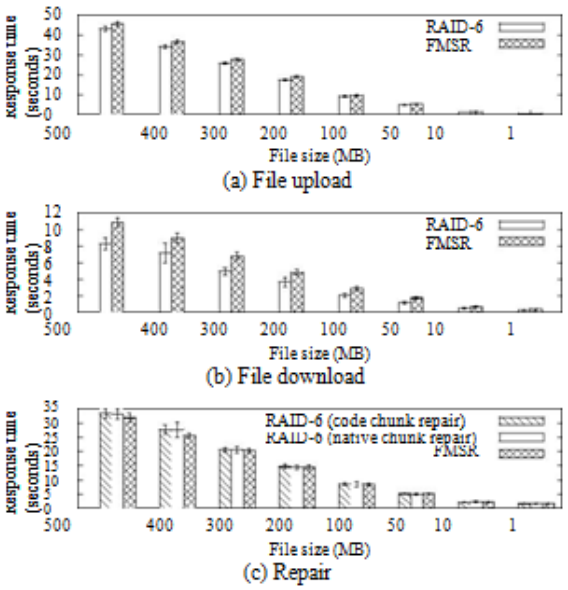

Fig. 8. Response times of FTCloud operations when n=4 and k = 2.

6.2.1 On a Local Cloud

OpenStackSwift 1.4.2 [42] is the basis for object-based storage platform on which the experiments on local cloud are carried out. FTCloud is mounted on a machine which consists of Intel Xeon E5620 processor with 2.4GHz speed and RAM of size 16GB. This machine is bridged to an OpenStack Swift platform which is attached to a number of storage servers, and that each server would have Intel Core i5-2400 and 8GB RAM. We create virtual cloud repositories by creating (n +1) containers on Swift, in which each container is equivalent to a cloud repository (out of them one is a node used as a spare during times of repair). Two experiments are

conducted on the local cloud. In the first experiment, we compare RAID-6 and FMSR codes when the values of n and k are 4and 2 respectively with file size being varied. Whereas, in the second experiment we perform the same comparisonbetweenRAID-6 and FMSR codes but this time with different values for n and k and file size being fixed.

During the first experiment, the response times is tested under the three basic operations, i.e., file upload, file download, and repair operations of FTCloud with values as n = 4 and k = 2. We make use of eight files from that are generated randomly for 1MB to 500MB as the data set. The path of a repository that is chosen is set to a non-existent location to simulate a node failure in repair. We must notice that when it comes to repair, there are two types for RAID-6, and the type is selected based on the fact that whether the failed node contained a native chunk or a code chunk. Figure 8 shows the response times of all three operations versus the file size.

During the second experiment, the file size is fixed at 500MB and now the response time is tested under the three operations again under four different pair of values for n and k as, n = 4, k = 2 and n = 6, k = 4 and n = 8, k = 6 and n = 10, k = 8. Figure 9 poses the results of response time, in which each is decomposed into many key parts.

Figures 8 and 9 show that the response time for RAID-6 codes is comparatively less than FMSR codes in operations of file upload and

download, no matter what the values of n and k are. Using Figure 9, we show the overhead of FMSR codes over RAID-6.FMSR codes show similar data transfer time as that of RAID-6 while uploading and downloading, this is because of having the same MDS property in them. However, there is a significant over head of encoding/decoding in FMSR codes over RAID-6 codes. For instance, in the case of n = 4 and k = 2, while uploading a 500MB file,RAID-6 codes consumes 1.53s to encode, whereas FMSR codes consumes 5.48s; in the operation of downloading the 500MB file, there is no requirement of decoding in the case of RAID-6 codes as there is availability of native, but FMSR codes consumes 2.71s for the decoding process. This increase in difference is due to n and k

While on the other hand, there is a merit of FMSR codes because the response time here is slightly less during the operation of repair. We must notice that the amount of data that is being downloaded by FMSR codes during repair is less. This is the main advantage of having FMSR codes. Example, to repair a file of size 500MB with n = 4 and k = 2, the time spent by FMSR codes is 4.02s in download and 5.04s is the time spent byRAID-6 codes.

RAID-6 codes may have less response time than FMSR codes when deployed on a local .But we think that the overhead of encoding/decoding in FMSR codes can be easily covered by the fluctuations in the network over the Internet, as we would discuss next.

6.2.2 On a Commercial Cloud

This experiment is conducted on a machine that includes an Intel Xeon E5530 2.4GHz CPU and RAM of 16GB size. This machine has the 64-bit Operating system, Ubuntu 9.10. We set the same values, n = 4 and k = 2, and repeat performing all the three operations as in Section 6.2.1 on four files that are randomly generated from 1MB to 10MB on top of Windows Azure Storage [13]. On Azure, we now try to create virtual cloud repositories by creating (n+1) = 5 containers. The same operations are run for both RAID-6 and FMSR codes and provide interval in order to reduce the effects of fluctuations in the network. We must notice that, Azure is the only provider which is being used here. But in actual usage, FTCloud is supposed to stripe data over different providers and locations. This is to provide better availability guarantees.

Figure 10 poses the results for different file sizes plotted with 95% confidence intervals. From the figure, we can see no same differences in response time between RAID-6 codes and FMSR codes under all the three operations. Also, FMSR codes consume 0.150s for encoding purpose and 0.064s for decoding a file of size 10MB (not reflected in the figures). This contributes roughly 3% to the total time of uploading and downloading which is 4.962s and 2.240s respectively. The 95% confidence intervals for the operation of upload and download are 0.550s and 0.438s respectively. Fluctuation in network plays a very vital role in calculating the response time. Finally to brief, we show that the performance overhead by FMSR codes is not significant over the implementation of RAID-6 code.

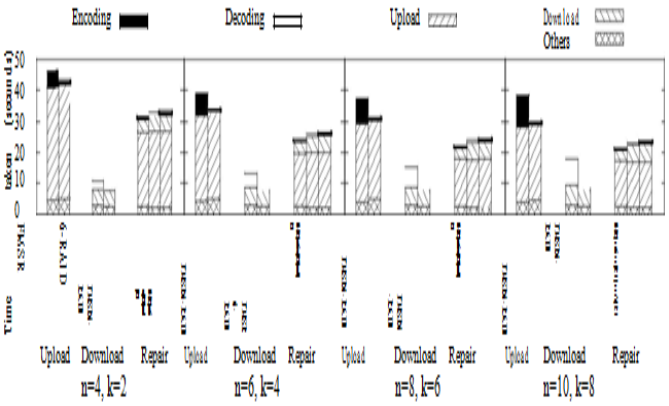

Fig. 9. Breakdown of the response time.

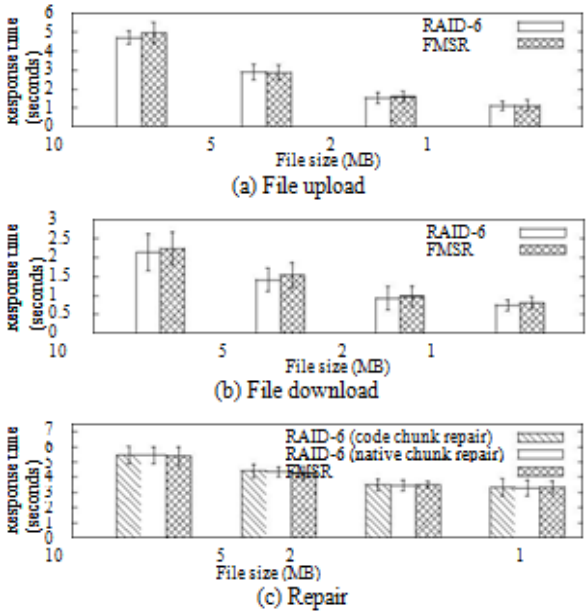

Fig. 10. Response times of FTCloud on Azure.

7 RELATED WORK

Let us now look at the work that is related in multiple-cloud storage and recovery during failure.

Multiple-cloud storage. We can find many systems that are proposed for multiple-cloud storage. Such as, HAIL [11], this system provides integrity and assures availability for stored data. Another such system is RACS [1],this system makes use of erasure coding in order to solve the vendor lock-in problem while switching from one cloud vendor to another. Here, the data from the cloud that is probably to fail is moved from it into a new cloud. But FTCloud does not include the failed cloud which is in repair. Vukolic´ [61] works using multiple clouds which are independent in order to provide Byzantine fault tolerance. DEPSKY [10] provides Byzantine fault tolerance, it does it by performing the combination of encryption and erasure coding for stored data. WE must notice that all the systems that are discussed above are built using erasure codes in order to provide fault tolerance. But our FTCloud is one step ahead because it includes regenerating codes with giving importance both fault tolerance as well as storage repair.

Minimizing I/Os. There are several studies that provide efficient failure recovery schemes for single node that reduces the amount of data that is read (or I/Os) for erasure codes based on XOR. For example, authors of work [62], [63] provide optimal recovery for specific RAID-6 codes and drops down the amount of data that is read by a percentage of around 25 for nodes of any number. OurFMSR codes can succeed in saving of 25% when there are four nodes, and when the number of nodes increases the savings is raised to 50%. According to work [35],in order to search for an optimal solution for recovery for arbitrary erasure codes based on XOR, it proposes an enumeration-based approach. In recent days, commercial cloud storage systems are having recovery efficiently. For example, Azure [31] and Facebook [53] are getting efficient recovery in them through the new builds of erasure codes which are designed with non-MDS. The overhead of storage is shifted for the purpose of performance using of[31], [53], and their emphasis of design is for computing of intensive data. Our main focus is the applications that are available for cloud backup.

Minimizing repair traffic.
Network coding is the basis for Regenerating codes [16]and they tend to provide reduction in the repair traffic among storage nodes. They also achieve the optimal movement between cost due to storage and repair traffic, and consists of two optimal points. One optimal point reduces the repair bandwidth with the condition that minimum amount of data is being stored by every node. This optimal point is referred to as the *minimum storage regenerating (MSR)* codes. The other optimal point allows every node to store more amounts of data to still reduce the repair bandwidth. This optimal point is referred to as *minimum bandwidth regenerating (MBR)* codes. We can see the building of MBR codes in [51] and interference alignment that can be found in [50], [57] is the basis of MSR codes. In this work, our emphasis is on the MSR codes.

Many studies (e.g., [29], [34], [55], [56]) provide recovery for multiple failures cooperatively. The idea behind them is that new nodes have to exchange the constructed data among themselves in order to reduce the overall re-pair traffic. Our work emphasizes on single-failure recovery, which is the cause for the majority of failures in cloud storage systems [31]. Also the aspect of security issues for regenerating-coded data is solved by studies (e.g., [14], [41, while the concept of security in case of FMSR codes is solved in our before work [15]. We suggest readers to refer the survey paper [17] in order to study regarding the “state of the art” research in regenerating codes.

The main advantage of our FMSR code implementation is that it removes the requirement of encoding during the operation of repair, while still maintaining the recovery performance of MSR codes. But the existing MSR codes (e.g., [50], [57]) require nodes to perform encoding operations.

Empirical studies on regenerating codes. It is only the theoretical analysis that is being provided by the existing studies on regenerating codes. Based on observation, many studies (e.g., [18], [23], [37]) find random linear codes for storage in peer-to-peer. In order to reduce the number of surviving nodes to contact during recovery which causes a higher storage cost and to evaluate the codes on a cloud storage simulator, authors of [44] propose simple regenerating codes. Authors of [33] calculate the performance of the operations of encoding/decoding of regenerating codes. In our work, we perform the implementation of a storage system and calculate the actual performance of read/write with regenerating codes, but the existing studies do not do so. Regenerating codes are implemented by NCFS [30], but does not consider MSR codes which are based on linear combinations. But here, we consider the FMSR code

implementation and perform experiments in cloud storage.

Follow-up studies on FMSR codes. We have some studies that follow after the conference version [27. Integrity checking of FMSR-coded data is supported by our FTCloud against Byzantine attacks [15]. The aspect of two-phase checking which can preserve the MDS property of the stored data after iterative repairs [28] is also theoretically proved by us. Our focus is on the deployment of regenerating codes practically. We propose a design of regenerating codes that is implementable and we also perform a knowledgeable observation in cloud storage environment practically.

CONCLUSIONS

To uphold the key concept of today's practical cloud backup storage, the reliability, we extend our FTCloud, a proxy-based multiple cloud storage system.
The main advantage of our FTCloud is that it provides fault tolerance in storage, on the other hand, it also allows in a cost-effective manner when there is permanent cloud storage. A practical version of the functional minimum storage regenerating (FMSR) codes is implemented. This regenerates new parity chunks while repairing according to the requirement of data redundancy. The encoding requirement of storage nodes or cloud while repairing is eliminated through the help of our FMSR code implementation. It ensures that the required fault tolerance is preserved by the new set of stored chunks after each round of repair. The effective capability of FMSR codes in the cloud backup usage is shown in our FTCloud prototype. Hence there is advantage from our FTCloud in terms of both monetary costs and response times.

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*, 46(4):1204–1216, Jul 2000.
[3] Amazon. AWS Case Study: Backupify. http://aws.amazon.com/solutions/case-studies/backupify/.
[4] Amazon. Case Studies. https://aws.amazon.com/solutions/casestudies/#backup.
[5] Amazon Glacier. http://aws.amazon.com/glacier/.
[6] Amazon S3. http://aws.amazon.com/s3.
[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] Asigra. Case Studies. http://www.asigra.com/product/casestudies/.
[9] AWS Service Health Dashboard. Amazon s3 availability event: July 20, 2008. http://status.aws.amazon.com/s3-20080720.html.
[10] A. Bessani, M. Correia, B. Quaresma, F. Andr´e, and P. Sousa. DEPSKY: Dependable and Secure Storage in a Cloud-of-Clouds. In *Proc. of ACM EuroSys*, 2011.
[11] K. D. Bowers, A. Juels, and A. Oprea. HAIL: A High-Availability and Integrity Layer for Cloud Storage. In *Proc. of ACM CCS*, 2009.
[12] Business Insider. Amazon's Cloud Crash Disaster Permanently Destroyed Many Customers' Data. http://www.businessinsider.com/amazon-lost-data-2011-4/, Apr 2011.
[13] B. Calder et al. Windows Azure Storage: A Highly Available Cloud Storage Service with Strong Consistency. In *Proc. of ACM SOSP*, 2011.
[14] 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.
[15] 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.
[16] 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.
[17] A. G. Dimakis, K. Ramchandran, Y. Wu, and C. Suh. A Survey on Network Codes for Distributed Storage. *Proc. of the IEEE*, 99(3):476–489, Mar 2011.
[18] A. Duminuco and E. Biersack. A Practical Study of Regenerating Codes for Peer-to-Peer Backup Systems. In *Proc. of IEEE ICDCS*, 2009.
[19] B. Escoto and K. Loafman. Duplicity. http://duplicity.nongnu.org/.
[20] 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.
[21] FUSE. http://fuse.sourceforge.net/.
[22] S. Ghemawat, H. Gobioff, and S.-T. Leung. The Google File System. In *Proc. of ACM SOSP*, 2003.
[23] C. Gkantsidis and P. Rodriguez. Network coding for large scale content distribution. In *Proc. of INFOCOM*, 2005.
[24] GmailBlog. Gmail back soon for everyone. http://gmailblog.blogspot.com/2011/02/gmail-back-soon-for-everyone.html.
[25] K. M. Greenan, E. L. Miller, and T. J. E. Schwarz. Optimizing Galois Field Arithmetic for Diverse Processor Architectures and Applications. In *Proc. of IEEE MASCOTS*, 2008.
[26] K. M. Greenan, J. S. Plank, and J. J. Wylie. Mean time to meaningless: MTTDL, Markov models, and storage system reliability. In *Proc. of USENIX HotStorage*, 2010.
[27] Y. Hu, H. C. H. Chen, P. P. C. Lee, and Y. Tang. NCCloud: Applying Network Coding for the Storage Repair in a Cloudof-Clouds. In *Proc. of FAST*, 2012.
[28] Y. Hu, P. P. C. Lee, and K. W. Shum. Analysis and Construction of Functional Regenerating Codes with Uncoded Repair for Distributed Storage Systems. In *Proc. of IEEE INFOCOM*, Apr 2013.
[29] Y. Hu, Y. Xu, X. Wang, C. Zhan, and P. Li. Cooperative recovery of distributed storage systems from multiple losses with network coding. *IEEE JSAC*, 28(2):268–276, Feb 2010.
[30] Y. Hu, C.-M. Yu, Y.-K. Li, P. P. C. Lee, and J. C. S. Lui. NCFS: On the Practicality and Extensibility of a Network-Co
[31] 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.
[32] P. Hunt, M. Konar, F. P. Junqueira, and B. Reed. ZooKeeper: Wait-Free Coordination for Internet-Scale Systems. In *Proc. of USENIX ATC*, 2010.
[33] S. Jiekak, A.-M. Kermarrec, N. L. Scouarnec, G. Straub, and A. Van Kempen. Regenerating Codes: A System Perspective. *CoRR*, abs/1204.5028, 2012.
[34] A. Kermarrec, N. Le Scouarnec, and G. Straub. Repairing Multiple Failures with Coordinated and Adaptive Regenerating Codes. In *Proc. of NetCod*, Jun 2011.
[35] O. Khan, R. Burns, J. S. Plank, W. Pierce, and C. Huang. Rethinking Erasure Codes for Cloud File Systems: Minimizing I/O for Recovery and Degraded Reads. In *Proc. of USENIX FAST*, 2012.
[36] N. Kolakowski. Microsoft's cloud azure service suffers outage. http://www.eweekeurope.co.uk/news/news-solutionsapplications/microsofts-cloud-azure-service-suffers-outage-395.
[37] M. Martal ´o, M. Picone, M. Amoretti, G. Ferrari, and R. Raheli. Randomized Network Coding in Distributed Storage Systems with Layered Overlay. In *Information Theory and ApplicationWorkshop*, 2011.
[38] M. Mayer. This site may harm yoyur computer on every search results. http://googleblog.blogspot.com/2009/01/this-site-mayharm-your-computer-on.html.
[39] MSPmentor. CloudBerry Labs Unveils Support for Low- Cost Amazon Glacier. http://mspmentor.net/managed-services/cloudberry-labs-unveils-support-low-cost-amazon-glacier/, Jan 2013.
[40] E. Naone. Are We Safeguarding Social Data? http://www.technologyreview.com/blog/editors/22924/, Feb 2009.

[41] F. Oggier and A. Datta. Byzantine Fault Tolerance of Regenerating Codes. In *Proc. of P2P*, 2011.
[42] OpenStack Object Storage. http://www.openstack.org/projects/storage/.
[43] Panzura. US Department of Justice Case Study. http://panzura.com/us-department-of-justice-case-study/.
[44] D. Papailiopoulos, J. Luo, A. Dimakis, C. Huang, and J. Li. Simple Regenerating Codes: Network Coding for Cloud Storage. In *Proc.of IEEE INFOCOM*, Mar 2012.
[45] D. A. Patterson, G. Gibson, and R. H. Katz. A case for redundant arrays of inexpensive disks (raid). In *Proc. of ACM SIGMOD*, 1988.
[46] 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.
[47] J. S. Plank, J. Luo, C. D. Schuman, L. Xu, and Z. Wilcox-O'Hearn. A Performance Evaluation and Examination of Open-Source Erasure Coding Libraries For Storage. In *Proc. of USENIX FAST*, 2009.
[48] C. Preimesberger. Many data centers unprepared for disasters:Industrygroup. http://www.eweek.com/c/a/ITManagement/Many-Data-Centers-Unprepared-for-Disasters-Industry-Group-772367/, Mar 2011.
[49] M. O. Rabin. Efficient Dispersal of Information for Security, Load Balancing, and Fault Tolerance. *Journal of the ACM*, 36(2):335–348, Apr 1989.
[50] K. Rashmi, N. Shah, and P. Kumar. Optimal Exact-Regenerating Codes for Distributed Storage at the MSR and MBR Points via a Product-Matrix Construction. *IEEE Trans. on Information Theory*, 57(8):5227–5239, Aug 2011.
[51] K. V. Rashmi, N. B. Shah, P. V. Kumar, and K. Ramchandran. Explicit Construction of Optimal Exact Regenerating Codes for Distributed Storage. In *Proc. of Allerton Conference*, 2009.
[52] 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.
[53] M. Sathiamoorthy, M. Asteris, D. Papailiopoulos, A. G. Dimakis, R. Vadali, S. Chen, and D. Borthakur. XORing Elephants: Novel Erasure Codes for Big Data. *Proc. of VLDB Endowment*, 2013.
[54] B. Schroeder and G. A. Gibson. Disk Failures in the Real World: What Does an MTTF of 1,000,000 Hours Mean to You? In *Proc. Of USENIX FAST*, Feb 2007.
[55] K. Shum. Cooperative Regenerating Codes for Distributed Storage Systems. In *Proc. of IEEE Int. Conf. on Communications (ICC)*, Jun 2011.
[56] 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.
[57] C. Suh and K. Ramchandran. Exact-Repair MDS Code Construction using Interference Alignment. *IEEE Trans. on InformationTheory*, 57(3):1425–1442, Mar 2011.
[58] TechCrunch. Online Backup Company Carbonite Loses Customers' Data, Blames And Sues Suppliers. http://techcrunch.com/2009/03/23/online-backup-companycarbonite-loses-customers-data-blames-and-sues-suppliers/, Mar 2009.
[59] TechTarget. Cloud case studies: Data storage pros offer firsthandexperiences.Http://searchcloudstorage.techtarget.com/feature/Cloud-case-studies-Data-storage-pros-offer-first-handexperiences/.
[60] M. Vrable, S. Savage, and G. Voelker. Cumulus: Filesystem backup to the cloud. In *Proc. of USENIX FAST*, 2009.
[61] M. Vukoli´c. The Byzantine Empire in the Intercloud. *ACM SIGACT News*, 41:105–111, Sep 2010.
[62] Z. Wang, A. Dimakis, and J. Bruck. Rebuilding for Array Codes in Distributed Storage Systems. In *IEEE GLOBECOM Workshops*, 2010.
[63] 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.
[64] ZDNet. AWS cloud accidentally deletes customer data. http://www.zdnet.com/aws-cloud-accidentally-eletescustomer- data-3040093665/, Aug 2011.
[65] zfec. http://pypi.python.org/pypi/zfec.