IJARCCE

ISSN (Online) 2278-1021
ISSN (Print) 2319 5940

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 4, Issue 11, November 2015

A Case Study on Markov Model for Double Fault Tolerance, Comparing Cloud based Storage System

Swaroop Tewari[1], Mohana Kumar. S[2], Dr. S N Jagadeesh[3]

Abstract: Cloud storage has gained massive popularity in the IT industry. It has proven to be cost effective and reliable. Research has shown that striping of data across multiple cloud vendors is a remedy for providing fault tolerance. In case the cloud suffers from a permanent failure it leads to loss of data in the cloud. In this scenario the lost data can be repaired or recovered using the other surviving clouds. This multiple cloud-storage system is called a Network-Coding based cloud storage system. NC Cloud is a proxy based system which provides fault tolerance and provides cost-effective repairs for systems which suffer from permanent single-cloud failure. The NC Cloud is built on top of Functional Minimum Storage Regenerating (FMSR) codes which provide the same fault tolerance as traditional erasure codes like RAID 5 and RAID 6 but use less data repair traffic. Therefore, the FMSR codes provide significant cost savings in repair over RAID 6 codes but have similar performance during upload and download of data. The concept of Network-Coding based cloud storage is gaining mass popularity due to its desirable properties like data recovery, cost effectiveness and fault tolerance.

Keywords: FMSR codes; NC Cloud; Fault tolerance; data recovery; regenerating codes; Traditional erasure codes; Mean-time-to-data-loss (MTTDL); MDS property.

I. INTRODUCTION

Cloud computing has been embraced by the IT industry which has lead to the rise of network-centric computing. The advancement of networking has made it possible to concentrate resources in large data centres where the users can pay as they consume and store data. One of the major challenges of cloud storage is dealing with huge amount of data and to securely maintain data in its data centres. It is not a feasible solution to store data using a single cloud storage provider as it raise a concern of a single point failure. To eliminate the risk of single cloud failure the data is stored in multiple clouds which improves fault tolerance of the cloud storage system [2]. Another challenge in the cloud storage is the vendor lock-in. It is not possible to store the large amount of data with different cloud service providers because it would be very expensive as the providers charge the users for outbound data. Moving these huge chunks of data can introduce significant monetary costs. Therefore it is necessary to reliably store data within single cloud service providers which makes the service providers focus on data repair and recovery.

In the existing systems used in the industry, conventional erasure codes (RAID 5) is used to stripe data to improve fault tolerance [2],[5]. This technique is suitable for short term failures or a foreseeable permanent failure. Clouds are susceptible to permanent failures. Our work focuses on unexpected permanent cloud failure. It is necessary to maintain data redundancy and fault tolerance properties on a cloud storage system to activate repair when the cloud unexpectedly fails. In the case of a permanent cloud failure, a repair operation is launched in which the data is retrieved from existing surviving clouds in the same network and lost data is reconstructed into the new cloud.

It is absolutely essential to reduce data traffic during data migration to have lower monetary costs [3].

The main goal to implement a NC cloud based storage system is to reduce data traffic and monetary costs. To achieve this, regenerating codes has been implemented for the data to be stored in a distributed storage system redundantly. The data is stored across different nodes in which each node can be referred as a storage device or a cloud storage provider. The NC cloud is built on top of the Functional Minimum Storage Regenerating (FMSR) codes. When the data repair operation begins, the encoded chunks of data are retrieved from the surviving nodes and sent to the new node. In the new node the lost data is regenerated. The advantage of regenerating codes with the same fault-tolerance level as that of the traditional erasure codes is that it requires less repair traffic resulting in lesser monetary costs.

NC cloud provides a fault tolerant storage system over multiple cloud storage providers. NC cloud is a proxy based storage system which can interconnect different clouds and can also transparently stripe data across different clouds. Functional Minimum Storage Regenerating (FMSR) codes are implemented along with NC Cloud which maintains double fault tolerance and use less traffic during repair operations. One important benefit is the elimination of performing encoding operations within storage nodes during repair.

FMSR codes are stored as encoded data chunks formed by the linear combinations of the original data. These codes are non-systematic in nature. FMSR codes have been implemented for archival applications in various organizations. These codes serve the purpose of a long term archive for data. FMSR codes design allows us to restore the whole file rather than the parts of the file during the recovery operation. This helps the lost file to be obtained completely which maintains data integrity. They can also provide an alternate solution for a system which would help store data using multiple clouds storage, along with the properties of fault tolerance and cost-effectiveness [4].

In this paper, FMSR codes are discussed and compared with the erasure codes like RAID-6 [15]. FMSR codes tend to save the repair costs by 25 percent compared to RAID-6 codes when four storage nodes are involved. It can even save repair costs up to 50 percent as the number of storage nodes further increases. On the other hand, FMSR codes have the same amount of storage overhead as RAID-6 codes and these codes can be deployed in a thin cloud setting, very suitable for today's cloud storage services. Regenerating codes are extensively studied in theoretical context [6], [7],[10],[11].

II. REPAIR OPERATIONS IN A MULTIPLE-CLOUD STORAGE

Cloud failure is a devastating event which may result in the data being lost permanently. If the data is unrecoverable then the cloud service providers would struggle to maintain their clients. In the case of cloud failures, we generally consider two types of failures: Permanent failure and Transient failure.

Permanent failure:
Permanent failure is the type of failure which is long-term. The data in the outsourced cloud will eventually be unrecoverable permanently. This unavailable data can be disastrous for the cloud service providers as well as the users. These types of failures are very unlikely to happen, but there are some instances which have lead to a permanent failure.
Malicious attacks: It is absolutely crucial to encrypt the data from the client application before the data can be stored into the cloud. This is a necessary measure to provide client confidentiality and security. It should also be noted that when the outsourced data is corrupt, it would not be useful at all [12].
Disasters in Data Centres: Data centres may suffer from unexpected disasters once in a while. There are incidents where data centres have been struck by lightning, hit by earthquake and suffered from floods. These natural disasters may lead to permanent loss of data [12].

Transient failure:
Transient failure is a short-term failure in which the cloud which is unavailable temporarily would return to service and function normally after a short period. In this type of failure no outsourced data is lost. Some transient failure may last a few minutes and some may last a few days. These failures are common and occur regularly, but will eventually be recovered [13].

III.IMPORTANCE OF FMSR CODES

In this paper, the storage system is based on a distributed multi-cloud storage where the data to be stored is striped over multiple cloud providers [1], [2], [14]. A proxy based design is responsible for the interconnection of multiple cloud repositories. The design of this storage system is shown in Fig 1. The proxy based design is important because it acts as an interface between the client and the cloud. A repair operation is activated in case the cloud experiences a permanent cloud failure.

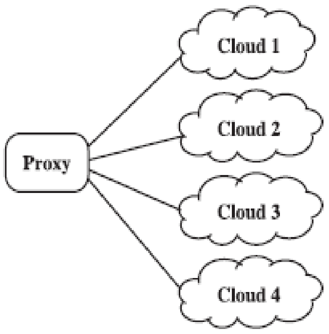


Fig1. Normal operation

The repair operation mechanism can be shown in Fig 2. In the repair operation, the proxy reads the data pieces that are essential for the reconstruction of the new data pieces from the other surviving clouds. These data pieces that are retrieved from the remaining clouds are stored in the new cloud. This repair operation does not involve any sort of direct interaction between the clouds that are active.

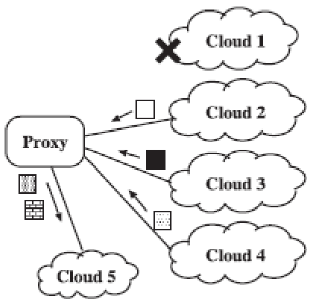


Fig2. Repair operation

Level-3 Heading: A level-3 heading must be indented, in Italic and numbered with an Arabic numeral followed by a right parenthesis. The level-3 heading must end with a colon.

To attain fault tolerance in the storage system, maximum distance separable (MDS) codes have been implemented. The file of size M is considered which is divided into equal chunks of data. These data chunks are referred to as native chunks. These native chunks are again linearly combined to obtain code chunks. These native codes are distributed over n nodes in the case where (n , k) MDS code is implemented. The total size of the codes would be M/k. With the use of FMSR codes, failures of n-k nodes can be tolerated.

In Fig 3, a double fault tolerant implementation of FMSR codes is considered. The file of size M is divided into four native chunks. They are further divided onto eight distinct code chunks $P_1,\ldots,P_8$. These distinct code chunks are formed by linear combinations of the native chunks where the size of the code chunks is M/4. In case of a node failure, any two different nodes can be used to recover the original native code chunk. In Fig 3, node 1 is assumed to have failed. The proxy collects a single code chunk from the surviving node. The operation involves a download procedure of three code chunks of the size M/4. Then the proxy performs an operation where two code chunks P_1' and P_2' are regenerated from the different linear combinations of the code chunks. The P_1'and P_2' regenerated are stored in a new node by the proxy. In FMSR the code size is considered as 2M, and the repair traffic is 0.75M. The key feature of FMSR codes is that it does not perform encoding during repair [1].

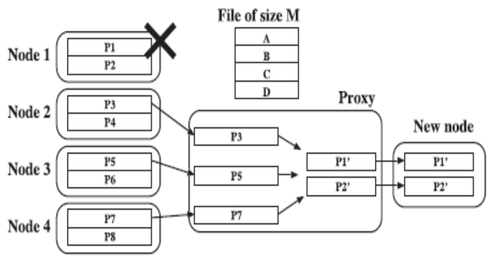


Fig3. FMSR codes

The file of size M is divided into 2(n-2) native chunks to generalize double-fault-tolerant FMSR codes for n storage nodes. These native chunks are used to generate 2n code chunks. Each node stores two code chunks of size M/2(n-2) each and the total storage size is Mn/(n-2). To perform repair operation for a failed node a chunk is downloaded from each of the other n-1 nodes, which results in the repair traffic of M(n-1)/2(n-2). Considering n is large, FMSR codes can save repair traffic by close to 50 percent [1], [16].

IV. IMPLEMENTATION OF FMSR CODES

After describing the importance and the advantages of FMSR codes in single cloud storage, the details of implementing them in multiple cloud storage is specified in this section. The nodes in the cloud repository are viewed as a logical storage node. On the basis of the property of FMSR codes, the lost chunks do not need to be exactly reconstructed. Instead, in each repair operation code chunks are regenerated that are not completely identical to the originally stored on the node that failed as long as the MDS property is satisfied. A two-phase checking scheme is proposed here which ensures that code chunks on all nodes always satisfy MDS property. There are three operations performed on a file using FMSR codes: 1. File upload, 2. File download, 3. Repair.

File upload: For a file to be uploaded, it is first divided into k(n-k) equal size native chunks which are denoted by $(F_i)_{i=1,2,\ldots,k(n-k)}$. these k(n–k) native chunks are then encoded into n(n-k) code chunks which are denoted by $(P_i)_{i=1,2,\ldots,n(n-k)}$. Each P_i is formed by a linear combination of the k(n-k) native chunks. Specifically, we let EM=$[\alpha_{i,j}]$ be an [n(n-k) X k(n-k)] encoding matrix for some coefficients $\alpha_{i,j}$ (where i=1, . . . ,n(n-k) and j=1, . . . , k(n-k)) in the Galois field GF(2^8). The row vector of EM is called an encoding coefficient vector (ECV) which contains k(n-k) elements. The ECV is used to denote the ith row vector of EM. Every P_i is calculated by the product of ECV_i and all the native chunks F_1, F_2,…,$F_{k(n-k)}$, where all the arithmetic operations are performed over GF(2^8). The code chunks of the file are then evenly stored in the n storage nodes, each having (n-k) chunks. The whole EM is stored as a metadata object that is then replicated to all the storage nodes. EM can be constructed in a number of ways as long as it passes two-phase checking which is mentioned below in iterative repairs. The implementation details of arithmetic in Galois Field are briefly discussed in [8].

File download: In the download operation, all the corresponding metadata objects that contain the ECVs are downloaded initially. Any k of the n storage nodes is selected and k(n-k) code chunks from k nodes are downloaded. The ECVs of the k(n-k) code chunks can from a [k(n - k) X k(n – k)] square matrix. According to the MDS property the inverse of the square matrix must exist. The original k(n-k) native chunks are obtained when the inverse square matrix is multiplied with the code chunks. In the download operation, FMSR codes are treated as standard Reed-Solomon codes and the technique of creating an inverse matrix to decode the original data has been described in [9].

Iterative Repairs: A File f is considered for repair of FMSR codes in a permanent single-node failure in the cloud storage system. A major challenge is to make sure that the MDS property holds after iterative repairs given that FMSR codes regenerate different chunks in each repair. A two-phase checking heuristics is proposed as follows:
It is assumed that the (r-1)th repair is successful and rth repair (where r>1) is considered for the operation where a single node failure has occurred. First, the new set of chunks in all the storage nodes are checked if it satisfies the MDS property after the rth repair. All the other new set of chunks in the storage nodes are checked if they satisfy the MDS property after the (r+1) th repair, should another single permanent node failure occur. This property is referred to as the repair MDS (rMDS) property [18]. The rth repair is now briefly described in the following steps:

Step 1: Downloading the encoding matrix from an existing node (Surviving node).
The encoding matrix EM specifies the ECVs for constructing all the code chunks by using the linear combinations of native chunks. The EVCs are used later for two-phase checking. The EM is embedded in a metadata object which is replicated, the metadata object can be simply be downloaded from one of the surviving nodes.
Step 2: Selecting one ECV from each of the n-1 surviving nodes.
Each ECV in EM corresponds to a code chunk where one ECV is picked from the n-1 surviving nodes. These ECVs are referred as ECV_{i1}, ECV_{i2},…, $ECV_{i(n-1)}$
Step 3: Generating a repair matrix.
An (n-k) X (n-1) repair matrix RM= $[Y_{I,j}]$ is constructed, where each element

$Y_{I,j}$ (where i=1,…,n-k and j=1,…,n-1) is randomly selected in GF(2^8).

Step 4: Computing the ECVs for the new code chunks where a new encoding matrix is reproduced.

The RM is multiplied with the ECVs selected in Step 2 to construct n-k new ECVs, denoted by

$$ECV'_i = \Sigma^{n-1}_{j=1} Y_{i,j}\, ECV_i \text{ , for } i=1,2,\ldots,n-k.$$

A new encoding matrix EM’ is reproduced which is formed by the substitution of ECVs of EM of the failed node with the corresponding new ECVs.

Step 5: Checking whether the newly reproduced EM’ is satisfying both the MDS and the rMDS properties.

The MDS property is verified intuitively by enumerating all $(^n{}_k)$ subsets of k nodes to check if each of their corresponding encoding matrices forms a full rank. The rMDS property, any possible node failure is checked where one out of n-k chunks can be collected from each of the other n-1 surviving nodes and reconstruct the chunk in the new node to maintain the MDS property. The number of checks performed for the rMDS property is at most $n(n-k)^{n-1}\,(^n{}_k)$. if n is small, then the enumeration complexities for both MDS and rMDS properties are manageable. If either of the two phases fails, then we return back to the step 2 and repeat. Step 1 to Step 5 only deals with ECVs, so their overhead does not depend on the chunk size.

Step 6: Downloading the actual chunk data and regenerating the new chunk data.

If the two-phase checking performed in Step5 is successful, the n-1 chunks are downloaded that correspond to the selected ECVs in Step 2 from n-1 surviving storage nodes to NC Cloud. By using new ECVs computed in the step 4 new chunks are regenerated and uploaded from the NC Cloud to a new node.

V. ANALYSIS

A It is absolutely essential for checking rMDS property in each repair to maintain MDS property after iterative repairs. Counter-example is used to demonstrate that without rMDS property check, the MDS property would be lost in the next repair. A simulation is also used to demonstrate that the two-phase checking can sustain much iteration of repairs in more general cases.

A Counter-example

A counter-example is shown in Fig 4, which is used to illustrate the necessity of rMDS property. The notations used are the same as Fig 3 with n=4 and k=2. It is assumed that the code chunks $P_1,\ldots,P_8$ are linearly combined from native chunks A,B,C and D. It is not difficult to verify that the code chunks $P_1,\ldots,P_8$ satisfy the MDS property, which can be further stated that the four chunks from any two nodes can be used to reconstruct the native chunks A,B,C and D. there is no operation performed to check whether the rMDS property is also being satisfied or not.

It is now considered that node 4 has failed. The repair operation selects one chunk from each of the Nodes 1, 2 and 3 according to FMSR codes. These chunks are used to regenerate the new code chunks P'_7 which is represented in (eq1) and P'_8 is represented in (eq2) which are stored in the new node.

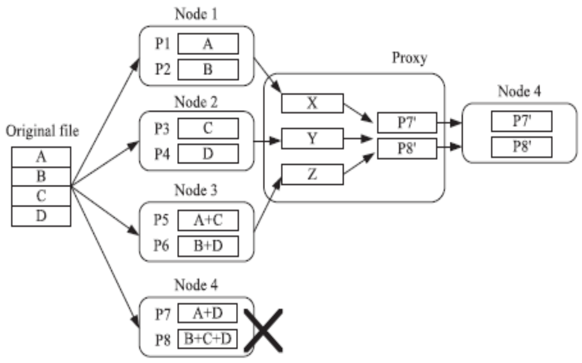


Fig4. Counter-example, code chunks that satisfy the MDS property but not the rMDS property

There are 2^3=8 possible selections of{X , Y , Z}. One possible selection is considered here where the new code chunks become

$$P'_7 = Y_{1,1} P_1 + Y_{1,2} P_3 + Y_{1,3} P_5,\ldots\text{(eq1)}$$
$$P'_8 = Y_{2,2} P_1 + Y_{2,2} P_3 + Y_{2,3} P_5.\ldots\text{(eq2)}$$

Where $Y_{i,j}$ (i=1,…,n-k and j=1,…,n-1) are some random coefficients used for generating new code chunks. Then we have

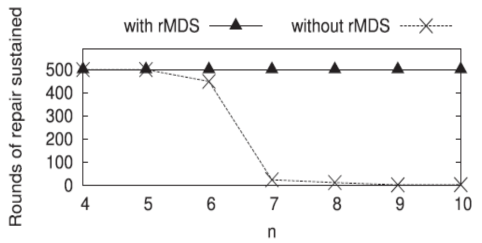
$$P'_7 = (Y_{1,1} + Y_{1,3})A + (Y_{1,2} + Y_{1,3})C,$$
$$P'_8 = (Y_{2,1} + Y_{2,3})A + (Y_{2,2} + Y_{2,3})C,$$

According to figure 4, P_1=A and P_2=C due to which it is not possible to reconstruct a native chunk D from P_1, P_2, P'_7 ,P'_8.The MDS property would not be satisfied as nodes 1 and 4 cannot be used to reconstruct the native chunks which would result in a failure of the repair operation.

Simulation

Simulations are performed to evaluate the overhead of the two-phase checking and to justify that checking the rMDS property can make sustainable iterative repairs. Initially, multiple rounds of node repairs are considered for different values of n. In each round of the simulation, a random node is picked to fail and then repair operation is performed on the failed node. A repair is considered to be bad if the Step 2 to Step 5 of the Two-phase checking is repeated over a threshold number of times, but no suitable encoding matrix has been obtained. A number of rounds of repair are carried out and the repair is stopped when a bad repair is encountered [1].

In Figure 5, an example of the simulation is described where there is a comparison between repairs where rMDS property is checked and repairs where only MDS property is checked.



.

Fig5. Comparison between repairs with MDS and rMDS.

From this example it is clear that checking rMDS properties enables more round of repairs to be sustained before a bad repair is encountered. Suppose the threshold is set to be 20 loops. Then, we can sustain 500 rounds of repair for different values of n by checking rMDS property. When we check only MDS property we quickly encounter a bad repair in three rounds of repair for n=10.

Reliability analysis

Reliability is a very desirable property for a storage system. We compare the reliability of FMSR codes and the traditional RAID-6 codes with respect to different failure rates using the mean-time-to-data-loss (MTTDL) metric [17]. Mean-time-to-data-loss (MTTDL) is defined as the expected time elapsed until the original data become unrecoverable. MTTDL is widely adopted reliability metric which is only used for comparative study of different coding schemes with different repair performances.

The Markov model for double-fault-tolerant codes where k= (n-2) is used to solve the MTTDL which is shown in Fig 6 . In the Markov model the state i (where i=0,1,2,3) denotes the number of failed nodes in the storage system. State 3 signifies that there are more than two failed nodes where data is permanently lost. MTTDL is computed as the expected time to move from state 0 (where all nodes are normal) to state 3.

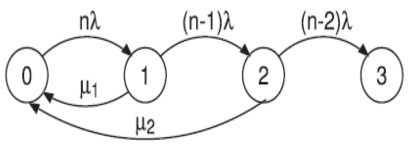


Fig 6. Markov model for double fault tolerant codes.

The node failures and repairs are assumed to be independent events that follow an exponential distribution [19], [21]. Let λ be the node failure rate where 1/λ is the expected time to failure of a node. The transition rate from state i to state i+1 is (n-i) λ, where i=0, 1, 2. The repair rates for single-node and double-node failure is μ_1 and μ_2 respectively. S is considered to be the size of the data stored in each node where amount of original data stored is (n-2) S and B is considered the network capacity between the surviving nodes and the proxy.

Initially, repair for a single-node failure is considered. The repair traffic for FMSR codes is (n-1)S/2, therefore μ_1=2B/(n-1)S. The repair traffic for RAID-6 codes is (n-2)S, hence μ_1=B/(n-1)S. Finally, in case of double node failure the FMSR codes and the RAID-6 codes reconstruct lost data by downloading the amount of original data ((n-2)S) from remaining k=n-2 surviving nodes where μ_2=B/(n-2)S. MTTDL is evaluated for specific parameters where n=10, k=8 and S=1. The MTTDL for different values of λ from 0.1 to 1 (in units per year) is shown in FIG7.a when B=1Gbps while FIG 7.b shows the MTTDL for different values of B from 0.1 to 1 (in units of Gbps) when λ-0.5 per year. Under these circumstances, the MTTDL of FMSR codes is 50 to 80 percent longer than traditional RAID-6 codes. [1], [15]

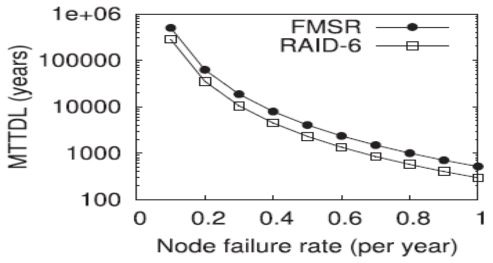


Fig 7.a. MTTDL vs node failure rate

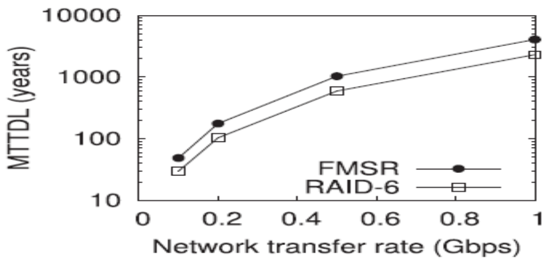


Fig 7.b. MTTDL vs node transfer rate

Response Time Analysis

The response time is calculated for file upload, file download and repair. These operations are performed on the local cloud storage. The experiment is performed to test the response time for File Upload, File download and Repair where the value of n=4 and k=2 with varying file sizes. There are eight files randomly selected from 1MB to 500MB as the data set. The response times of all the three operations are plotted versus the file size in Fig 8 [20].

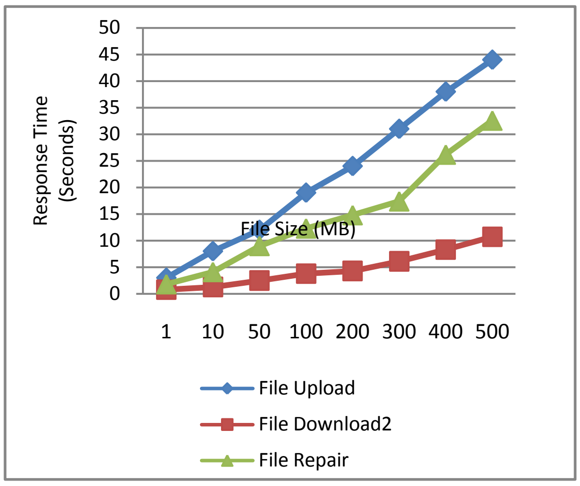


Fig8. Response time analysis for File downloads

The path of the chosen repository is set to a non-existent location in order to stimulate a node failure in repair. The Experiment is performed for each operation where the different file sizes are considered and the time taken for response is noted. After performing these experiments and comparing the response time with RAID-6, it is abundantly clear that the .RAID-6 codes have less response time than FMSR codes during File upload and File download regardless of n and k. On the other hand, FMSR codes have slightly less response time than RAID-6 during repair operations [1]. This is mainly due to the fact that FMSR codes download lesser data in file repair which

is explained briefly below.
For example, a file of size 500 MB is considered where n=4 and k=2. While uploading the file of size 500 MB RAID-6 codes takes 1.53 seconds to encode while FMSR codes takes 5.48 seconds. When downloading a 500 MB size file, FMSR codes takes 2.71 seconds to decode while RAID-6 does not take any time to decode as the native chunks are available. These differences increase with n and k. The main advantage of FMSR codes is observed during repair operations. The FMSR codes have slightly less response time compared to RAID-6. This is attained because FMSR codes download less data during repair.For repairing a 500-MB file with n=4 and k=2, FMSR codes spend 4.02 seconds in download, while the native chunk repair of RAID-6 codes spends 5.04 s.
FMSR codes generally have slightly longer response time compared to RAID-6 which may raise the argument that RAID-6 is a more feasible solution for achieving repair in Cloud storages. In the case of a local cloud, this difference in response time might favor RAID-6, but in a commercial cloud the FMSR codes have a clear advantage.[1],[2],[6]. The encoding/decoding overhead which occurs in FMSR codes can be easily masked with network fluctuations in the internet. This advantage of FMSR codes in a commercial cloud is achieved by lesser repair traffic during the repair operations. FMSR codes implementation eliminates the encoding requirement of nodes, while maintaining the recovery performance of Minimum Storage Regenerating codes. Since, the NC-Cloud storage system's main aim is to provide a fault tolerant system which performs repairs during permanent node failures, FMSR codes are implemented to achieve a reliable and efficient storage system.

VI. CONCLUSION AND FUTURE WORK

The NC Cloud is a proxy-based, multiple-cloud storage system that practically addresses the reliability of today's cloud backup storage. NC Cloud not only provides fault tolerance in storage, but also allows cost-effective repair when a cloud permanently fails. NC Cloud implements a practical version of the FMSR codes, which regenerates new parity chunks during repair subject to the required degree of data redundancy. 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. Our NC Cloud prototype shows the effectiveness of FMSR codes in the cloud backup usage, in terms of monetary costs and response times. This provides a very good scope for future because of the system's efficiency, reliability and its ability to recover from permanent single cloud failures.

REFERENCES

1. NCCloud: A Network-Coding-Based Storage System in a Cloud-of-Clouds Henry C.H. Chen, Yuchong Hu, Patrick P.C. Lee, and Yang Tang
2. A H. Abu-Libdeh, L. Princehouse, and H. Weatherspoon, "RACS: A Case for Cloud Storage Diversity," Proc. ACM First ACM Symp
3. 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," Comm. the ACM, vol. 53, no. 4, pp. 50-58, 2010.
4. R. Ahlswede, N. Cai, S.-Y.R. Li, and R.W. Yeung, "Network Information Flow," IEEE Trans. Information Theory, vol. 46, no. 4, pp. 1204-1216, July 2000.
5. A. Bessani, M. Correia, B. Quaresma, F. Andre´, and P. Sousa, "DEPSKY: Dependable and Secure Storage in a Cloud-of-Clouds," Proc. ACM European Conf. Computer Systems (EuroSys '11), 2011.
6. B. Chen, R. Curtmola, G. Ateniese, and R. Burns, "Remote Data Checking for Network Coding-Based Distributed Storage Systems," Proc. ACM Workshop Cloud Computing Security Workshop (CCSW '10), 2010.
7. A.G. Dimakis, P.B. Godfrey, Y. Wu, M. Wainwright, and K. Ramchandran, "Network Coding for Distributed Storage Systems," IEEE Trans. Information Theory, vol. 56, no. 9, pp. 4539-4551, Sept. 2010.
8. K.M. Greenan, E.L. Miller, and T.J.E. Schwarz, "Optimizing Galois Field Arithmetic for Diverse Processor Architectures and Applications," Proc. IEEE Int'l Symp. Modeling, Analysis and Simulation of Computers and Telelcomm. Systems (MASCOTS '08), 2008.
9. J.S. Plank, "A Tutorial on Reed-Solomon Coding for Fault-Tolerance in RAID-Like Systems," Software—Practice & Experience vol. 27, no. 9, pp. 995-1012, Sept. 1997.
10. 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. Information Theory, vol. 57, no. 8, pp. 5227-5239, Aug. 2011.
11. K.V. Rashmi, N.B. Shah, P.V. Kumar, and K. Ramchandran, "Explicit Construction of Optimal Exact Regenerating Codes for Distributed Storage," Proc. Allerton Conf., 2009.
12. C. Preimesberger, "Many Data Centers Unprepared for Disasters: Industry Group," http://www.eweek.com/c/a/ITManagement/ Many-Data-Centers-Unprepared-for-Disasters- Industry-Group-772367/, Mar. 2011.
13. H. Blodget, "Amazon's Cloud Crash Disaster Permanently Destroyed Many Customers' Data," http://www.businessinsider. com/amazon-lost-data-2011-4/, Apr. 2011.
14. 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-Coding-Based Distributed File System," Proc. Int'l Symp. Network Coding (NetCod '11), 2011.
15. I. Reed and G. Solomon, "Polynomial Codes over Certain Finite Fields," J. the Soc. Industrial and Applied Math., vol. 8, no. 2, pp. 300- 304, 1960.
16. 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. Storage, vol. 7, no. 3, article 11, 2011
17. K.M. Greenan, J.S. Plank, and J.J. Wylie, "Mean Time to Meaningless: MTTDL Markov Models and Storage System Reliability," Proc. USENIX Second Workshop Hot Topics in Storage and File Systems (HotStorage '10), 2010.
18. Y. Hu, P.P.C. Lee, and K.W. Shum, "Analysis and Construction of Functional Regenerating Codes with Uncoded Repair for Distributed Storage Systems," Proc. IEEE INFOCOM, Apr. 2013.
19. B. Schroeder and G.A. Gibson, "Disk Failures in the Real World: What Does an MTTF of 1,000,000 Hours Mean to You?" Proc. Fifth USENIX Conf. File and Storage Technologies (FAST '07), Feb. 2007. OpenStack Cloud Software, "OpenStack Object Storage," http:// www.openstack.org/projects/storage/, 2013.