A Proxy-based Multiple Cloud Storage System

Kavya P M[1] Vydehi M R[2] S K Pushpa[3] Gireesh Babu C N[4]
[1,2]8thSemester, Department of Information Science & Engineering, BMSIT&M, Bangalore, India
[3]Associate Professor, Department of Information Science & Engineering, BMSIT&M, Bangalore, India
[4]Assistant Professor, Department of Information Science & Engineering, BMSIT&M, Bangalore, India

Abstract:- **Cloud computing is a phrase used to describe a variety of computing concepts that involve a large number of computers connected through a real-time Communication network such as the Internet.To provide fault tolerance for cloud storage, recent studies show to stripe data across multiple cloud vendors. However, if a cloud suffers from a permanent failure and loses all its data, we need to repair the lost data with the help of the other surviving clouds to preserve data redundancy. Proxy-based storage system is proposed for fault-tolerant multiple-cloud storage called NCCloud, which achieves cost-effective repair for a permanent single-cloud failure. NCCloud is built on top of a network-coding-based storage scheme called the functional minimum-storage regenerating (FMSR) codes, which maintain the same fault tolerance and data redundancy and use less repair traffic and hence, incur less monetary cost due to data transfer. One key design feature of FMSR codes is that the encoding requirement of storage nodes is relaxed during repair, while preserving the benefits of network coding in repair. FMSR codes provide significant monetary cost savings in repair over RAID-6 codes, while having comparable response time performance in normal cloud storage operations such as upload/download.**

Keywords- Regenerating codes; network coding; fault tolerance; recovery; FMSR;

1. INTRODUCTION

Cloud computing is a phrase used to describe a variety of computing concepts that involve a large number of computers connected through a real-time Communication network such as the Internet. Cloud computing predominantly deals with network based services and these services appear to be provided by real server hardware, but they are provided by virtual hardware in actuality. The virtual hardware is simulated by software running on one or more real physical machines. Cloud computing can be leveraged to offer applications, platforms and infrastructure as services through the internet loud storage provides an on-demand remote backup solution. However, using a single-cloud storage provider increases chances of having a single point of failure and vendor lock-ins. A feasible solution is to stripe data across different cloud providers. By exploiting the diversity of multiple clouds [8], the fault tolerance of cloud storage can be improved. While striping data with conventional erasure, codes performs well when some clouds experience short-term transient failures or permanent failures [2], there are real-life cases showing that permanent failures do occur and are not always foreseeable. When a cloud fails permanently, it is necessary to activate repair to maintain data redundancy and fault tolerance. A repair operation retrieves data from existing surviving clouds over the network and reconstructs the lost data in a new cloud. Today's cloud storage providers charge users for outbound data, so moving an enormous amount of data across clouds can introduce significant monetary costs [12]. It is important to reduce the repair traffic and hence the monetary cost due to data migration. To minimize repair traffic, regenerating codes have been proposed for storing data redundantly in a distributed storage system. Each node could refer to a simple storage device, or a cloud storage provider. Regenerating codes are built on the concept of network coding [3], in the sense that nodes perform encoding operations and send encoded data. During repair, each surviving node encodes its stored data chunks and sends the encoded chunks to a new node, which then regenerates the lost data. It is shown that regenerating codes require less repair traffic than traditional erasure codes with the same fault-tolerance level. Regenerating codes have been extensively studied in the theoretical context. However, the practical performance of regenerating codes remains uncertain. One key challenge for deploying regenerating codes in practice is that most existing regenerating codes require storage nodes to be equipped with computation capabilities for performing encoding operations during repair. On the other hand, to make regenerating codes portable to any cloud storage service, it is desirable to assume only a thin-cloud interface, where storage nodes only need to support the standard read/write functionalities [11]. This motivates to explore, from an applied perspective, how to practically deploy regenerating codes in multiple-cloud storage, if only the thin-cloud interface is assumed.

In this paper, the design and implementation of NCCloud is discussed, a proxy-based storage system designed for providing fault-tolerant storage over multiple cloud storage providers. NCCloud can interconnect different clouds and transparently stripe data across the clouds. On top of NCCloud, the first implementable design for the functional minimum-storage regenerating (FMSR) codes is proposed. FMSR code implementation maintains double-fault tolerance and has the same storage cost as in traditional erasure coding schemes based on RAID-6 codes, but uses less repair traffic when recovering a single-cloud failure. In particular, the need to perform encoding operations is eliminated within storage nodes during repair, while preserving the benefits of network coding in reducing repair traffic. One tradeoff of FMSR codes is that they are non-systematic that store only encoded chunks formed by the linear combination of the original data chunks and do not keep the original data chunks as in systematic coding schemes. FMSR codes is mainly designed for long-term archival applications, in which 1) data backups are rarely read in practice, and 2) it is common to restore the whole file

rather than parts of the file should a lost file needs to be recovered.

There are many real life examples in which enterprises and organizations store an enormous amount of archival data using cloud storage [5],[9]. In August 2012, Amazon further introducedGlacier [6], a cloud storage offering optimized for low-costdata archiving and backup (with slow and costly data retrieval) that is being adopted by cloud backup solutions [4]. FMSR codes provide an alternative option for enterprises and organizations to store data using multiple-cloud storage in a fault tolerant and cost-effective manner.While this work is motivated by and established withmultiple-cloud storage in mind, FMSRcodes can also find applications in general distributed storage systems is pointed out where storage nodes are prone to failuresand network transmission bandwidth is limited. In thiscase, minimizing repair traffic is important for reducing theoverall repair time. Inmultiple-cloud storage, FMSR codes can save therepair cost by 25 percent compared to RAID-6 codeswhen four storage nodes are used, and up to 50 percent as the number of storage nodes furtherincreases. In the meantime, FMSR codes maintain the same amount of storage overhead as RAID-6codes. Note that FMSR codes can be deployed in athin-cloud setting as they do not require storagenodes to perform encoding during repair, while stillpreserving the benefits of network coding in reducingrepair traffic. Thus, FMSR codes can be readilydeployed in today's cloud storage services. The implementation details of how a file object can be stored via FMSR codes are described. In particular, a two-phase checking scheme is proposed whichensures that double-fault tolerance is maintained inthe current and next round of repair. By performingtwo-phase checking, double-faulttolerance is maintained after iterative rounds ofrepair of node failures.

2. IMPORTANCE OF REPAIR IN MULTIPLE-CLOUDSTORAGE

In this section, the importance of repair in cloudstorage is discussed. Twotypes of failures normally encountered: transient failure and permanent failure.

Transient failure: A transient failure is expected to beshort-term, such that the "failed" cloud will return tonormal after some time and no outsourced data are lost.Table 1 show several real-life examples for the occurrences of transient failures in today's clouds, where the durationsof such failures range from several minutes to several days.

Cloud Service	Failure reason	Duration	Date
Google Gmail	Software bug	4 days	Feb27-Mar2 2011
Google search	Programming error	40 min	Jan 31 2009
Amazon S3	Gossip protocol blowup [10]	6-8 hours	July 20 2008
Microsoft Azure	Malfunction	22 hours	Mar 13,14 2008

TABLE 1:Examples of Transient Failures in Different Cloud Services

We highlight that even though Amazon claimsthat its service is designed for providing 99.99 percent availability [7], there are arising concerns about this claimand the reliability of other cloud providers after Amazon'soutage in April 2011. If multiple-cloud storage is deployed with enough redundancy,then data can be retrieved from the other survivingclouds during the failure period.

Permanent failure: A permanent failure is long-term, inthe sense that the outsourced data on a failed cloud willbecome permanently unavailable. Clearly, a permanentfailure is more disastrous than a transient one. Although weexpect that a permanent failure is unlikely to happen, thereare several situations where permanent cloud failures arestill possible:

*Data center outages in disaster:*For example, 50 percent of the respondents have noplans to repair damages after a disaster. It wasreported that the earthquake and tsunami innortheastern Japan in March 11, 2011 knocked out several data centers there.

*Data loss and corruption:*There are real-life caseswhere a cloud may accidentally lose data. Sometimes it is also possible to corrupt the data present in the cloud. Hackers can easily get access to the cloud.

Malicious attacks: To provide security guarantees foroutsourced data, one solution is to have the clientapplication encrypt the data before putting the dataon the cloud. On the other hand, if the outsourced data are corrupted (e.g., by virus or malware), theneven though the content of the data is encrypted andremains confidential to outsiders, the data itself areno longer useful.

Unlike transient failures, where the cloud is assumed tobe able to return to normal, permanent failures will makethe hosted data in the failed cloud no longer accessible, sowe must repair and reconstruct the lost data in a differentcloud or storage site to maintain the required degree of faulttolerance. In our definition of repair, we mean to retrievedata only from the other surviving clouds, and reconstructthe data in a new cloud or another storage site. So it is essential to overcome transient and permanent failure.

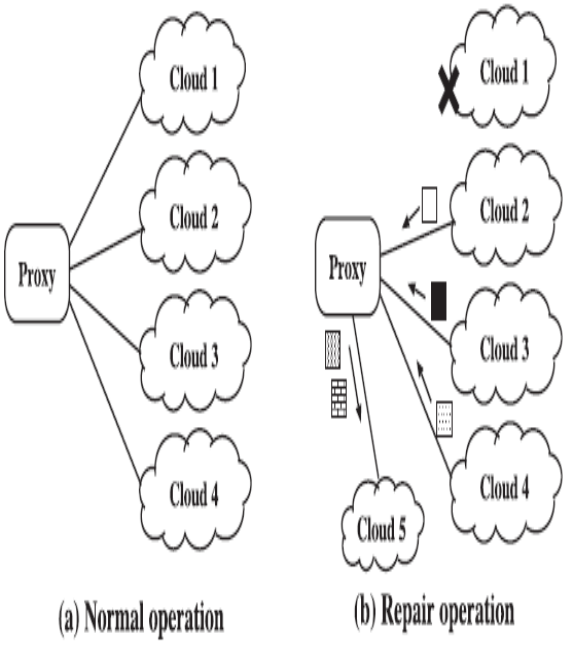

Fig.1. Proxy-based design for multiple-cloud storage: (a) normal operation and (b) repair operation when cloud node 1 fails. During repair, the proxy regenerates data for the new cloud.[1]

3. MOTIVATION OF FMSR CODES

Consider a distributed, multiple-cloud storage settingfrom a client's perspective, where data are striped overmultiple cloud providers. A proxy-based design interconnects multiple cloud repositories, asshown in Fig. 1a. The proxy serves as an interface betweenclient applications and the clouds. If a cloud experiences apermanent failure, the proxy activates the repair operation,as shown in Fig. 1b. That is, the proxy reads the essentialdata pieces from other surviving clouds, reconstructs newdata pieces, and writes these new pieces to a new cloud.Note that this repair operation does not involve directinteractions among the clouds.

Consider fault-tolerant storage based on a type ofmaximum distance separable (MDS) codes. Given a file objectof size M, we divide it into equal-size native chunks,which are linearly combined to form code chunks. The native/code chunks are thendistributed over n (larger than k) nodes, each storing chunksof a total size M=k, such that the original file object may bereconstructed from the chunks contained in any k of then nodes. Thus, it tolerates the failures of any n - k nodes.This fault-tolerance feature is termed as the MDS property. Theextra feature of FMSR codes is that reconstructing thechunks stored in a failed node can be achieved bydownloading less data from the surviving nodes thanreconstructing the whole file.

This paper considers a multiple-cloud setting with twolevels of reliability: *fault tolerance and recovery*. First assume that the multiple-cloud storage is double-faulttolerant (e.g., as in conventional RAID-6 codes) andprovides data availability under the transient unavailabilityof at most two clouds. That is set k = n -2. Thus, clients can always access their data as long as no more thantwo clouds experience transient failures (see examples inTable 1) or any possible connectivity problems.Second, single-fault recovery is considered in multiple-cloud storage,given that a permanent cloud failure is less frequent butpossible. The primary objective is to minimize the cost ofstorage repairfor a permanent single-cloud failure. In this work, focusis on comparing two codes: traditional RAID-6 codes and FMSR codes with double-fault tolerance. Repair traffic is defined as the amount of outbounddata being downloaded from the other surviving cloudsduring the single-cloud failure recovery. The repair traffic needs to be minimized for cost-effective repair. The repair traffic involved in differentcoding schemes can be studied via examples. Suppose that we store a fileof size M on four clouds, each viewed as a logical storagenode. First consider conventional RAID-6 codes,which are double-fault tolerant. Here, we consider a RAID-6 code implementation based on the Reed-Solomon code, as shown in Fig. 2a. We divide the file into two native chunks (i.e., A and B) of size M=2 each. We add two code chunks formed by the linear combinations of the native chunks. Suppose now that Node 1 is down. Then, the proxy must download the same number of chunks as the original file from two other nodes (e.g., B and A þ B from Nodes 2 and 3, respectively). It then reconstructs and stores the lost chunk A on the new node. The total storage size is 2M, while the repair traffic is M. Regenerating codes have been proposed to reduce the repair traffic. One class of regenerating codes is called the exact minimum-storage regenerating (EMSR) codes. EMSR codes keep the same storage size as in RAID-6 codes, while having the storage nodes send encoded chunks to the proxy so as to reduce the repair traffic. Fig. 2b illustrates the double-fault-tolerant implementation of EMSR codes. We divide a file into four chunks, and allocate the native and code chunks as shown in the figure. Suppose Node 1 is down. To repair it, each surviving node sends the XOR summation of the data chunks to the proxy, which then reconstructs the lost chunks. We can see that in EMSR codes, the storage size is 2M (same as RAID-6 codes), while the repair traffic is 0:75M, which is 25 percent of saving. EMSR codes leverage the notion of network coding [3], as the nodes generate encoded chunks during repair.

Consider the double-fault-tolerant implementation of FMSR codes as shown in Fig. 2c. We divide the file into four native chunks, and construct eight distinct code chunks P1; ...; P8 formed by different linear combinations of the native chunks. Each code chunk has the same size M/4 as a native chunk. Any two nodes can be used to recover the original four native chunks. Suppose Node 1 is down.

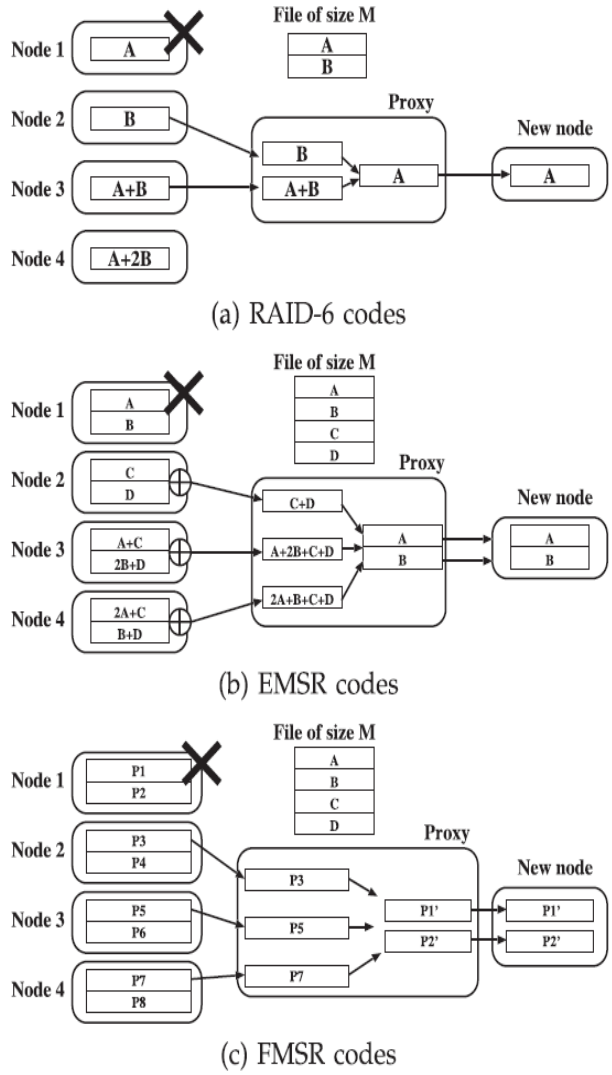

Fig.2. Examples of repair operations in different codes with n = 4 and k=2

The proxy collects one code chunk from each surviving node, so it downloads three code chunks of size M=4 each. Then, the proxy regenerates two code chunks $P_1^!$And $P_2^!$ formed by different linear combinations of the three code chunks. Note that $P_1^!$ and $P_2^!$ are still linear combinations of the native chunks. The proxy then writes $P_1^!$ and $P_2^!$ to the new node. In FMSR codes, the storage size is 2M (as in RAID-6 codes), yet the repair traffic is 0:75M, which is the same as in EMSR codes. A key property of our FMSR codes is that nodes do not perform encoding during repair. To generalize double-fault-tolerant FMSR codes for n storage nodes, we divide a file of size M into 2(n-2) native chunks, and use them to generate 2n code chunks. Then, each node will store two code chunks of size $\frac{M}{2(n-2)}$ each. Thus, the total storage size is$\frac{Mn}{n-2}$. To repair a failed node, we download one chunk from each of the other n-1 nodes.In contrast, for RAID-6 codes, the total storage size is also $\frac{Mn}{n-2}$, while the repair traffic is M. When n is large, FMSR codes can save the repair traffic by close to 50 percent. Note that FMSR codes are non-systematic, as they keep only code chunks but not native chunks. To access a single chunk of a file, we need to download and decode the entire file for that particular chunk. This is opposed to systematic codes (as in traditional RAID storage), in which native chunks are kept. Nevertheless, FMSR codes are acceptable for long-term archival applications, where the read frequency is typically low. Also, to restore backups, it is natural to retrieve the entire file rather than a particular chunk.

4. FMSR CODE IMPLEMENTATION

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

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

5. NCCLOUD DESIGN AND IMPLEMENTATION

NCCloud is implemented as a proxy that bridges user applications and multiple clouds. Its design is built on three layers. The *file system layer* presents NCCloud as a mounted drive, which can, thus, be easily interfaced with general user applications. The *coding layer* deals with the encoding and decoding functions. The *storage layer* deals with read/ writes requests with different clouds. Each file is associated with a metadata object, which is replicated at each repository. The metadata object holds the file details and the coding information.

NCCloud is mainly implemented in Python, while the coding schemes are implemented in C for better efficiency.
The file system layer is built on FUSE. The coding layer implements both RAID-6 and FMSR codes. RAID-6 code implementation is based on the Reed- Solomon code for baseline evaluation. Recall that FMSR codes generate multiple chunks to be stored on the same repository. To save the request cost overhead, multiple chunks destined for the same repository are aggregated before upload. Thus, FMSR codes keep only one chunk per file object on each cloud, as in RAID-6 codes. To retrieve a specific chunk, calculate it's offset within the combined chunk and issue a range GET request and make NCCloud deployable in one or multiple machines. In the latter case, use ZooKeeper to implement a distributed file-based shared lock to avoid simultaneous updates on the same file and conduct preliminary evaluations in a LAN environment and observe that the overhead due to ZooKeeper is minimal. Here the focus is on deploying NCCloud on a single machine and mount NCCloud as a local file system.

The Design part of NCCloud can be easily understood with the help of Architecture diagram and Dataflow diagram.

The Architecture diagram of NCCloud is as shown below

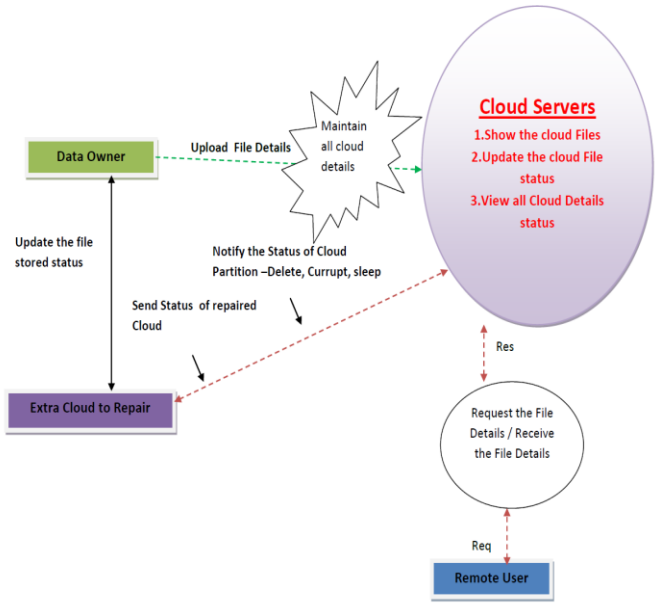

Fig.3. Architecture diagram

The Dataflow diagram of NCCloud is as shown below

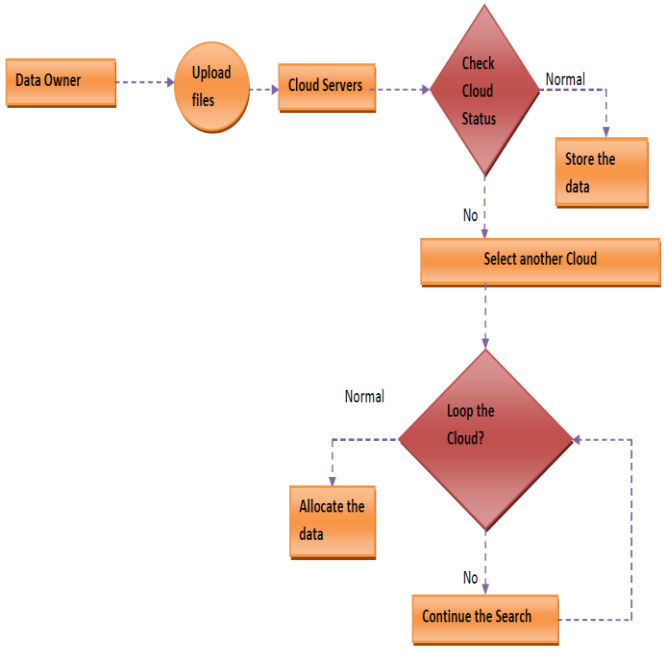

Fig.4. Dataflow diagram

The implementation modules of NCCloud is divided into 6 parts

- ***Data Owner:***In this module, the data owner uploads their data in the cloud server. For the security purpose the data owner splits file into four packets, encrypts the data file and then store in the multiple clouds.
- ***Proxy Server:***The Proxy server is a proxy-based design that interconnects multiple cloud repositories, as shown in this system. The proxy serves as an interface between client applications and the clouds. If a cloud experiences a permanent failure, the proxy activates the repair operation.
- ***Cloud Server:***The cloud service provider manages a cloud to provide data storage service. Data owner encrypts and splits the data files and store them in the multiple clouds
- ***NC Cloud*:** NCCloud as a proxy that bridges user applications and multiple clouds.
- ***Data Consumer (End User)*:** In this module, the user can only access the data file with the encrypted key to access the file. Then Proxy based NC cloud combines all the packets and sends to Remote user
- ***Threat Model (Attacker):***Attacker can attempt to transient failure for a cloud by making Doze Off for a particular period of time. The Attacker can also attempts to permanent failure by Deleting and corrupting the cloud.

6. EVALUATION AND TESTCASES

NCCloud prototype is used to evaluate RAID-6 codes and FMSR codes in multiple-cloud storage. The goal of the experiments is to explore the practicality of using FMSR codes in multiple-cloud storage. Evaluation consists of two parts. First compare the monetary costs of using RAID-6 and FMSR codes based on the price plans of today's cloud providers and evaluate the response time performance of NCCloud prototype atop a local cloud and also a commercial cloud provider.

*Summary of evaluation result:*The emphasis is on the monetary cost advantage of FMSR codes over RAID-6 codes, while still maintaining acceptable response time performance. In terms of monetary costs, the normal operations, both RAID-6 and FMSR codes incur similar storage costs, while in the repair operation, FMSR codes save a significant amount of transfer costs over RAID-6 codes. In terms of response time, both FMSR and RAID-6 codes have comparable response time performance (within 5 percent) when deployed on a commercial cloud (Azure). The resulting response time is mainly determined by the transmission performance of the Internet.

The following table shows the different test cases of NC Cloud.

Test Id	Test Name	Input	Output	Expected Result	Status
1	Owner browsing the data	file	Data stored	Data stored in encrypted form	PASS
	Owner browsing data	data	Data is not Present in directory	Display Data	FAIL
2	Owner adding data to server	Server IP address	Data stored in server	Data stored in server	Pass
	Owner adding data to server	Invalid server IP address	Data cannot stored in server	Data stored in server	Fail
3	End user browsing the data	file	Data stored	Data stored in encrypted form	PASS
	End user browsing data	data	Data is not Present in directory	Display Data	FAIL
4	End user downloading data from server	Server IP address	Downloaded Data from the server	Data should be downloaded	PASS
	End user downloading data from server	Invalid server IP address	Cant Downloaded Data from the server	Data should be downloaded	FAIL
5	Type1attack Doze off state	File name & corresponding ip address for the file he wants to hack	File is successfully hacked under doze off state	File should not work till doze off time gets expired	PASS
	Type1 attack Doze off state	Invalid File name and corresponding ip address for the file he wants to hack	File is not hacked under doze off state	File should not work till doze off time gets expired	FAIL
6	Type2 Attack Hacker Deletes the file	File name and corresponding ip address for the file he wants to delete	File is deleted	Selected File should be deleted	PASS
	Type2 Attack Hacker Deletes the file	Invalid File name & corresponding ip address for the file he wants to delete	File is not deleted	Selected File should be deleted	FAIL
7	Type3 Attack Hacker adding malicious data into the file	File name and corresponding ip address and selects the file and adds malicious data	Malicious data stored in the file	File MAC address should be changed	PASS
	Type3 Attack Hacker adding malicious data into the file	Invalid File name and corresponding ip address and adds malicious data	Malicious data not stored in the file	File MAC address should be changed	FAIL

TABLE 2: NCCloud Test cases

7. ADVANTAGES OF NCCLOUD

NCCloud typically provides many advantages in cloud storage which are as follows

- Simple and Cost effective
- Fault tolerance among the clouds
- Data back and recovery
- Regenerating codes
- Repair operations among the cloud
- Iterative Repairs

8. CONCLUSION

NCCloud, a proxy-based, multiple-cloud storage system is discussed that practically addresses the reliability of today's cloud backup storage. NCCloud not only provides fault tolerance in storage, but also allows cost-effective repair when a cloud permanently fails. NCCloud 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. NCCloud prototype shows the effectiveness of FMSR codes in the cloud backup usage, in terms of monetary costs and response times.

REFERENCES

[1] Henry C.H. Chen, Yuchong Hu, Patrick P.C. Lee, and Yang Tang, "NCCloud: A Network-Coding-Based Storage System in a Cloud-of-Clouds", IEEE Transactions on computers, VOL. 63, NO. 1, January 2014

[2] H. Abu-Libdeh, L. Princehouse, and H. Weatherspoon, "RACS: A Case for Cloud Storage Diversity," Proc. ACM First ACM Symp.Cloud Computing (SoCC '10), 2010.

[3] 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.

[4] Amazon Web Services, "AWS Case Study: Backupify," http:// aws.amazon.com/solutions/case-studies/backupify/, 2013.

[5] Amazon Web Services, "Case Studies," https://aws.amazon.com/solutions/case-studies/#backup, 2013.

[6] Amazon Web Services, "Amazon Glacier," http://aws.amazon.com/glacier/, 2013.

[7] Amazon Web Services, "Amazon S3," http://aws.amazon.com/ s3, 2013.

[8] 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.

[9] Asigra, "Case Studies," http://www.asigra.com/product/casestudies/,2013.

[10] Amazon Web Services, "Amazon S3 Availability Event: July 20, 2008," http://status.aws.amazon.com/s3-20080720.html, July 2008.

[11] 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.

[12] K.D. Bowers, A. Juels, and A. Oprea, "HAIL: A High-Availability and Integrity Layer for Cloud Storage," Proc. 16th ACM Conf.Computer and Comm. Security (CCS '09), 2009.