NC-Cloud on Fault Tolerant Multiple clouds Storage using Data Redundancy Networking Coding

A.Kalpana
M.Tech (CSE), kalpana2914@gmail.com
Department of Computer Science & Engineering. Nagole Institute of Technology & Science.

Internal guide details

M. NareshChoudary
Assistant Professor, Department of Computer Science & Engineering.Nagole Institute of Technology & Science.
E-mail id: naresh.makkena@hotmail.com

HOD details

Dr. P. Venkateswarlu
Professor & HOD, Department of Computer Science & Engineering, Nagole Institute of Technology & Science.
E-mail id: venkat123.pedakolmi@gmail.com

Abstract

This Paper, present a proxy predicated storage system for fault tolerance multiple-cloud storage called NC (network coding), which overcome quandaries such as sempiternal failure and loss of data, lost data is rehabilitated with the avail of data redundancy. NC achieves cost-efficacious rehabilitation for a sempiternal single-cloud failure, it is built on top of networking coding predicated storage scheme called the storage regenerating code(SR) unlike traditional RAID-6 utilized for fault tolerance and data redundancy SR use less repair traffic and hence incur less monetary cost, and more preponderant replication time performance in mundane cloud operation such as, upload/download. Implementation of a proof-of-concept prototype of NC and deploy it atop both local and commercial cloud. Proof-of-concept is designed to determine feasibility , but does not represent deliverables is withal kenned as proof of principle. It is utilized to check system requisites, such as how system can be integrated or throughput can be achieved through a given configuration. Key feature of SR code is that we relinquish the encoding requisite of storage nodes during repair, to make regenerating code portable to any cloud storage it is desirable to postulate only a thin-cloud interface, where storage node only need to fortify the standard read/indite functionalities.

Keywords: Fault Tolerents; Multiple clouds Storage; Data Redundancy; Networking Coding

Introduction

Cloud computing denotes a family of increasingly popular on-line accommodations for archiving, backup, and even primary storage of files, and transforming business by offering incipient options for businesses to increment efficiencies while reducing costs [2]. It lets utilizer can access all applications and documents from anywhere in the world, liberating from the confines of the desktop and making it more facile for group members in different locations to collaborate. It is a model for enabling convenient, on-demand network access to a shared pool of configurable and reliable computing resources (e.g., networks, servers, storage, applications, accommodations) that can be rapidly provisioned and relinquished with minimal consumer management effort or accommodation provider interaction. Cloud computing provides computation, software, data access, and storage resources without requiring cloud users to ken the location and other details of the computing infrastructure.

Cloud storage provides data on-demand and solution. A plausible solution is to stripe data across different cloud providers, by exploiting the diversity of multiple clouds, the fault tolerance of cloud storage [3]. While striping data with conventional erasure, codes performs well when some clouds experience short-term transient failures or prognosticable perpetual failures, there is authentic-life case exhibiting that sempiternal failures do occur and are not always prognosticable. a distributed cryptographic system that sanctions a set of servers to prove to a client that a stored file is intact and retrievable. Storage providers charge users for outbound data, so moving a cyclopean amount of data across clouds can introduce paramount monetary costs. It is paramount to reduce the rehabilitation traffic .To minimize repair traffic, storage regenerating codes have been proposed for storing data redundantly in a distributed storage system. A proxy-predicated storage system is designed for providing fault-tolerant storage over multiple cloud storage providers. NC can interconnect different clouds and transparently stripe data across the clouds. On top of NC, we propose the first implementable design for the storage regenerating (SR) code [1].

1. Related Work

2.1 Existing System:

When a cloud fails sempiternally, it is obligatory to activate repair to maintain data redundancy and fault tolerance. A rehabilitation operation retrieves data from subsisting surviving clouds over the network and reconstructs the lost data in an incipient cloud. Today's cloud storage providers charge users for outbound data (optically discern the pricing models in Section 6.1), so moving a cyclopean amount of data across clouds can introduce paramount monetary costs. One key challenge for deploying regenerating codes in practice is that most subsisting regenerating codes require storage nodes to be equipped with computation capabilities for performing encoding operations during repair.

2.2 Proposed System:

To provide fault tolerance for cloud storage, recent studies propose to stripe data across multiple cloud vendors. However, if a cloud suffers from a perpetual failure and loses all its data, we require to rehabilitate the lost data with the avail of the other surviving clouds to preserve data redundancy. To minimize repair traffic, regenerating codes [16] have been proposed for storing data redundantly in a distributed storage system (an amassment of interconnected storage nodes). Each node could refer to a simple storage contrivance, a storage site, or a cloud storage provider. In particular, we propose a two-phase checking scheme, which ascertains that double-fault tolerance is maintained in the current and next round of rehabilitation. By performing two-phase checking, we ascertain that double-fault tolerance is maintained after iterative rounds of rehabilitation of node failures.

2.3 Proposed System Model:

We currently show however F-MSR preserves the rehabilitation traffic via associate example. Suppose that we incline to store a file of size M on four clouds, every viewed as a logical storage node. Sanction us to initial contemplate RAID-6 that is double-fault tolerant. Here, we incline to cogitate the RAID-6 implementation fortified Reed-Solomon codes [26], as shown in Figure 2(a). We incline to divide the file into 2 native chunks (i.e., A and B) of size M/2 every. We incline to integrate 2 code chunks fashioned by the linear coalescences of the native chunks. Suppose currently that Node one is down. Then the proxy should transfer an equipollent range of chunks because the pristine file from 2 different nodes (e.g., B and A + B from Nodes two and three, respectively). It then reconstructs and stores the lost chunk A on the incipient node. The entire storage size is 2M, whereas the rehabilitation traffic is M. we incline to currently contemplate the double-fault tolerant implementation of F-MSR in an exceedingly proxy-predicated setting, F-MSR divides the file into four native chunks, and constructs eight distinct code chunks P1, • • •, P8 fashioned by plenarily different linear coalescences of the native chunks. Every code chunk has an equipollent size M/4 as a native chunk. Any 2 nodes may be habituated recuperate the first four native chunks. Suppose Node one is down. The proxy accumulates one code chunk from every living node, thus it downloads 3 code chunks of size M/4 every. Then the proxy regenerates 2 code chunks P'1 and P'2 fashioned by thoroughly different linear coalescences of the 3 code chunks. Note that P'1 and P'2 square measure still linear coalescences of the native chunks. The proxy then indites P'1 and P'2 to the incipient node. In F-MSR, the storage size is 2M (as in RAID-6), however the rehabilitation traffic is zero.75M that is twenty fifth of preserving.Note that F-MSR keeps solely code chunks in lieu of native chunks. To access one chunk of a file, we'd relish to transfer and re-indite the whole file for the genuine chunk. All equipollent, F-MSR is opportune for long deposit applications, whose scan frequency is often low [6]. Additionally, to revive backups, it\'s natural to retrieve the whole file in lieu of a culled chunk. This paper considers the baseline RAID-6 implementation victimization Reed-Solomon codes. Its repair methodology is to reconstruct the total file, and is applicable for all erasure codes mundanely. Recent studies show that cognizance reads are often decremented concretely for XOR predicated erasure codes. For instance, in RAID-6, cognizance reads Associate in Nursing be reduced by twenty fifth compared to reconstructing the total file [28, 29]. Though such approaches area unit suboptimal (recall that F-MSR will lay aside to five hundredth of rehabilitation traffic in RAID-6), their utilization of economical XOR operations are often of sensible interest.

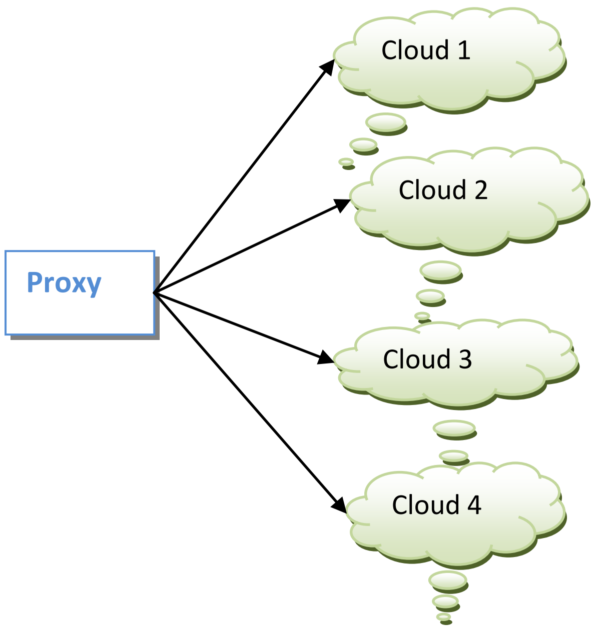


Fig 1: Normal Operation

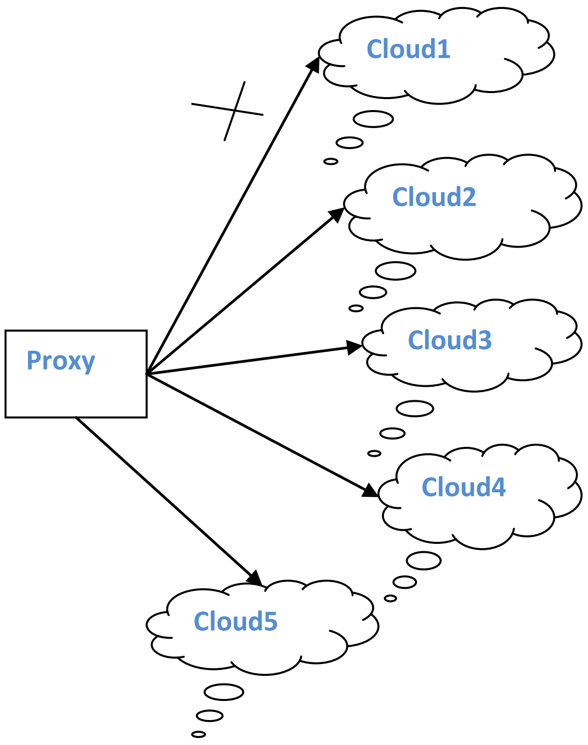


Fig 2: Repair Operation When Cloud 1 fail.

2.4 FMSR code description:

FMSR codes preserve the benefits of network coding as they minimize the rehabilitation bandwidth (e.g., the rehabilitation and width preserving compared to RAID-6 codes is up to

50% [22][21]). FMSR codes use uncoded repair without requiring encoding of surviving nodes during repair,and this can minimize disk reads as the amount of data read from disk is identically tantamount to that being transferred. FMSR codes are designed as non-systematic codes as they do not keep the pristine uncoded data as their systematic counterparts, but instead store only linear cumulations of pristine data called parity chunks. Each round of rehabilitation regenerates incipient parity chunks for the incipient node and ascertains that the fault tolerance level is maintained. A trade-off of FMSR codes is that the whole encoded file must be decoded first if components of a file are accessed. Nevertheless, FMSR codes are suited to long-term archival applications, since data backups are infrequently read and it is mundane to instaurate the whole file rather than file components.

Because of above advantages of FMSR code we consider a distributed, multiple-cloud storage setting from a client's perspective, where data is striped over multiple cloud providers. We propose a proxy-predicated design [1], [30] that interconnects multiple cloud repositories, as shown in Fig 1. The proxy accommodates as an interface between client applications and the clouds. If a cloud experiences a perpetual failure, the proxy activates the rehabilitation operation, as shown in Fig 2. From the above diagram, proxy reads the essential data pieces from other surviving clouds, reconstructs incipient data pieces, and indites these incipient pieces to an incipient cloud. Note that this rehabilitation operation does not involve direct interactions among the clouds.

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

2. Implementation

FMSR codes:

We propose a proxy predicated design that interconnects multiple cloud repositories. The proxy accommodates as an interface between client applications and the clouds. The extra feature of FMSR codes is that reconstructing the chunks stored in a failed node can be achieved by downloading less data from the surviving nodes than reconstructing the whole file.

File Upload:

To upload a file F , we first divide it into k(n −k) equalsize native chunks, denoted by (Fi)i=1,2,•••,k(n−k).We then encode these k(n − k) native chunks into n(n − k) code chunks, denoted by (Pi)i=1,2,•••,n(n−k). Each Pi is composed by a linear amalgamation of the k(n − k) native chunks.

File Download:

To download a file, we first download the corresponding metadata object that contains the ECVs. Then we cull any k of the n storage nodes, and download the k(n−k) code chunks from the k nodes.

Repair:

We now consider the double-fault tolerant implementation of FMSR codes.We divide the file into four native chunks, and construct eight distinct code chunks P1, • • • , P8 composed by different linear amalgamations of the native chunks. Each codechunk has the same size M/4 as a native chunk. Any two nodes can be habituated to recuperate the pristine four native chunks. Suppose Node1 is down. The proxy accumulates one code chunk from each surviving node, so it downloads three code chunks of size M/4 each. Then theproxy regenerates two code chunks P1' and P2' composed by different linear coalescences of the three code chunks. Note that P1 ' and P2 ' are still linear amalgamations of the native chunks. The proxy then indites P1 ' and P2 ' to the incipient node. In FMSR codes, the storage size is 2M (as in RAID-6 codes), yet the rehabilitation traffic is 0.75M, which is identically tantamount to in EMSR codes. A key property of our FMSR codes is that nodes do not perform encoding duringrepair.

3. Experimental Results

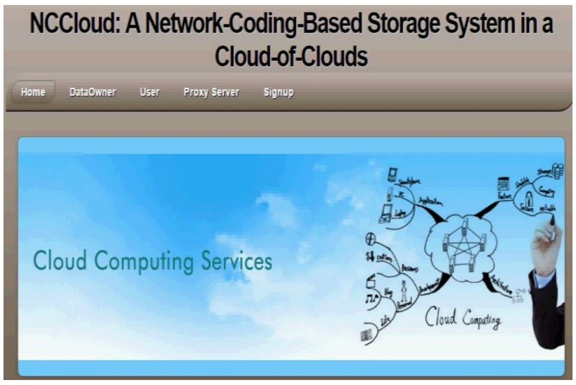
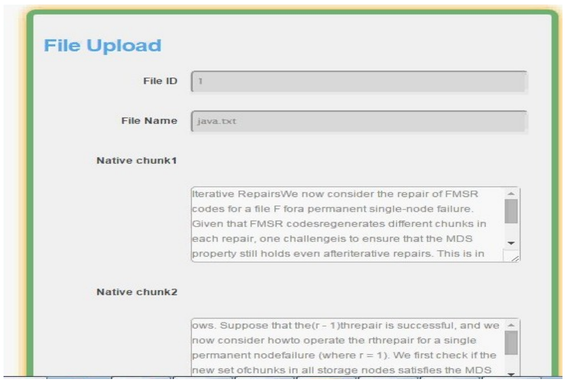


Fig 3: Home Page.


File Upload

File ID 1

File Name java.txt

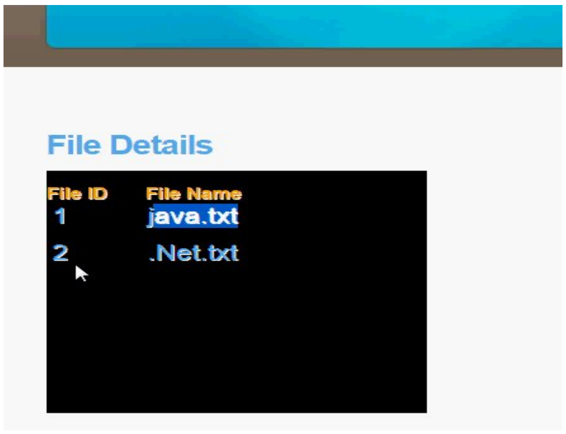
Native chunk1

Iterative RepairsWe now consider the repair of FMSR codes for a file F fora permanent single-node failure. Given that FMSR codesregenerates different chunks in each repair, one challengeis to ensure that the MDS property still holds even afteriterative repairs. This is in

Native chunk2

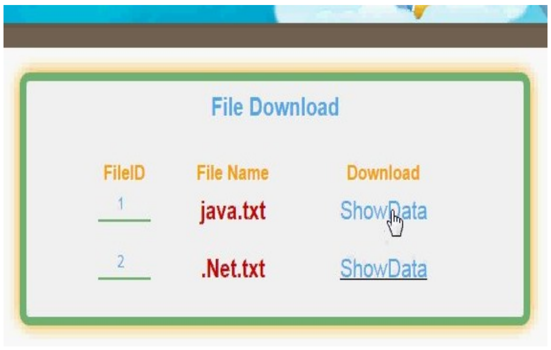
ows. Suppose that the(r - 1)threpair is successful, and we now consider howto operate the rthrepair for a single permanent nodefailure (where r = 1). We first check if the new set ofchunks in all storage nodes satisfies the MDS

Fig 4: File Upload.


File Details

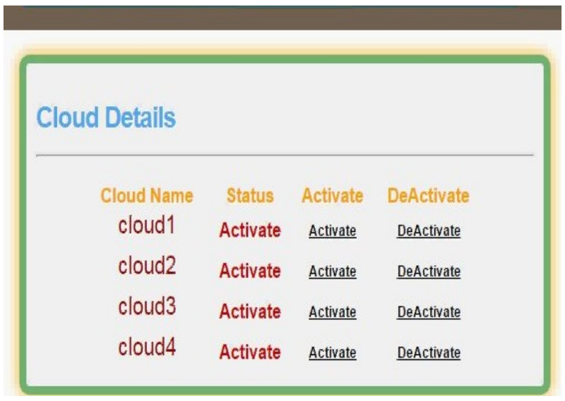
File ID	File Name
1	java.txt
2	.Net.txt

Fig 5: User Uploaded File Details.


File Download

FileID	File Name	Download
1	java.txt	ShowData
2	.Net.txt	ShowData

Fig 6: File Download Details.


Cloud Details

Cloud Name	Status	Activate	DeActivate
cloud1	Activate	Activate	DeActivate
cloud2	Activate	Activate	DeActivate
cloud3	Activate	Activate	DeActivate
cloud4	Activate	Activate	DeActivate

Fig 7: Cloud Details Page.

4. Conclusion

The quandary verbalization of the project is to provide fault tolerance for cloud storage and to study propose to stripe data across multiple cloud vendors. However, if a cloud suffers from a sempiternal failure and loses all its data, we require to rehabilitate the lost data with the avail of the other surviving clouds to preserve data redundancy. Hence we make us of NC (network code), a proxy-predicated server, multiple-cloud storage system that technically addresses the reliability of cloud backup storage. NC not only provides fault tolerance in storage, but withal sanctions cost-efficacious repair when a cloud sempiternally fails. It implements a practical version of SR codes, which regenerates incipient parity chunks during repair. Ascertaining that the incipient set of stored chunks after each round of rehabilitation preserves the required fault tolerance.

5. References

[1] H. Abu-Libdeh, L. Princehouse, and H. Weatherspoon. RACS: ACase for Cloud Storage Diversity. In *Proc. of ACM SoCC*, 2010.

[2] R. Ahlswede, N. Cai, S.-Y. R. Li, and R. W. Yeung. NetworkInformation 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-Availabilityand Integrity Layer for Cloud Storage. In *Proc. of ACM CCS*, 2009.

[12] Business Insider. Amazon's Cloud Crash Disaster PermanentlyDestroyed Many Customers' Data. http://www.businessinsider.com/amazon-lost-data-2011-4/, Apr 2011.

[13] B. Calder et al. Windows Azure Storage: A Highly AvailableCloud Storage Service with Strong Consistency. In *Proc. of ACMSOSP*, 2011.

[14] B. Chen, R. Curtmola, G. Ateniese, and R. Burns. RemoteData Checking for Network Coding-Based Distributed StorageSystems. In *Proc. of ACM CCSW*, 2010.

[15] H. C. H. Chen and P. P. C. Lee. Enabling Data Integrity Protectionin Regenerating-Coding-Based Cloud Storage. In *Proc. of IEEESRDS*, 2012.

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

[17] A. G. Dimakis, K. Ramchandran, Y. Wu, and C. Suh. A Surveyon 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 RegeneratingCodes 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 GloballyDistributed Storage Systems. In *Proc. of USENIX OSDI*, 2010.