INTERNATIONAL JOURNAL OF ADVANCED RESEARCH IN COMPUTER SCIENCE AND ENGINEERING TECHNOLOGIES

ISSN : 2454-9924

RANDOMIZED DISPERSED STORAGE SYSTEM FOR MULTIPLE CLOUD WITH NETWORK CODING

G. Hemavathi,
M.E*, Dept. of CSE,
Dr. Pauls Engineering College,
ghemavathicse@gmail.com

Mrs. D. Udaya,
Assistant Professor,
Dept. of CSE,
Dr. Pauls Engineering College,

Abstract

In cloud storage data are striped across multiple cloud vendors to provide fault tolerance. When the cloud permanently fails due to any disaster, can be repaired using other surviving clouds to preserve data redundancy. In existing system they implemented a conventional erasure codes performs when some cloud experience short term transient failure not the permanent failure. In proposed, implements a proxy-based storage system for fault-tolerant multiple-cloud storage called NC Cloud, which achieves cost-effective repair for a permanent single-cloud failure. NC Cloud 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 as in traditional erasure codes (e.g., RAID-6), but use less repair traffic and hence incur less monetary cost due to data transfer. This system implement a proof-of-concept prototype of NC Cloud and deploy it at top of both local and commercial clouds. By using FMSR, provides key feature to relax encoding requirement when the repair operation.

Keywords: Regenerating codes, network coding, fault tolerance, recovery,

1. INTRODUCTION

Cloud storage provides an on-demand remote backup solution. However, using a single cloud storage provider raises concerns such as having a single point of failure and vendor lock-ins. While striping data with conventional erasure

ISSN : 2454-9924

codes performs well when some clouds experience short-term transient failures or foreseeable permanent failures , there are real-life cases showing that permanent failures do occur and are not always foreseeable . In view of this, this work focuses on unexpected permanent cloud failures. When a cloud fails permanently, it is necessary to activate repair to maintain data redundancy and fault tolerance. A repair operation retrieves data from existing surviving clouds over the network and reconstructs the lost data in a new cloud. Today's cloud storage providers charge users for outbound data, so moving an enormous amount of data across clouds can introduce significant monetary costs. It is important to reduce the repair traffic (i.e., the amount of data being transferred over the network during repair), 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 (a collection of interconnected storage nodes). Each node could refer to a simple storage device, a storage site, or a cloud storage provider. Regenerating codes are built on the concept of network coding, 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 inter- face, where storage nodes only need to support the standard read/write functionalities.

2. IMPORTANCE OF REPAIR IN MULTIPLE- CLOUD STORAGE

We consider two types of failures: transient failure and permanent failure.

Transient failure: A transient failure is expected to be short-term, such that the "failed" cloud will return to normal after some time and no outsourced data is lost. We highlight that even though Amazon claims that its service is designed for providing 99.99% availability , there are arising concerns about this claim and the reliability of other cloud providers after Amazon's outage in April 2011 . We thus expect

ISSN : 2454-9924

that transient failures are common, but they will eventually be recovered. If we deploy multiple-cloud storage with enough redundancy, then we can retrieve data from the other surviving clouds during the failure period.

Permanent failure: A permanent failure is long-term, in the sense that the outsourced data on a failed cloud will become permanently unavailable. Clearly, a permanent failure is more disastrous than a transient one. Although we expect that a permanent failure is unlikely to happen, there are several situations where permanent cloud failures are still possible: • Data center outages in disasters. AFCOM [48] found that many data centers are ill-prepared for disasters. For example, 50% of the respondents have no plans to repair damages after a disaster. It was reported [48] that the earthquake and tsunami in northeastern Japan in March 11, 2011 knocked out several data centers there. • Data loss and corruption. There are real-life cases where a cloud may accidentally lose data [12], [40], [58]. In the case of Magnolia [40], half a terabyte of data, including its backups, are all lost and unrecoverable. • Malicious attacks. To provide security guarantees for outsourced data, one solution is to have the client application encrypt the data before putting the data on the cloud. On the other hand, if the outsourced data is corrupted (e.g., by virus or malware), then even though the content of the data is encrypted and remains confidential to outsiders, the data itself is no longer useful. AFCOM found that about 65 percent of data centers have no plan or procedure to deal with cyber-criminals.

3. MOTIVATION OF FMSR CODES

We consider a distributed, multiple-cloud storage set- ting from a client's perspective, where data is striped over multiple cloud providers. We propose a proxy- based design that interconnects multiple cloud repositories. The proxy serves as an interface between client applications and the clouds. If a cloud experiences a permanent failure.

We consider fault-tolerant storage based on a type of maximum distance separable (MDS) codes. Given a file object of size M, we divide it into equal-size native chunks, which are linearly combined to form code chunks. When an (n,k)-MDS code is used, the native/code chunks are then distributed over n (larger than k) nodes, each storing chunks of a total size M/k, such that the original file object may be reconstructed from the chunks contained in any k of the n nodes. Thus, it tolerates the failures of any n − k nodes. We call this fault tolerance feature the MDS property. The extra feature of FMSR codes is that reconstructing the chunks stored in a failed node can be achieved

ISSN : 2454-9924

by downloading less data from the surviving nodes than reconstructing the whole file. This paper considers a multiple-cloud setting with two levels of reliability: fault tolerance and recovery. First, we assume 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 (see examples in Table 1) or any possible connectivity problems. We expect that such a fault tolerance level suffices in practice. Second, we consider single-fault recovery in multiple-cloud storage, given that a permanent cloud failure is less frequent but possible. Our primary objective is to minimize the cost of storage repair for a permanent single-cloud failure. In this work, we focus on comparing two codes: traditional RAID-6 codes and our FMSR codes with double-fault tolerance3. We define the repair traffic as the amount of outbound data being downloaded from the other surviving clouds during the single-cloud failure recovery. We seek to minimize the repair traffic for cost-effective repair. Here, we do not consider the inbound traffic (i.e., the data being written to a cloud), as it is free of charge for many cloud providers (see Table 3 in Section 6). We now study the repair traffic involved in different coding schemes via examples. Suppose that we store a file of size M on four clouds, each viewed as a logical storage node. Let us 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. 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 + 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. Figure 2(b) 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,

ISSN : 2454-9924

which is 25% of saving (compared with RAID-6 codes). EMSR codes leverage the notion of network coding, as the nodes generate encoded chunks during repair.

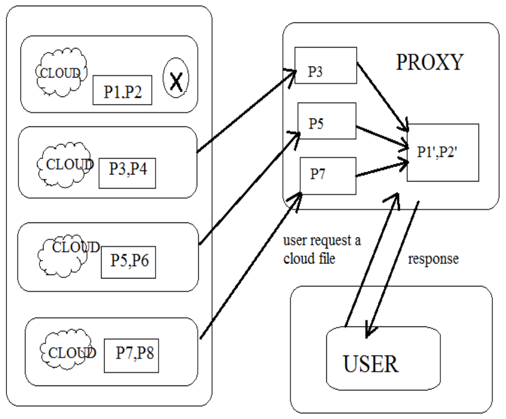

Figure: System model for Repair operation

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; (3) repair. Each cloud repository is viewed as a logical storage node. Our implementation assumes a thin-cloud interface [60], 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. In this section, we analyze the importance of the two-phase checking scheme.

4.1 Basic operations

4.1.1 File Upload

To upload a file F, we first divide it into k(n−k) equal- size native chunks, denoted by (Fi)i=1,2,···,k(n−k). We then encode these k(n − k) native chunks into n(n − k) code chunks, denoted by (Pi)i=1,2,···,n(n−k). Each Pi is formed by a linear combination of the k(n − k) native chunks. Specifically, 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(28). We call a row vector of EM an encoding coefficient vector (ECV), which contains k(n−k) elements. We let ECVi denote the ith row vector of EM. We com- pute each Pi by the product of ECVi and all the native chunks F1,F2,···, Fk(n−k), i.e., Pi = Pk(n−k) j=1 αi,jFj for i = 1,2,···,n(n−k), where all arithmetic operations are performed over GF(28). The code chunks are then evenly stored in the n storage nodes, each having (n−k) chunks. Also, we store the whole EM in a metadata object that is then replicated to all storage nodes There are many ways of constructing EM, as long as it passes our two-phase checking. Note that the implementation details of the arithmetic operations in Galois Fields are extensively discussed in

4.1.2 File Download

To download a file, we first download the corresponding metadata object that contains the ECVs. Then we select any k of the n storage nodes, and download the k(n−k) code chunks from the k nodes. The ECVs of the k(n−k) code chunks can form a k(n−k)×k(n−k) square matrix. If the MDS property is maintained, then by definition, the inverse of the square matrix must exist. Thus, we multiply the inverse of the square matrix with the code chunks and obtain the original k(n − k) native chunks. The idea is that we treat FMSR codes as standard Reed-Solomon codes, and our technique of creating an inverse matrix to decode the original data has been described in the tutorial [46].

4.1.3 Iterative Repairs

We now consider 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 challenge is to ensure that the MDS property still holds even after iterative repairs. This is in contrast to regenerating the exact lost chunks as in RAID-6, which guarantees the invariance of the stored chunks. Here, we propose a two-phase checking heuristic as follows. Suppose that the (r −1)th repair is successful, and we now consider how to operate the rth repair for a single permanent node failure (where r ≥ 1). We first check if the new set of chunks in all storage nodes satisfies the MDS property after the rth repair. In addition, we also check if another new set of chunks in all storage nodes still satisfies the MDS property after the (r + 1)th repair, should another single permanent node failure

ISSN : 2454-9924

occur (we call this the repair MDS (rMDS) property).

5. NCCLOUD DESIGN AND IMPLEMENTATION

We implement NC Cloud as a proxy that bridges user applications and multiple clouds. Its design is built on three layers. The file system layer presents NC Cloud 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/write 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 (e.g., encoding coefficients for FMSR codes). 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. Our RAID-6 code implementation is based on the Reed-Solomon code (as shown in Figure 2(a)) for baseline evaluation. We use zfec to implement the RAID-6 codes, and we utilize the optimizations made in zfec to implement FMSR codes for fair comparison. 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 (aggregated) chunk per file object on each cloud, as in RAID-6 codes. To retrieve a specific chunk, we calculate its offset within the combined chunk and issue a range GET request.

6. EVALUATION

6.1 Cost Analysis

6.1.1 Repair Cost Saving

We first analyze the saving of monetary costs in repair in practice. Table 3 shows the monthly price plans for three major providers as of May 2013. We take the cost from the first chargeable usage tier (i.e., storage usage within 1TB/month; data transferred out more than 1GB/month but less than 10TB/month). From the analysis in Section 3, we can save 25-50% of the download traffic during storage repair.

The storage size and the number of chunks being generated per file object are the same in both RAID-6 and FMSR codes (assuming that we aggregate chunks in FMSR codes as described in). However, in the analysis, we have ignored two practical considerations: the size of metadata (Section 5) and the number of requests issued during repair. We now argue that they are negligible and that the simplified calculations based only on file size suffice for real-life applications. Metadata size: Our implementation currently keeps the metadata size of FMSR codes within 160 bytes when n = 4 and k = 2, regardless of the file size. For a large n, say when n = 12 and k = 10, the metadata size

10 is still within 900 bytes. NCCloud aims at long-term backups, and can be integrated with other backup applications. Existing backup applications typically aggregate small files into a larger data chunk in order to save the processing overhead. For example, the default setting for Cumulus creates chunks of around 4MB each. Thus, the metadata size overhead can be made negligible. Since both RAID-6 and FMSR codes store the same amount of file data, they incur very similar storage costs in normal usage (assuming that the metadata costs are negligible). Number of requests providers charge for requests. RAID-6 and FMSR codes differ in the number of requests when retrieving data during repair. Suppose that we store a file object of size 4MB with n = 4 and k = 2. During repair, RAID-6 and FMSR codes retrieve two and three chunks, respectively. The cost overhead due to the GET requests for RAID-6 codes is at most 0.171%, and that for FMSR codes is at most 0.341%, a mere 0.17% increase.

6.2 Response Time Analysis

We deploy our NCCloud prototype in real environments. We evaluate the response time performance of three basic operations, namely file upload, file download, and repair, in two scenarios. The first part analyzes in detail the time taken by different NCCloud operations. It is done on a local cloud storage tested in order to lessen the effects of network fluctuations. The second part evaluates how NCCloud actually performs in a commercial cloud environment. All results are averaged over 40 runs. We assume that repair coefficients are generated offline , so the time taken by

ISSN : 2454-9924

two-phase checking is not accounted for in the repair operation.

7. RELATED WORK

We review the related work in multiple-cloud storage and failure recovery. Multiple-cloud storage. There are several systems proposed for multiple-cloud storage. HAIL provides integrity and availability guarantees for stored data. RACS uses erasure coding to mitigate vendor lock- ins when switching cloud vendors. It retrieves data from the cloud that is about to fail and moves the data to the new cloud. Unlike RACS, NCCloud excludes the failed cloud in repair. Vukoli´c advocates using multiple independent clouds to provide Byzantine fault tolerance. DEPSKY [10] addresses Byzantine fault tolerance by combining encryption and erasure coding for stored data. Single-node failure recovery schemes that minimize the amount of data read (or I/Os) for XOR-based erasure codes. For example, authors of [62], [63] propose opti- mal recovery for specific RAID-6 codes and reduce the amount of data read by up to around 25% (compared to conventional repair that downloads the amount of orig- inal data) for any number of nodes. Note that our FMSR codes can achieve 25% saving when the number of nodes is four, and up to 50% saving if the number of nodes increases. Authors of propose an enumeration-based approach to search for an optimal recovery solution for arbitrary XOR-based erasure codes. Efficient recovery is recently addressed in commercial cloud storage systems. For example, new constructions of non-MDS era- sure codes designed for efficient recovery are proposed for Azure and Facebook . The codes used in trade storage overhead for performance, and are mainly designed for data-intensive computing. Our work targets the cloud backup applications. Minimizing repair traffic. Regenerating codes stem from the concept of network codin2g and minimize the repair traffic among storage nodes. They exploit the optimal trade-off between storage cost and repair traffic, and there are two optimal points.

8 .CONCLUSIONS

We present NCCloud, a proxy-based, multiple-cloud storage system that practically addresses the reliability of today’s cloud backup storage. NCCloud not only provides fault tolerance in storage, but also allows cost-effective

ISSN : 2454-9924

repair when a cloud permanently fails. NCCloud implements a practical version of the functional minimum storage regenerating (FMSR) codes, which regenerates new parity chunks during repair subject to the required degree of data redundancy. Our FMSR code implementation eliminates the en- coding 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 NCCloud prototype shows the effectiveness of FMSR codes in the cloud backup usage, in terms of monetary costs and response times.

9. 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/case-studies/#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. Kon- winski, 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/case-studies/. [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.

ISSN : 2454-9924

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 code.