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

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 4, Issue 5, May 2015

NC-CLOUD: A NETWORK-CODING BASED DISRTIBUTED STORAGE SYSTEM IN A MULTI-CLOUD

Dr. Mohammed Abdul Waheed[1], Sushmita BN[2]

Associate Professor, Dept. of Computer Science and Engineering, VTU RO PG Centre, Kalaburgi, India[1]

IV SEM M.tech, Dept. of Computer Science and Engineering, VTU RO PG Centre, Kalaburgi, India[2]

Abstract: NC-Cloud is a proof of concept prototype of a network coding based storage system that aims at providing fault tolerance and reducing data repair cost when storing storage using multiple cloud storage. To provide fault tolerance for cloud storage, recent studies propose 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. We design a proxy based storage system for fault tolerant multiple cloud storage called NC-Cloud, which achieves reduction in repair traffic for a permanent multi-cloud failure. NC-Cloud is built on top of a network-coding-based storage scheme called regenerating codes. Specifically we propose an implementable design for 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. We validate that FMSR codes provide significant monetary cost savings in repair over RAID-6 codes, having comparable response time performance in normal cloud storage operations such as upload/download.

Keywords: Regenerating codes, Network coding, fault tolerance, Recovery, Implementation.

I. INTRODUCTION

With the increasing growth of data to be managed, distributed storage systems provide a reliable platform for storing massive amounts of data over a set of storage nodes that are distributed over a network. A real-life business model of distributed storage is cloud storage (e.g., Amazon S3 [3] and Windows Azure), which enables enterprises and individuals to outsource their data backups to third-party repositories in the Internet. One key feature of distributed storage is data consistency, which normally refers to the redundancy of data storage specifically, given the pre-determined level of redundancy the distributed storage system must sustain normal I/O operations within a tolerable number of node failures. In addition, in order to maintain the necessary redundancy, the storage structure must carry data repair, which involves reading data from existing nodes and reconstructing essential data in the new nodes.

Recent studies propose a class of fast data repair schemes based on network coding [2] for distributed storage systems. Such network-coding-based schemes, or called regenerating codes, seek to intelligently mix and combine data blocks in existing nodes, and renew data blocks at original nodes. It is theoretically shown that regenerating codes can improve the data repair performance over traditional redundancy approaches such as erasure codes (RAID-6).

In this paper, we propose NC-Cloud, a proxy-based system designed for multiple-cloud storage. We propose the first implementable design for the functional minimum-storage regenerating code (F-MSR) [6], and in particular, we Eliminate the need of performing encoding operations within storage nodes as in existing theoretical studies. Our F-MSR implementation maintains double-fault tolerance and has the same storage cost as in traditional erasure coding schemes based on RAID-6, but uses less repair traffic when recovering a single-cloud failure.

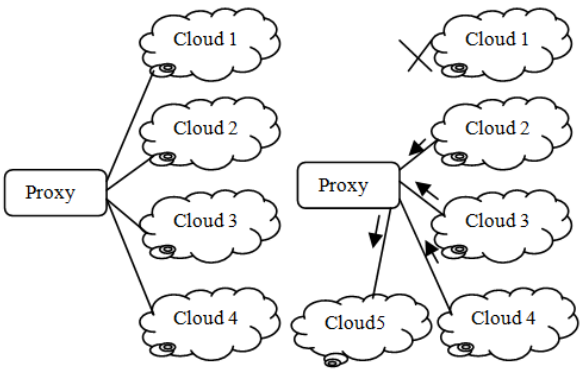
II. MOTIVATION OF F-MSR

We consider a distributed, multiple-cloud storage setting from a client's perception, such that we streak data over several cloud vendors. We propose a proxy-based design [1] that interconnects multiple cloud preservation, as shown in Figure 1(a). 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, as shown in Figure 1(b). That is, the proxy reads the important data pieces from other current clouds, reconstructs new data pieces, and writes these new pieces to a new cloud.

We consider fault-tolerant storage based on maximum distance separable (MDS) codes. Given a file object, we divide it into equal-size native chunks, which in a non-coded system, would be stored on k clouds. With coding, the native chunks are encoded by linear combinations to form code chunks. The native and code chunks are distributed over $n > k$ clouds. When an MDS code is used, the original file object may be reconstructed from the chunks contained in any k sf any $n - k$ clouds. We call this feature the MDS property. The extra feature of F-MSR is that reconstructing a single native or code chunk may be achieved by reading up to 50% less datas

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

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 4, Issue 5, May 2015



(a)Normal operation (b) Repair operation

Figure 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 from the surviving clouds than reconstructing the whole file This paper considers a multiple-cloud setting that is double-fault tolerant (e.g., RAID-6) and provides data availability toward at most two cloud failures (e.g., a few days of outages [7]). That is, we set k = n − 2.

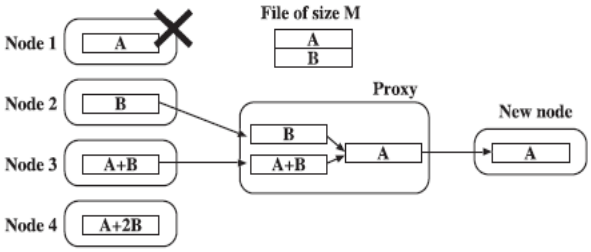
We now consider the double-fault tolerant implementation of F-MSR in a proxy-based setting, as shown in Figure 2(b). F-MSR divides the file into four native chunks, and constructs eight distinct code chunks P1, · · · , P8 formed by different linear combinations of the native chunks. Each code chunk has 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. 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 P1' and P2' and P0 2 formed by different linear combinations of the three code chunks. To generalize F-MSR for n storage nodes, we divide a file of size M into 2(n − 2) native chunks, and generate 4(n − 2) code chunks.

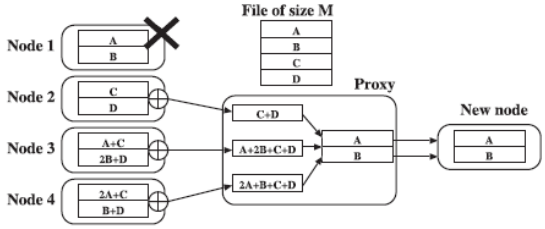
Then each node will store two code chunks of size M/ 2(n−2) each. Thus, the total storage size is Mn/n−2 . To repair a failed node, we download one chunk from each of n − 1 nodes, so the repair traffic is M (n−1)/2(n−2). This paper considers the baseline RAID-6 implementation using Reed-Solomon codes.

Its repair method is to reconstruct the whole file, and is applicable for all erasure codes in general. Recent studies [18, 28, 29] show that data reads can be minimized specifically for XOR based erasure codes.

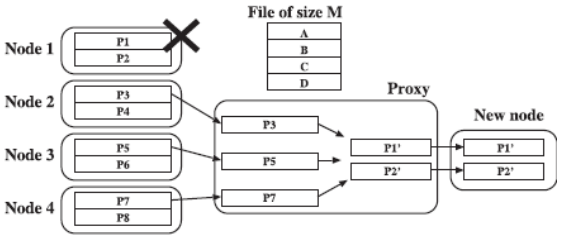
We can see that in EMSR codes, the storage size is 2M (same as RAID-6 codes), while the repair traffic which is 25 percent of saving (compared with RAID-6 codes). EMSR codes leverage the notion of network coding [2], as the nodes generate encoded chunks during repair.



(a)RAID-6 Codes



(b) EMSR Codes



(c)FMSR Codes

III. FMSR CODE IMPLEMENTATION

In this section, we present a systematic approach for implementing FMSR. We specify three operations for FMSR on a particular file object: (1) file upload (2) file download (3) repair. 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.

3.1 Basic Operations

3.1.1 File Upload

To upload a file F, we first divide it into k(n-k) equal-size native chunks, denoted by$(F_i)_{i=1,2.....k(n-k)}$.We then encode these k(n-k) native chunks into n(n-k) code chunks, denoted by $(P_i)_{i=1.2.....n(n-k)}$.Each P_i is formed by a linear combination of the k(n-k) native chunks. Specifically , we let EM=[α_{ij}] be an n(n-k)*k(n-k) encoding matrix for some coefficients α_{ij} (where i=1,...n(n-k) and j=1,..,k(n-k)) in the Galois field GF(2^8).We call a row vector of EM an encoding coefficient vector (EVC),which contains k(n-k) elements.

We compute each P_i by the product of ECV_i and ll the native chunks F_1, $F_{2,.....,}F_{k(n-k),}$ i.e,. $P_i=\sum^{k(n-k)}_{j=1}\ \alpha_{i,j}\ F_j$ for i=1,2,...,n(n-k),where all arithmetic operations are performed over GF(2^8). 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.

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

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 4, Issue 5, May 2015

3.1.2 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.

3.1.3 Iterative Repairs

We now consider the repair of F-MSR for a file F for a permanent single-node failure. Given that F-MSR regenerates different chunks in each repair, one challenge is to ensure that the MDS property still holds even after iterative repairs. . 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 r^{th} 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 r^{th} 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 occur (we call this the repair MDS property). We now describe the r^{th} repair as follows:

Step 1: Download the encoding matrix from a surviving node. Recall that the encoding matrix EM specifies the ECVs for constructing all code chunks via linear combinations of native chunks. We use these ECVs for our later two-phase checking heuristic. Since we embed EM in a metadata object that is replicated, we can simply download the metadata object from one of the surviving nodes.

Step 2: Select one random ECV from each of the n − 1 surviving nodes. Note that each ECV in EM corresponds to a code chunk. We pick one ECV from each of the n-1 surviving nodes. We call these ECVs to be $ECV_{i1,}$ $ECV_{i2,...,}$ ECV_{in-1}.

Step 3: Generate a repair matrix. We construct an (n-k)*(n-1) repair matrix RM=[Υ_{ij}], where each element Υ_{ij} (where. i=1,...,n-k and j=1,...,n-1) is randomly matrix for reliable storage is consistent with that in.

Step 4: Compute the ECVs for the new code chunks and reproduce a new encoding matrix. We multiply RM with the ECVs selected in Step 2 to construct n − k new ECVs, denoted by $ECV'_i=\sum^{n=1}_{j=1}\Upsilon_{i,j}$ ECV_{ij} for i=1,2,....n-k. Then we reproduce a new encoding matrix, denoted by EM' which is formed by substituting the ECVs of EM of the failed node with the corresponding new ECVs.

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

Step 6: Download the actual chunk data and regenerate new chunk data. If the two-phase checking in Step 5 succeeds, then we proceed to download the n−1 chunks that correspond to the selected ECVs in Step 2 from the n-1 surviving storage nodes to NC-Cloud. Also, using the new ECVs computed in Step 4, we regenerate new chunks and upload them from NC-Cloud to a new node.

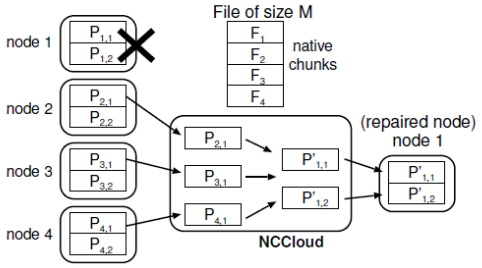


Fig. 1. FMSR codes with $n = 4$ and $k = 2$.

IV. FMSR CODES PRESERVE THE FAULT TOLERANCE AND STORAGE EFFICIENCY OF MDS CODES.

MDS codes are defined by two parameters n and k (k < n). An (n, k)-MDS code divides a file of size M into k pieces of size M/k each, and encodes them into n pieces such that any k out of n encoded pieces suffice to recover the original file. By storing the n encoded pieces over n nodes, a storage system can tolerate at most n − k node failures. An example of MDS codes is Reed-Solomon codes.

Figure 1 shows the FMSR codes for a special case n = 4 and k = 2. To store a file of size M units, an (n, k)-FMSR code splits the file evenly into k(n − k) native chunks, say $F_1, F_2, \ldots, F_{k(n-k),}$ and encodes them into n(n − k) parity chunks of size M/k(n−k) each. Each l^{th} parity chunk is formed by a linear combination of the k(n − k) native chunks, i.e., $\sum^{K(n-k)}_{m=1}$ $\alpha_{l,m}$ Fm for some encoding coefficients $\alpha_{l,m}$. All encoding coefficients and arithmetic are operated over a finite field F_q of size q. We store the n(n − k) parity chunks on n nodes, each keeping n − k parity chunks. Note that native chunks need not be stored. The original file can be restored by decoding k(n − k) parity chunks of any k nodes, where decoding can be done by inverting an encoding matrix [16]. Let $P_{i,j}$ be the j^{th} parity chunk stored on node i, where i = 1, 2, . . . , n and j = 1, . . . , n − k.

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

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 4, Issue 5, May 2015

V. NC-CLOUD 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 F-MSR).

VI. CONCLUSION

In this NC-Cloud we proposed a multi node failure recovery using network coding (FMSR) method. The NCCLOUD we present a proxy-based, multiple cloud storage system that practically addresses the reliability of today's cloud backup storage system. NCCLOUD provides a recovery as well as fault tolerance in storage; NC-Cloud presents a practical version of the FMSR codes, which generates parity chunks when node becomes failure. Our NC-Cloud prototype shows the effectiveness of FMSR codes in the backup usage in terms of response times.

ACKNOWLEDGMENT

I would like to thank our Guide **Dr. Mohammed Abdul Waheed.** For assisting me in this Paper.

REFERENCES

[1]. 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

[2]. K. D. Bowers, A. Juels, and A. Oprea. HAIL: A High-Availability and Integrity Layer for Cloud Storage. In Proc. of ACM CCS, 2009

[3]. B. Escoto and K. Loafman. Duplicity. http:// duplicity.nongnu.org/.

[4]. 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.

[5]. 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.

[6]. H.C.H. Chen and P.P.C. Lee, "Enabling Data Integrity Protection in Regenerating-Coding-Based Cloud Storage," Proc. IEEE 31st Int'l Symp. Reliable Distributed Systems (SRDS '12), 2012.

[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

[8]. A.G. Dimakis, K. Ramchandran, Y. Wu, and C. Suh, "A Survey on Network Codes for Distributed Storage," Proc. IEEE, vol. 99, no. 3, pp. 476-489, Mar. 2011.

[9]. A. Duminuco and E. Biersack, "A Practical Study of Regenerating Codes for Peer-to-Peer Backup Systems," Proc. IEEE Int'l Conf. Distributed Computing Systems (ICDCS '09), 2009.

[10]. 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.

[11]. Y. Hu, Y. Xu, X. Wang, C. Zhan, and P. Li, "Cooperative Recovery of Distributed Storage Systems from Multiple Losses with Network Coding," IEEE J. Selected Areas in Comm., vol. 28, no. 2, pp. 268-276, Feb. 2010.

[12]. N. Kolakowski, "Microsoft's Cloud Azure Service SuffersOutage,"http://www.eweekeurope.co.uk/news/news-solutionsapplications

[13]. 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

[14]. [14] Y. Hu, H.C.H. Chen, P.P.C. Lee, and Y. Tang, NCCloud:Applying Network Coding for the Storage Repair in a Cloudof-Clouds," Proc. 10th USENIX Conf. File and Storage Technologies(FAST), 2012.

[15]. K. Rashmi, N. Shah, and P. Kumar, "Optimal Exact Regenerating Codes for Distributed Storage at the MSR and MBR Points via aProduct-Matrix Construction," IEEE Trans. Information Theory,vol. 57, no. 8, pp. 5227-5239, Aug. 2011.

[16]. 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.

[17]. Z. Wang, A. Dimakis, and J. Bruck, "Rebuilding for Array Codes in Distributed Storage Systems," Proc. IEEE Globe Com Workshops,2010.

[18]. F. Oggier and A. Datta, "Byzantine Fault Tolerance of Regenerating Codes," Proc. IEEE Int'l Conf. Peer-to-Peer Computing (P2P '11),2011.