Secured Public Auditing Scheme for Regenerating-Code-Based Cloud Storages

R. Sugumar[1], Dr.A.Rajesh[2], Deepa.C[3]

[1]R.Sugumar, Research Scholar, CSE Department, /SCSVMV University, Kanchipuram, India,
[1]sugumar_prof@rediffmail.com.
[2]Dr.A. Rajesh, Prof. and Head, CSE Department,/C.Abdul Hakeem College of Engineering and Technology, Melvisharam, Vellore, India,
[2]amrajesh73@gmail.com.
[3]Deepa.C, Computer Science and Engineering/ C. Abdul Hakeem College of Engineering and Technology, Melvisharam, Vellore, India
[3]Deepa.cse72@gmail.com

ABSTRACT

In cloud storage service the user data is outsourced in which it provides control over security problem towards the correctness of the data stored in the cloud. By adding fault tolerance to cloud storage with data integrity checking and failure reparation becomes critical. Regeneration codes have lower repair bandwidth by providing fault tolerance. Regenerating codes provide recovering codes by striping data across multiple servers, while using less repair traffic than traditional recover codes during failure recovery. Existing remote checking method for regenerating-coded data provide private auditing, which requires data owners to always stay online and handle auditing and repairing which is sometimes impossible. Therefore, the problem of remotely checking the integrity of regenerating-coded data against fault occurrence under a real-life cloud storage setting. To design and implement a handy data integrity protection (DIP) scheme for a particular recovering code, while protecting its inherent properties of adaptation to internal failure and repair-movement spares. DIP scheme is designed under a mobile Byzantine adversarial model, and empowers a user to practically confirm the respectability of arbitrary subsets of outsourced information against general or pernicious defilements. It works under the straightforward supposition of dainty distributed storage and permits distinctive parameters to be adjusted for an execution security exchange off. Actualize and assess the overhead of DIP plan in a genuine distributed cloud storage testbed under various parameter decisions. In further analyze the security strengths of DIP scheme via mathematical models. Demonstrate that remote integrity checking can be feasibly integrated into regenerating codes in practical deployment.

Keywords — remote checking, internal failure, integrity checking

1.INTRODUCTION

Cloud storage is currently picking up fame since it offers an adaptable on-interest information outsourcing administration with engaging advantages: alleviation of the weight for capacity administration, all inclusive information access with area freedom, and evasion of capital consumption on equipment, programming, and individual systems for upkeeps, and so on., [1]. By this new worldview of information facilitating benefit additionally brings new security dangers toward clients information, accordingly making people or enterprisers still feel reluctant.

Cloud storage offer on-interest information outsourcing administration demonstrate, and is picking up ubiquity because of its versatility and low support cost. Be that as it may, security concerns emerge when information stockpiling is outsourced to outsider distributed storage suppliers. It is attractive to empower cloud customers to check the uprightness of their outsourced information, on the off chance that their information have been incidentally undermined or vindictively traded off by insider/outcast assaults.

One noteworthy utilization of distributed storage is long haul chronicled, which speaks to a workload that is composed once and once in a while read. While the put away information are once in a while read, it stays important to guarantee its honesty for calamity recuperation or consistence with legitimate prerequisites. Since it is regular to have colossal measure of chronicled information, entire document checks gets to be restrictive. Proof of retrievability(POR)[19] and provable Data possession(PDP) [3] have along these lines been proposed to confirm the uprightness of a substantial document by spot-checking just a small amount of the record through different crypto-realistic primitives.

It is noticed that information proprietors lose extreme control over the destiny of their outsourced information; therefore, the accuracy, accessibility and honesty of the information are being put at danger. From one perspective, the cloud administration is generally confronted with an expansive scope of inner/outer enemies, who might malevolently erase or degenerate clients' information; then again, the cloud administration suppliers might act deceptively, endeavoring to conceal information misfortune or defilement and asserting that the records are still accurately put away in the cloud for notoriety or fiscal reasons. Therefore it bodes well for clients to execute an effective convention to perform periodical checks of their outsourced information to guarantee that the cloud to be sure keeps up their information accurately.

Numerous instruments managing the trustworthiness of outsourced information without a neighborhood duplicate have been proposed under various framework and security models up to now. The most huge work among these studies are the PDP (provable Data possession) model[3] and POR (proof of retrievability) model[19], which were initially proposed for the single-server situation. Considering that records are typically

striped and needlessly put away crosswise over multi-servers or multi-mists, [4]–[10] investigate honesty check plans suitable for such multi-servers or multi-mists setting with various excess plans, for example, replication, eradication codes, and,more as of late, recovering codes.

Assume that the outsource capacity to a server, which could be a capacity site or a distributed storage supplier. On the off chance to identify debasements in outsourced information (e.g., when a server crashes or is traded off), then ought to repair the adulterated information and restore the first information. Be that as it may, putting all information in a solitary server is helpless to the single-purpose of-disappointment issue[2] and merchant lock-ins[5]. As proposed in[5],[2], a conceivable arrangement is to stripe information over different servers. In this way, to repair a fizzled server, that are 1)read information from the other surviving servers, 2)reproduce the tainted information of the fizzled server, and 3)compose the recreated information to another server. POR[19] and PDP[3] are initially proposed for the single-server case. MR-PDP[9] stretch out trustworthiness checks to a multiserver setting utilizing replication and deletion coding, separately. Specifically, deletion coding (e.g., Reed-Solomon codes[21]) has a lower stockpiling overhead than replication under the same adaptation to internal failure level.

Field estimations[17],[22],[23] demonstrate that vast scale stockpiling frameworks generally encounter plate/area disappointments, some of which can bring about changeless information misfortune. For instance, the annualized substitution rate for circles underway capacity frameworks is around 2-4 percent. Information misfortune occasions are additionally found in business distributed storage administrations. With the exponential development of authentic information, a little disappointment rate can infer noteworthy information misfortune in recorded stockpiling. This persuades us to investigate elite recuperation to lessen the window of weakness. Recovering codes[6] have as of late been proposed to minimize repair activity (i.e., the measure of information being perused from surviving servers). Basically, they accomplish this by not perusing and recreating the entire record amid repair as in customary deletion codes, yet rather perusing a set of pieces littler than the first document from other surviving servers and recreating just the lost (or ruined) information lumps. An open inquiry is, would be able to empower uprightness checks on recovering codes, while safeguarding the repair activity sparing over conventional deletion codes?

A related methodology is HAIL[6], which applies honesty insurance for deletion codes. It develops assurance information on for each document premise and conveys the security information crosswise over various servers. To repair any lost information amid a server disappointment, one needs to get to the entire document and this damages the outline of recovering codes. Therefore, it requires an alternate outline of uprightness assurance customized for recovering codes. In this paper, the outline and actualize a down to data integrity protection(DIP) plan for recovering coding-based distributed storage. It increase the execution of utilitarian least stockpiling recovering (FMSR) codes[18] and develop FMSR-DIP codes, which permit customers to remotely confirm the trustworthiness of irregular subsets of long haul recorded information under a multiserver setting. FMSR-DIP codes protect adaptation to non-critical failure and repair activity sparing as in FMSR codes [15]. Likewise, it expect just a dainty cloud interface [20], implying that servers just need to bolster standard read/compose functionalities. This adds to the transportability of FMSR-DIP codes and permits straightforward sending by and large sorts of capacity administrations. By joining honesty checking and effective recuperation, FMSR-DIP codes give a minimal effort answer for keeping up information accessibility in distributed storage. In synopsis, it make the accompanying commitments:

FMSR-DIP codes, which empower honesty security, adaptation to non-critical failure, and productive recuperation for distributed storage. A few tunable parameters from FMSR-DIP codes, such that customers can make an exchange off in the middle of execution and security. Direct numerical examination on the security of FMSR-DIP codes for various parameter decisions. To execute FMSR-DIP codes, and assess their overhead over the current FMSR codes through broad testbed tests in a distributed storage environment. The running times of various essential operations, including Upload, Check, Download, and Repair, for various parameter decisions.

In this paper, they concentrate on the trustworthiness confirmation issue in recovering code-based distributed storage, particularly with the practical repair technique [11]. Comparative studies have been performed by Chen et al. [7] and Chen and Lee [8] independently and autonomously. [7] broadened the single-server CPOR plan (private rendition in [12]) to the recovering code-situation; [8] planned and executed an information uprightness insurance (DIP) plan for FMSR [13]-based distributed storage and the plan is adjusted to the slight cloud setting. Be that as it may, them two are intended for private review, just the information proprietor is permitted to check the honesty and repair the flawed servers. Considering the extensive size of the outsourced information and the client's obliged asset ability, the undertakings of evaluating and reparation in the cloud can be impressive and costly for the clients [14]. The overhead of utilizing distributed storage ought to be minimized however much as could be expected such that a client does not have to perform an excess of operations to their outsourced information (in extra to recovering it) [15]. Specifically, clients might not have any desire to experience the multifaceted nature in checking and reparation.

The evaluating plans in [7] and [8] suggest the issue that clients need to dependably stay on the web, which might hinder its reception by and by, particularly for long haul chronicled capacity. To completely guarantee the information uprightness and recovery the clients' calculation assets and additionally online weight, it propose an open examining plan for the recovering code-based distributed storage, in which the trustworthiness checking and recovery (of fizzled information pieces and authenticators) are executed by an outsider evaluator and a semi-trusted intermediary independently for the benefit of the information proprietor. Rather than specifically adjusting the current open evaluating plan [12] to the multi-server setting, the outline of the novel authenticator, which is more proper for recovering codes. In addition, "scramble" the coefficients to secure information protection against the reviewer, which is more light weight than applying the verification blind procedure in [14] and [15] and information blind strategy in [16].

3. SYSTEM ARCHITECTURE

The reviewing framework model for Regenerating-Code-based cloud storage as Fig.1, which includes four substances: the information proprietor, who possesses a lot of information

records to be put away in the cloud; the cloud, which are overseen by the cloud administration supplier, give stockpiling benefit and have critical computational assets; the Third Party Auditor(TPA), who has mastery and capacities to direct open reviews on the coded information in the cloud, the TPA is trusted and its review result is fair-minded for both information proprietors and cloud servers; and an intermediary specialists, who is semi-trusted and follows up in the interest of the information proprietor to recover authenticators and information obstructs on the fizzled servers amid the repair method.

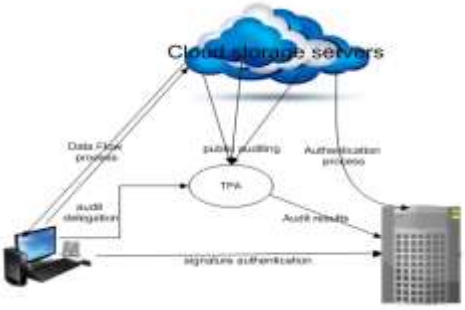

Fig 1.System Model

Notice that the information proprietor is limited in computational and capacity assets contrasted with different substances and might gets to be logged off even after the information transfer strategy. The intermediary, who might dependably be on the web, should be a great deal more effective than the information proprietor yet not exactly the cloud servers regarding calculation and memory limit. To spare assets and additionally the online weight conceivably brought by the occasional inspecting and inadvertent repairing, the information proprietors resort to the TPA for uprightness check and delegate the reparation to the intermediary.

Contrasted and the conventional open examining framework shows, the framework model includes an extra intermediary specialists. An organization utilizes a business recovering code-based open cloud and gives long haul recorded capacity administration for its staffs, the staffs are outfitted with low end calculation gadgets (e.g., Laptop PC, Tablet PC, and so forth.) and will be oftentimes disconnected from the net. For open information examining, the organization depends on a trusted outsider association to check the information uprightness; similarly, to discharge the staffs from substantial online weight for information and authenticator recovery, the supply to an intense workstation (or group) as the intermediary and give intermediary reparation administration to the information. In this framework it ensure outsourced information in distributed storage against defilements, adding adaptation to internal failure to distributed storage, alongside productive information trustworthiness checking and recuperation strategies, gets to be basic.

Recovering codes give adaptation to non-critical failure by striping information over various servers, while utilizing less repair movement than conventional eradication codes amid disappointment recuperation. In DIP plan for a particular recovering code, while saving its natural properties of adaptation to non-critical failure and repair-activity sparing. Plunge plan is outlined under a portable Byzantine antagonistic model, and empowers a customer to attainably check the uprightness of irregular subsets of outsourced information against general or vindictive defilements. It works under the straightforward suspicion of dainty distributed storage and permits distinctive parameters to be calibrated for an execution security exchange off. Further break down the security qualities of DIP plan by means of scientific models. Show that remote trustworthiness checking can be possibly coordinated into recovering codes in pragmatic sending

4.SYSTEM IMPLEMENTATION

1.Storage Data in Thin-cloud
2.Upload Data File and Metadata File.
3.Download and decode the needed chunks based on FMSR-DIP
4.Row verification for downloaded Chunk file.

4.1 STORAGE DATA IN THIN-CLOUD

In this module, initial tend to expand a REST-full interface that embody the directions place and acquire. Place permits writing to a file as a entire (no restricted updates), and acquire permits reading(scanning) from a specific vary of bytes from file via a spread GET request. DRS theme uses individually the place and acquires information to work with every cloud server. Thin-cloud setting permits DRS theme to be convenient to general styles of storage procedure or services, since no execution changes area unit needed on the storage backend. It differs from different "thick" cloud-storage services wherever servers have automatic capabilities and area unit capable of aggregating the proofs of multiple checks. There can't be any limits on the quantity of probable challenges that the shopper will build, the files is reserved for ensuring repository. Also, the brave size ought to be variable with totally different limitation selections, and this is often helpful once the lower invention rate once the keep knowledge grow minor over time.

4.1.1.Algorithms implemented

i. Rivest-Shamir-Adleman Security Scheme **(RSASS)**

Algorithm: Generate an RSA key pair.
INPUT: Required modulus bit length, *k*.
OUTPUT: An RSA key pair ((N,e), d) where N is the modulus, the product of two primes (N=pq) not exceeding *k* bits in length; e is the public exponent, a number less than and coprime to (p-1)(q-1); and d is the private exponent such that $ed \equiv 1 \pmod{(p-1)(q-1)}$.

1. Select a value of *e* from {3, 5, 17, 257, 65537}
2. repeat
3. p ← genprime(k/2)
4. until (p mod e) ≠ 1
5. repeat
6. q ← genprime(k - k/2)
7. until (q mod e) ≠ 1
8. N ← pq
9. L ← (p-1)(q-1)
10. d ← modinv(e, L)
11. return *(N, e, d)*

ii. Elliptic Curve Cryptosystem Security Scheme

ECCSS scheme consists of the following four algorithms.

KeyGen

Input: None
Output: public key p, private key x and point Q.
1) The client will run the algorithm which will generate the public key p and the private key x.
2) A point Q is chosen on the elliptic curve E (K), where K is a finite field.
3) It selects a pseudo random number x such that

$1 \leq x \leq (n_A-1)$.
4) Point p =xQ.
5) ECC key pair is (p, x), where p is the public key, x is the private key.

4.2 UPLOAD DATA FILE AND METADATA FILE

To minimize the key organization structure within the clouds, to develop multiple keys from one secret proximity key supply functions, as comprehensive in previous studies and standards. Additionally, to cut back the native storage load, to encipher all file keys with a passkey, and source the storage of the encrypted keys to the cloud. Since the files within the cloud area unit characteristically of huge size, tend to expect that the keys solely acquire a little constant within the clouds. the tend to conjointly append the MACs of all chunks to the information. Finally, the information is encrypted with ENC and virtual to every server to contribute solely a little storage within the clouds.

Algorithm Used:

Message processing code (MPC) technique is used in uploading the data file.

In cryptography, a message processing code (MPC) is a short bit of data used to verify a message—at the end of the day, to give honesty and credibility certifications on the message. Uprightness certifications identify unplanned and purposeful message changes, while realness affirmations assert the message's root. A MPC algorithm, called a keyed (cryptographic) hash function (be that as it may, cryptographic hash function is one of the conceivable approaches to create MPCs), acknowledges as info a mystery key and a subjective length message to be verified, and yields a MAC (known as a tag). The MPC esteem ensures both a message's information trustworthiness and additionally its realness, by permitting verifiers (who likewise have the mystery key) to distinguish any progressions to the message content.

4.3 DOWNLOAD AND DECODE THE NEEDED CHUNKS BASED ON FMSR-DIP

Transfer and decrypt the required chunks supported FMSR-DRS. The PRFs off the FMSR-DRS code chunks to selection the FMSR code chunks, that area unit then passed to NCCloud for cryptography if they're not corrupted. However, if it got a corrupted code chunk, then to fix it with one in all the subsequent criteria, obtain its AECC(Adversarial Error Correcting Code) parities and have an effect on error modification. Then it tends to attest the corrected chunk with its cover once more. Transfer the code chunks from another server. A last various is to transfer the code chunks from all n servers. To check all rows of the chunks as well as their AECC parities. The rows with a set of the bytes accurate is improved with FMSR codes; the rows with all bytes obvious corrupted area unit treated as erasures and can be recovered with AECC. In specific, if there's only 1 unsuccessful server, then rather than attempting to transfer K(n-k) chunks from any k servers, to transfer one chunk from all remaining (n – 1) servers as in FMSR codes.

4.4 ROW VERIFICATION FOR DOWNLOADED CHUNK FILE

Row verification for downloaded Chunk file . Probabilistic row authentication within the Check procedure. Note that there's a trade-off of selecting what number bytes to corrupt. a better corruption rate means the opponent will alter additional bytes in an exceedingly strip, however the corruption is additionally easier to be detected by row verification. The purpose is to produce as applied math structure that analyzes the militia of FMSR-DRS codes for uncommon constraint selections.

5.RESULT DISCUSSION

The evaluation results of the running time over-head of FMSR-DIP codes for the Download and Repair operations, and also analyze the monetary cost overhead with the pricing models of different commercial cloud providers.

Setup: First conduct testbed experiments on a local cloud platform that is built on OpenStack Swift. To deploy FMSR-DIP on a machine equipped with two Intel Xeon E5530 Quad-Core CPUs (i.e., a total of eight cores), 16-GB RAM, and 64-bit Ubuntu 11.04. The machine is connected via a Gigabit switch to an OpenStack Swift platform that is attached with 15 nodes. To create multiple containers on Swift, such that each container mimics a storage server.

To measure the running time of each operation. Let's assume that all file objects being processed remain intact (i.e., without corruptions) throughout an operation, so that it can measure the overhead of FMSR-DIP codes in normal usage. The results are averaged over 40 runs. It exploit parallelism in implementation. The spawn a DIP process for processing each FMSR-DIP code chunk (i.e., encoding an FMSR code chunk into an FMSR- DIP code chunk, or decoding an FMSR-DIP code chunk into an FMSR code chunk). All DIP processes are executed concurrently on eight-core testbed. Parallel implementation can achieve up to eight-fold speedup compared with sequential evaluations depending on the number of chunks that are needed to be processed. For the baseline evaluations of FMSR-DIP codes in sequential mode.

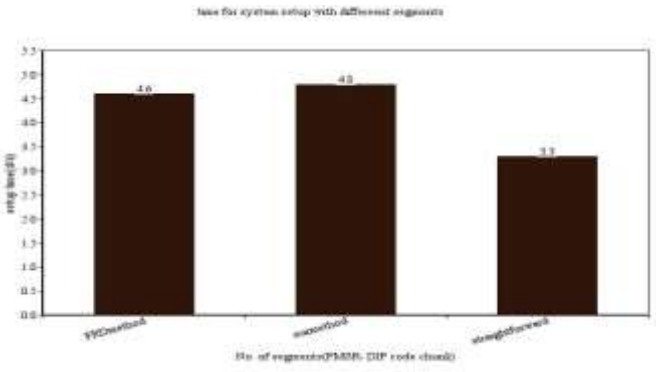

Fig 2.Time of system setup with different segments

Upload: To investigate the effects of four sets of parameters on the running time of the Upload operation, including 1) the input file size, 2) the parameters of FMSR codes, 3) the parameters of AECC, and 4) the block sizes of PRP and PRF. They vary one set of these parameters each time, while fixing the other three sets at default values. By default, the use of 100-MB file, (4, 2)-FMSR code, (110, 100)-AECC, and a block size of 256 B for both PRP and PRF. Further break down each running time result into three parts denoted by different labels: 1) “FMSR”, the time of encoding a file into FMSR code chunks by NCCloud, 2) “DIP-Encode”, the time of encoding the FMSR code chunks with DIP scheme, and 3) “Transfer-Up”, the network transfer time of uploading FMSR-DIP code chunks and metadata to the local cloud. By observing the fractional overhead of DIP encoding increases with the file size, and it ranges from 3.76 percent (for 1 MB)

to 9.92 percent (for 100 MB) of the overall time of Upload. The reason is that for larger files, the connection setup overhead of the data transmission becomes less dominant. The expected fractional overhead of DIP encoding to be smaller when the servers are deployed over the Internet, where the transmission time plays a major role in the Upload operation. The DIP encoding time increases with the redundancy level (i.e., the ratio of the amount of the redundant data being stored to that of the original data) of each of the underlying FMSR codes and AECC.

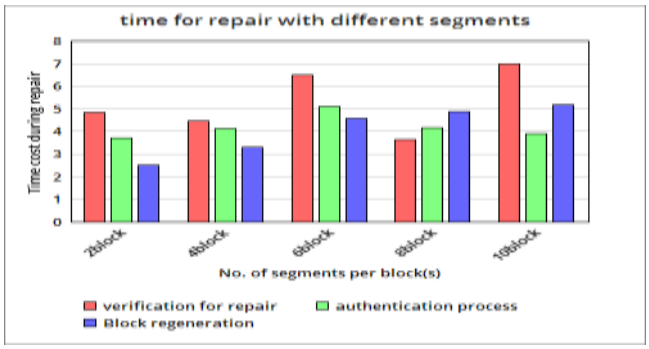

Fig 3.Time for repair with different segments

Check: To evaluate the effects of 1) the check block size, 2) the checking percentage, and 3) the parameters of FMSR codes in the Check operation. Apart from the default parameters in the evaluations of the Upload operation, and also use a check block size of 4 KB and a checking percentage of 1 percent by default. In evaluating the effect of checking percentage, to use check block size of 256 KB.

6.CONCLUSION

Given the prominence of outsourcing documented stockpiling to the cloud, it is attractive to empower customers to confirm the uprightness of their information in the cloud. In this system, public auditing scheme for the regenerating-code-based cloud storage system, where the data owners are privileged to delegate TPA for their data validity checking. To protect the original data privacy against the TPA, The outline and actualize a DIP plan for the FMSR codes under a multiserver setting. To develop FMSR-DIP codes, which protect the adaptation to internal failure and repair activity sparing properties of FMSR codes. To comprehend the reasonableness of FMSRDIP codes, the analysis of security quality by means of scientific demonstrating and assess the running time overhead by means of testbed investigations.

REFERENCES

[1] M. Armbrust et al. , "Above the clouds: A Berkeley view of cloud computing," Dept. Elect. Eng. Comput. Sci., Univ. California, Berkeley, CA, USA, Tech. Rep. UCB/EECS-2009-28, 2009.

[2] 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. ACM, vol. 53, no. 4, pp 50-58, 2010.

[3] G. Ateniese, R. Burns, R. Curtmola, J. Herring, O. Khan, L. Kissner, Z. Peterson, and D. Song, "Remote Data Checking Using Provable Data Possession," ACM Trans. Information and System Security, vol. 14, article 12, May 2011.

[4] R. Curtmola, O. Khan, R. Burns, and G. Ateniese, "MR-PDP: Multiple-replica provable data possession," in Proc. 28th Int. Conf. Distrib. Comput. Syst. (ICDCS), Jun. 2008, pp. 411–420.

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

[6] K. 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.

[7] B. Chen, R. Curtmola, G. Ateniese, and R. Burns, "Remote data checking for network coding-based distributed storage systems," in Proc. ACM Workshop Cloud Comput. Secur. Workshop, 2010, pp. 31–42.

[8] H. C. H. Chen and P. P. C. Lee, "Enabling data integrity protection in regenerating-coding-based cloud storage: Theory and implementation," IEEE Trans. Parallel Distrib. Syst., vol. 25, no. 2, pp. 407–416,Feb. 2014.

[9]R. Curtmola, O. Khan, R. Burns, and G. Ateniese, "MR-PDP: Multiple-Replica Provable Data Possession," Proc. IEEE 28th Int'l Conf. Distributed Computing Systems (ICDCS '08), 2008.

[10] Y. Zhu, H. Hu, G.-J. Ahn, and M. Yu, "Cooperative provable data possession for integrity verification in multicloud storage," IEEE Trans. Parallel Distrib. Syst.
, vol. 23, no. 12, pp. 2231–2244, Dec. 2012.

[11] 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.

[12] H. Shacham and B. Waters, "Compact proofs of retrievability," in Advances in Cryptology . Berlin, Germany: Springer-Verlag, 2008, pp. 90–107.

[13] Y. Hu, H. C. H. Chen, P. P. C. Lee, and Y. Tang, "NCCloud: Applying network coding for the storage repair in a cloud-of-clouds," in Proc. USENIX FAST , 2012, p. 21.

[14] C. Wang, Q. Wang, K. Ren, and W. Lou, "Privacy-preserving public auditing for data storage security in cloud computing," in Proc. IEEE INFOCOM, Mar. 2010, pp. 1–9.

[15] C. Wang, S. S. M. Chow, Q. Wang, K. Ren, and W. Lou, "Privacy-preserving public auditing for secure cloud storage," IEEE Trans. Comput., vol. 62, no. 2, pp. 362–375, Feb. 2013.

[16] C. Wang, Q. Wang, K. Ren, N. Cao, and W. Lou, "Toward secure and dependable storage services in cloud computing," IEEE Trans. Service Comput., vol. 5, no. 2, pp. 220–232, Apr./Jun. 2012.

[17] A. Dimakis, P. Godfrey, Y. Wu, M. Wainwright, and K. Ramchandran, "Network Coding for Distributed Storage Sys-tems," IEEE Trans. Information Theory, vol. 56, no. 9, 4539-4551, Sept. 2010.

[18] Y. Hu, H. Chen, P. Lee, and Y. Tang, "NCCloud: Applying Network Coding for the Storage Repair in a Cloud-of-Clouds," Proc. 10th USENIX Conf. File and Storage Technologies (FAST '12), 2012.

[19] A. Juels and B. Kaliski Jr., "PORs: Proofs of Retrievability for Large Files,"Proc. 14th ACM Conf. Computer and Comm. Security (CCS '07), 2007.

[20] M. Vrable, S. Savage, and G. Voelker, "Cumulus: File system Backup to the Cloud," Proc. USENIX Conf. File and Storage Technologies (FAST), 2009.

[21] I. Reed and G. Solomon, "Polynomial Codes over Certain Finite Fields," J. Soc. Industrial and Applied Math., vol. 8, no. 2, pp. 300-304, 1960.

[22] B. Schroeder, S. Damouras, and P. Gill, "Understanding Latent Sector Errors and How to Protect against Them," Proc.

USENIX Conf. File and Storage Technologies (FAST '10), Feb. 2010.
[23] B. Schroeder and G.A. Gibson, "Disk Failures in the Real World: What Does an MTTF of 1,000,000 Hours Mean to You?" Proc. Fifth USENIX Conf. File and Storage Technologies (FAST '07), Feb. 2007.

