Vol. 3 No. 2 October 2015 ISSN: 2321 – 788X

ENABLING DATA INTEGRITY PROTECTION IN REGENERATING-CODING-BASED CLOUD STORAGE

Mrs.T.R.Vithya*, Mrs.K.K.Kavitha & Mrs.V.Ramya*****

**Assistant Professor*

***HOD/Assistant Professor*

****Research Scholar*

Department of CS, Selvamm College of Arts & Science (Autonomous) Namakkal

Abstract

To protect outsourced data in cloud storage against corruptions, adding fault tolerance to cloud storage, along with efficient data integrity checking and recovery procedures, becomes critical. Regenerating codes provide fault tolerance by striping data across multiple servers, while using less repair traffic than traditional erasure codes during failure recovery. Therefore, we study the problem of remotely checking the integrity of regenerating-coded data against corruptions under a real-life cloud storage setting. We design and implement a practical data integrity protection (DIP) scheme for a specific regenerating code, while preserving its intrinsic properties of fault tolerance and repair-traffic is saving. Our DIP scheme is designed under a mobile Byzantine adversarial model, and enables a client to feasibly verify the integrity of random subsets of outsourced data against general or malicious corruptions. It works under the simple assumption of thin-cloud storage and allows different parameters to be fine-tuned for a performance-security trade-off. We implement and evaluate the overhead of our DIP scheme in a real cloud storage test bed under different parameter choices. We further analyze the security strengths of our DIP scheme via mathematical models. We demonstrate that remote integrity checking can be feasibly integrated into regenerating codes in practical deployment.

Index Terms—remote data checking secure and trusted storage systems, implementation, experimentation

Introduction

Cloud storage offers an on-demand data outsourcing service model, and is gaining popularity due to its elasticity and low maintenance cost. However, security concerns arise when data storage is outsourced to third party cloud storage providers. It is desirable to enable cloud clients to verify the integrity of their outsourced data, in case their data have been accidentally corrupted or maliciously compromised by insider/outsider attacks.

One major use of cloud storage is long-term archival, which represents a workload that is written once and rarely read. While the stored data are rarely read, it remains necessary to ensure its integrity for disaster recovery or compliance with legal requirements. Since it is typical to have a huge amount of archived data, whole-file checking becomes prohibitive. Proof of retrievability (POR) and proof of data possession (PDP) have thus been proposed to verify the integrity of a large file by spot checking only a fraction of the file via various cryptographic primitives.

Suppose that we outsource storage to a server, which could be a storage site or a cloud-storage provider. If we detect corruptions in our outsourced data (e.g., when a server crashes or is compromised), then we should repair the corrupted data and restore the original data. However, putting all data in a single server is susceptible to the single point- of-failure problem and vendor lock-ins. As suggested in a plausible solution is to stripe data across multiple servers. Thus, to repair a failed server, we can

1) Read data from the other surviving servers,

2) Reconstruct the corrupted data of the failed server, and

3) Write the reconstructed data to a new server.

POR and PDP are originally proposed for the single-server case. MR-PDP and HAIL extend integrity checks to a multi server setting using replication and erasure coding, respectively. In particular, erasure coding (e.g., Reed- Solomon codes) has a lower storage overhead than replication under the same fault tolerance level.

Field measurements show that large-scale storage systems commonly experience disk/sector failures, some of which can result in permanent data loss. For example, the annualized replacement rate (ARR) for disks in production storage systems is around 2-4 percent. Data loss events are also found in commercial cloud-storage services. With the exponential growth of archival data, a small failure rate can imply significant data loss in archival storage. This motivates us to explore high performance recovery so as to reduce the window of vulnerability. Regenerating codes have recently been proposed to minimize repair traffic (i.e., the amount of data being read from surviving servers). In essence, they achieve this by not reading and reconstructing the whole file during repair as in traditional erasure codes, but instead reading a set of chunks smaller than the original file from other surviving servers and reconstructing only the lost (or corrupted) data chunks. An open question is, can we enable integrity checks atop regenerating codes, while preserving the repair traffic saving over traditional erasure codes? A related approach is HAIL, which applies integrity protection for erasure codes. It constructs protection data on a per-file basis and distributes the protection data across different servers. To repair any lost data during a server failure, one needs to access the whole file, and this violates the design of regenerating codes. Thus, we need a different design of integrity protection tailored for regenerating codes.

In this paper, we design and implement a practical data integrity protection (DIP) scheme for regenerating-coding based cloud storage. We augment the implementation of functional minimum-storage regenerating (FMSR) codes and construct FMSR-DIP codes, which allow clients to remotely verify the integrity of random subsets of long-term archival data under a multiserver setting. FMSR-DIP codes preserve fault tolerance and repair traffic saving as in FMSR codes [15]. Also, we assume only a thin-cloud interface, meaning that servers only need to support standard read/write functionalities. This adds to the portability of FMSRDIP codes and allows simple deployment in general types of storage services. By combining integrity

checking and efficient recovery, FMSR-DIP codes provide a low-cost solution for maintaining data availability in cloud storage.

In summary, we make the following contributions:

- We design FMSR-DIP codes, which enable integrity protection, fault tolerance, and efficient recovery for cloud storage.
- We export several tunable parameters from FMSRDIP codes, such that clients can make a trade-off between performance and security.
- We conduct mathematical analysis on the security of FMSR-DIP codes for different parameter choices.
- We implement FMSR-DIP codes, and evaluate their overhead over the existing FMSR codes through extensive test bed experiments in a cloud-storage environment. We evaluate the running times of different basic operations, including Upload, Check, Download, and Repair, for different parameter choices.

Related Work

We briefly summarize the most recent and closely related work here. Further literature review can be found in Section 1 of the supplementary file, available online. We consider the problem of checking the integrity of static data, which is typical in long-term archival storage systems. This problem is first considered under a single server scenario by Juels and Kaliski and Ateniese ET all giving rise to the similar notions POR and PDP, respectively. A major limitation of the above schemes is that they are designed for a single-server setting. If the server is fully controlled by an adversary, then the above schemes can only provide detection of corrupted data, but cannot recover the original data. This leads to the design of efficient data checking schemes in a multi server setting. By striping redundant data across multiple servers, the original files can still be recovered from a subset of servers even if some servers are down or compromised. Efficient data integrity checking has been proposed for different redundancy schemes, such as replication, erasure coding, and regenerating coding.

Specifically, although Chen et al. also consider regenerating-coded storage, there are key differences with our work. First, their design extends the single-server compact POR scheme by Shacham and Waters. However, such direct adaptation inherits some shortcomings of the single-server scheme such as a large storage overhead, as the amount of data stored increases with a more flexible checking granularity in the scheme of. Second, the storage scheme of [6] assumes that storage servers have encoding capabilities for generating encoded data, while we consider a thin-cloud setting, where servers only need to support standard read/write functionalities for portability and simplicity. The most closely related work to ours is HAIL, which stores data via erasure coding. As stated in Section 1, HAIL operates on a per-file basis and it is nontrivial to directly apply HAIL to regenerating codes. In addition, our work focuses more on the practical issues, such as how different parameters can be adjusted for the performance-security trade-off in practical deployment.

Existing System

- The problem of checking the integrity of static data, which is typical in long-term archival storage system, is considered. This problem is first considered under a single-server scenario by Juels and Kaliski and Ateniese et al, giving rise to the similar notions POR and PDP, respectively.
- The existing system design extends the single-server compact POR scheme for regenerating-coded storage.
- HAIL system, which stores data via erasure coding. HAIL operates on a per-file basis and it is nontrivial to directly apply HAIL to regenerating codes.

Disadvantages of Existing System

- A major limitation in the existing system is that they are designed for a single-server setting.
- Efficient data integrity checking has been proposed for different redundancy schemes, such as replication erasure coding and regenerating coding.
- A large storage overhead, as the amount of data stored increases with a more flexible checking granularity
- The storage servers have encoding capabilities for generating encoded data.

Proposed System

- The aim of our proposed system is to protect outsourced data in cloud storage against corruptions, adding fault tolerance to cloud storage, along with efficient data integrity checking and recovery procedures, becomes critical.
- Regenerating codes provide fault tolerance by striping data across multiple servers, while using less repair traffic than traditional erasure codes during failure recovery. We design and implement a practical data integrity protection (DIP) scheme for a specific regenerating code, while preserving its intrinsic properties of fault tolerance and repair-traffic saving.
- Our DIP scheme is designed under a mobile Byzantine adversarial model, and enables a client to feasibly verify the integrity of random subsets of outsourced data against general or malicious corruptions. It works under the simple assumption of thin-cloud storage and allows different parameters to be fine-tuned for a performance-security trade-off.
- We further analyze the security strengths of our DIP scheme via mathematical models. We demonstrate that remote integrity checking can be feasibly integrated into regenerating codes in practical deployment.

Advantages of Proposed System

- By striping redundant data across multiple servers, the original files can still be recovered from a subset of servers even if some servers are down or compromised.
- A thin-cloud setting is used where servers only need to support standard read/write functionalities for portability and simplicity.
- Different parameters can be adjusted for the performance-security trade-off.

Implementation

Storage Data in Thin-cloud

In this module, first we develop a REST-full interface, which include the commands PUT and GET. PUT allows writing to a file as a whole (no partial updates), and GET allows reading from a selected range of bytes of a file via a range GET request. Our DIP scheme uses only the PUT and GET commands to interact with each server. Our thin-cloud setting enables our DIP scheme to be portable to general types of storage devices or services, since no implementation changes are required on the storage backend. It differs from other "thick-"cloud-storage services where servers have computational capabilities and are capable of aggregating the proofs of multiple checks. There should not be any limits on the number of possible challenges that the client can make, since files can be kept for long-term archival. Also, the challenge size should be adjustable with different parameter choices, and this is useful when we want to lower the detection rate when the stored data grow less important over time.

Upload Data File and Metadata File

To reduce the key management overhead, we can derive multiple keys from a single secret using key derivation functions, as detailed in prior studies and standards. In addition, to relieve the local storage burden, we can encrypt all file keys with a master key, and outsource the storage of the encrypted keys to the cloud. Since the files in the cloud are typically of large size, we expect that the secret keys only incur a small constant overhead. We also append the MACs of all chunks to the metadata. Finally, the metadata is encrypted with ENC and replicated to each server to contribute only a small storage overhead.

Download and Decode the Needed Chunks based on FMSR-DIP

The PRFs off the FMSR-DIP code chunks to form the FMSR code chunks, which are then passed to NC Cloud for decoding if they are not corrupted. However, if we have a corrupted code chunk, then we can fix it with one of the following criteria.

- Download its AECC parities and apply error correction. Then we verify the corrected chunk with its MAC again.
- Download the code chunks from another server.

- A last resort is to download the code chunks from all n servers. We check all rows of the chunks including their AECC parities. The rows with a subset of the bytes marked correct can be recovered with FMSR codes; the rows with all bytes marked corrupted are treated as erasures and will be corrected with AECC.
- In particular, if there is only one failed server, then instead of trying to download K(n-k) chunks from any k servers, we download one chunk from all remaining (n – 1) servers as in FMSR codes

Row Verification for Downloaded Chunk file

Our probabilistic row verification in the Check operation. Note that there is a trade-off of choosing how many bytes to corrupt. A higher corruption rate means that the adversary can corrupt more bytes in a stripe, but the corruption is also easier to be detected by our row verification. Our objective is to provide a mathematical framework that analyzes the security strength of FMSR-DIP codes for different parameter choices.

System Study

Feasibility Study

The feasibility of the project is analyzed in this phase and business proposal is put forth with a very general plan for the project and some cost estimates. During system analysis the feasibility study of the proposed system is to be carried out. This is to ensure that the proposed system is not a burden to the company. For feasibility analysis, some understanding of the major requirements for the system is essential.

Three key considerations involved in the feasibility analysis are

- Economical feasibility
- Technical feasibility
- Social feasibility

Economical Feasibility

This study is carried out to check the economic impact that the system will have on the organization. The amount of fund that the company can pour into the research and development of the system is limited. The expenditures must be justified. Thus the developed system as well within the budget and this was achieved because most of the technologies used are freely available. Only the customized products had to be purchased.

Technical Feasibility

This study is carried out to check the technical feasibility, that is, the technical requirements of the system. Any system developed must not have a high demand on the available technical resources. This will lead to high demands on the available technical resources. This will lead to high demands being placed on the client. The developed system

must have a modest requirement, as only minimal or null changes are required for implementing this system.

Social Feasibility

The aspect of study is to check the level of acceptance of the system by the user. This includes the process of training the user to use the system efficiently. The user must not feel threatened by the system, instead must accept it as a necessity. The level of acceptance by the users solely depends on the methods that are employed to educate the user about the system and to make him familiar with it. His level of confidence must be raised so that he is also able to make some constructive criticism, which is welcomed, as he is the final user of the system.

Conclusion and Future Work

Given the popularity of outsourcing archival storage to the cloud, it is desirable to enable clients to verify the integrity of their data in the cloud. We design and implement a DIP scheme for the FMSR codes under a multiserver setting. We construct FMSR-DIP codes, which preserve the fault tolerance and repair traffic saving properties of FMSR codes. To understand the practicality of FMSRDIP codes, we analyze the security strength via mathematical modeling and evaluate the running time overhead via test bed experiments. We show how FMSR-DIP codes trade between performance and security under different parameter settings.

References

1. H. Abu-Libdeh, L. Princehouse, and H. Weatherspoon, "RACS: A Case for Cloud Storage Diversity," Proc. First ACM Symp. Cloud Computing (SoCC '10), 2010.
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. 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.
5. K. Bowers, A. Juels, and A. Oprea, "Proofs of Retrievability: Theory and Implementation," Proc. ACM Workshop Cloud Computing Security (CCSW '09), 2009.
6. B. Chen, R. Curtmola, G. Ateniese, and R. Burns, "Remote Data Checking for Network Coding-Based Distributed Storage Systems," Proc. ACM Workshop Cloud Computing Security (CCSW '10), 2010.
7. H.C.H. Chen and P.P.C. Lee, "Enabling Data Integrity Protection in Regenerating-Coding-Based Cloud Storage," Proc. IEEE 31st Symp. Reliable Distributed Systems (SRDS '12), 2012.

8. L. Chen, "NIST Special Publication 800-108," Recommendation for Key Derivation Using Pseudorandom Functions (Revised), http://csrc.nist.gov/publications/nistpubs/800-108/sp800-108.pdf, Oct. 2009.
9. R. Curtmola, O. Khan, and R. Burns, "Robust Remote Data Checking," Proc. ACM Fourth Int'l Workshop Storage Security and Survivability (StorageSS '08), 2008.
10. 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.
11. A. Dimakis, P. Godfrey, Y. Wu, M. Wainwright, and K. Ramchandran, "Network Coding for Distributed Storage Systems," IEEE Trans. Information Theory, vol. 56, no. 9, 4539-4551, Sept. 2010.
12. D. Ford, F. Labelle, F.I. Popovici, M. Stokel, V.-A. Truong, L. Barroso, C. Grimes, and S. Quinlan, "Availability in Globally Distributed Storage Systems," Proc. Ninth USENIX Symp. Operating Systems Design and Implementation (OSDI '10), Oct. 2010.
13. O. Goldreich, Foundations of Cryptography: Basic Tools. Cambridge Univ. Press, 2001.
14. O. Goldreich, Foundations of Cryptography: Basic Applications. Cambridge Univ. Press, 2004.
15. 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.