DISTRIBUTED DE-DUPLICATION SYSTEM: MORE SECURE AND RELIABLE APPROACH

Sagar G. Khengat,
Department Of Computer Engineering,
ISB&M School Of Technology,
Pune, India.

Swapnil S. belorkar,
Department Of Computer Engineering,
ISB&M School Of Technology,
Pune, India.

Alok Y. Shukla,
Department Of Computer Engineering,
ISB&M School Of Technology,
Pune, India.

Prof. Shital D. Bachpalle,
Assistant Professor,
Department Of Computer Engineering,
ISB&M School Of Technology,
Pune, India.

Abstract: Data de-duplication is a technique for eliminating duplicate copies of data, and has been widely used in cloud storage to reduce storage space and upload bandwidth. However, there is only one copy for each file stored in cloud even if such a file is owned by a huge number of users. As a result, de-duplication system improves storage utilization while reducing reliability. Furthermore, the challenge of privacy for sensitive data also arises when they are outsourced by users to cloud. Aiming to address the abovesecurity challenges, this paper makes the first attempt to formalize the notion of distributed reliable de-duplication system. We propose new distributed de-duplication systems with higher reliability in which the data chunks are distributed across multiple cloud servers. The security requirements of data confidentiality and tag consistency are also achieved by introducing a deterministic secret sharing scheme in distributed storage systems, instead of using convergent encryption as in previous de-duplication systems. Security analysis demonstrates that our de-duplication systems are secure in terms of the definitions specified in the proposed security model. As a proof of concept, we implement the proposed systems and demonstrate that the incurred overhead is very limited in realistic environments.

Keywords: De-duplication, distributed storage system, reliability, secret sharing

I. INTRODUCTION

In this paper, we show how to design secure de-duplication systems with higher reliability in cloud computing. We introduce the distributed cloud storage servers into de-duplication systems to provide better fault tolerance. To further protect data confidentiality, the secret sharing technique is utilized, which is also compatible with the distributed storage systems. In more details, a file is first split and encoded into fragments by using the technique of secret sharing, instead of encryption mechanisms.These shares will be distributed across multiple independent storage servers. Furthermore, to supportde-duplication, a short cryptographic hash value of the content will also be computed and sent to each storage server as the fingerprint of the fragment stored at each server. Only the data owner who first uploads the data is required to compute and distribute such secret shares, while all following users who own the same data copy do not need to compute and store these shares any more. To recover data copies, users must access a minimum number of storage servers through authentication and obtain the secret shares to reconstruct the data. In other words, the secret shares of data will only be accessible by the authorized users who own the corresponding data copy.

A. Brief Description

With the explosive growth of digital data, de-duplication techniques are widely employed to backup data and minimize network and storage overhead by detecting and eliminating redundancy among data. Instead of keeping multiple data copies with the same content, de-duplication eliminates redundant data by keeping only one physical copy and referring other redundant data to that copy. De-duplication has received much attention from both academia and industry because it can greatly improves storage utilization and save storage space, especially for the applications with high de-duplication ratio such as archival storage systems.

There are two types of de-duplication in terms of the size:

a) file-level de-duplication, which discovers redundancies between different files and removes these redundancies to reduce capacity demands, and

b) block-level de-duplication, which discovers and removes redundancies between data blocks. The file can be divided into smaller fixed-size or variable-size blocks. Using fixed size blocks simplifies the computations of block boundaries, while using variable-size blocks (e.g., based on Rabin fingerprinting) provides better de-duplication efficiency.

Though de-duplication technique can save the storage space for the cloud storage service providers, it reduces the reliability of the system. Data reliability is actually a very critical issue in a de-duplication storage system because there is only one copy for each file stored in the server shared by all the owners. If such a shared file/chunk was lost, a

disproportionately large amount of data becomes inaccessible because of the unavailability of all the files that share this file/chunk. If the value of a chunk were measured in terms of the amount of file data that would be lost in case of losing a single chunk, then the amount of user data lost when a chunk in the storage system is corrupted grows with the number of the commonality of the chunk. Thus, how to guarantee high data reliability in de-duplication system is a critical problem. Most of the previous de-duplication systems have only been considered in a single-server setting. However, as lots of de-duplication systems and cloud storage systems are intended by users and applications for higher reliability, especially in archival storage systems where data are critical and should be preserved over long time periods. This requires that the de-duplication storage systems provide reliability comparable to other high-available systems.

Furthermore, the challenge for data privacy also arises as more and more sensitive data are being outsourced by users to cloud. Encryption mechanisms have usually been utilized to protect the confidentiality before outsourcing data into cloud. Most commercial storage service providers are reluctant to apply encryption over the data because it makes de-duplication impossible. The reason is that the traditional encryption mechanisms, including public key encryption and symmetric key encryption, require different users to encrypt their data with their own keys. As a result, identical data copies of different users will lead to different ciphertexts. To solve the problems of confidentiality and de-duplication, the notion of convergent encryption has been proposed and widely adopted to enforce data confidentiality while realizing de-duplication. However, these systems achieved confidentiality of outsourced data at the cost of decreased error resilience. Therefore, how to protect both confidentiality and reliability while achieving de-duplication in a cloud storage system is still a challenge.intended for environments where shoulder-surfing is a serious threat.

II. THE METHOD OF PROGRESS AND ALGORITHM

A. *Data De-Duplication*

Data de-duplication involves finding and removing duplication within data without compromising its fidelity or integrity. The goal is to store more data in less space by segmenting files into small variable-sized chunks (32–128 KB), identifying duplicate chunks, and maintaining a single copy of each chunk. Redundant copies of the chunk are replaced by a reference to the single copy. The chunks are compressed and then organized into special container files in the System Volume Information folder. After de-duplication, files are no longer stored as independent streams of data, and they are replaced with stubs that point to data blocks that are stored within a common chunk store. Because these files share blocks, those blocks are only stored once, which reduces the disk space needed to store all files. During file access, the correct blocks are transparently assembled to serve the data without calling the application or the user having any knowledge of the on-disk transformation to the file. This enables administrators to apply de-duplication to files without having to worry about any change in behaviour to the applications or impact to users who are accessing those files.

After a volume is enabled for de-duplication and the data is optimized, the volume contains the following:

- **Unoptimized files**. For example, unoptimized files could include files that do not meet the selected file-age policy setting, system state files, alternate data streams, encrypted files, files with extended attributes, files smaller than 32 KB, other reparse point files, or files in use by other applications (the "in use" limit is removed in Windows Server 2012 R2).
- **Optimized files**. Files that are stored as reparse points that contain pointers to a map of the respective chunks in the chunk store that are needed to restore the file when it is requested.
- **Chunk store**. Location for the optimized file data.
- **Additional free space**. The optimized files and chunk store occupy much less space than they did prior to optimization.

B. *Distributed de-duplication systems*

The distributed systems' proposed aim is to reliably store data in the cloud while achieving confidentiality and integrity. Its main goal is to enable and distributed storage of the data across multiple storage servers. Instead of encrypting the data to keep the confidentiality of the data, our new constructions utilize the secret splitting technique to split data into shards. These shards will then be distributed across multiple storage servers.

C. *Building Blocks Secret Sharing Scheme.*

There are two algorithms in a secret sharing scheme, which are Share and Recover. The secret is divided and shared by using Share. With enough shares, the secret can be extracted and recovered with the algorithm of Recover. In our implementation, we will use the Ramp secret sharing scheme (RSSS) ,[8] to secretly split a secret into shards. Specifically, the (n, k, r)-RSSS (where $n > k > r \geq 0$) generates n shares from a secret so that (i) the secret can be recovered from any k or more shares, and (ii) no information about the secret can be deduced from any r or less shares. Two algorithms, Share and Recover, are defined in the (n, k, r)-RSSS.

• Share divides a secret S into $(k - r)$ pieces of equal size, generates r random pieces of the same size, and encodes the k pieces using a non-systematic k-of-n erasure code into n shares of the same size;

• Recover takes any k out of n shares as inputs and then outputs the original secret S. It is known that when $r = 0$, the $(n, k, 0)$-RSSS becomes the (n, k) Rabin's Information Dispersal Algorithm (IDA) [9]. When $r = k-1$, the $(n, k, k-1)$-RSSS becomes the (n,k) Shamir's Secret Sharing Scheme (SSSS).

D. *The File-level Distributed De-duplication System*

To support efficient duplicate check, tags for each file will be computed and are sent to S-CSPs. To prevent a collusion attack launched by the S-CSPs, the tags stored at different storage servers are computationally independent and different. We now elaborate on the details of the construction as follows.

System setup. In our construction, the number of storage servers S-CSPs is assumed to be n with identities denoted by $id1, id2, \cdots, idn$, respectively. Define the security parameter

as 1_ and initialize a secret sharing scheme SS = (Share, Recover), and a tag generation algorithm TagGen. The file storage system for the storage server is set to be. *File Upload.* To upload a file *F*, the user interacts with S-CSPs to perform the de-duplication. More precisely, the user firstly computes and sends the file tag ϕF= TagGen(*F*) to S-CSPs for the file duplicate check.

• If a duplicate is found, the user computes and sends $\phi F;idj$= TagGen′(*F, idj*) to the *j*-th server with identity *idj*via the secure channel for $1 \le j \le n$ (which could be implemented by a cryptographic hash function *Hj*(*F*) related with index *j*). The reason for introducing an index *j* is to prevent the server from getting the shares of other S-CSPs for the same file or block, which will be explained in detail in the security analysis. If $\phi F;idj$matches the metadata stored with ϕF, the user will be provided a pointer for the shard stored at server *idj*.

• Otherwise, if no duplicate is found, the user will proceed as follows. He runs the secret sharing algorithm SS over *F* to get *{cj}* = Share(*F*), where *cj*is the *j*-th shard of *F*. He also computes $\phi F;idj$= TagGen′(*F, idj*), which serves as the tag for the *j*th S-CSP. Finally, the user uploads the set of values $\{\phi F,cj, \phi F;idj\}$ to the S-CSP with identity *idj*via a secure channel. The S-CSP stores these values and returns a pointer back to the user for local storage.

File Download. To download a file *F*, the user first downloads the secret shares *{cj}* of the file from *k* out of *n* storage servers. Specifically, the user sends the pointer of *F* to *k* out of *n*S-CSPs. After gathering enough shares, the user reconstructs file *F* by using the algorithm of Recover(*{cj}*). This approach provides fault tolerance and allows the user to remain accessible even if *any* limited subsets of storage servers fail.

E. The Block-level Distributed System

In this section, we show how to achieve the fine-grained block-level distributed de-duplication. In a block-level de-duplication system, the user also needs to firstly perform the file-level de-duplication before uploading his file. If no duplicate is found, the user divides this file into blocks and performs block-level de-duplication. The system setup is the same as the file-level de-duplication system, except the block size parameter will be defined additionally. Next, we give the details of the algorithms of File Upload and File Download. *File Upload*. To upload a file *F*, the user first performs the file-level de-duplication by sending $\phi F$to the storage servers. If a duplicate is found, the user will perform the file-level de-duplication. Otherwise, if no duplicate is found, the user performs the block-level de-duplication as follows. He firstly divides *F* into a set of fragments *{Bi}* (where $i = 1, 2, \cdots$). For each fragment *Bi*, the user will perform a block-level duplicate check by computing ϕBi= TagGen(*Bi*), where the data processing and duplicate check of block-level de-duplication is the same as that of file-level de-duplication if the file *F* is replaced with block *Bi*. Upon receiving block tags $\{\phi Bi\}$, the server with identity *idj*computes a block signal vector $\sigma Bi$for each *i*

i) If σBi=1, the user further computes and sends $\phi Bi;j$= TagGen′(*Bi, j*) to the S-CSP with identity *idj*. If it also matches the corresponding tag stored, S-CSP returns a block pointer of *Bi* to the user. Then, the user keeps the block pointer of *Bi* and does not need to upload *Bi*.

ii) If σBi=0, the user runs the secret sharing algorithm SS over *Bi* and gets *{cij}* = Share(*Bi*), where *cij*is the *j*-th secret share of *Bi*. The user also computes $\phi Bi;j$for $1 \le j \le n$ and uploads the set of values $\{\phi F, \phi F;idj, cij, \phi Bi;j\}$ to the server *idj*via a secure channel. The S-CSP returns the corresponding pointers back to the user.

File Download. To download a file *F = {Bi}*, the user first downloads the secret shares *{cij}* of all the blocks *Bi* in *F* from *k* out of *n*S-CSPs. Specifically, the user sends all the pointers for *Bi* to *k* out of *n* servers. After gathering all the shares, the user reconstructs all the fragments *Bi* using the algorithm of Recover(*{·}*) and gets the file *F = {Bi}*.

F. Algorithm-

Step 1- Cloud auditor server generate a random set challenge request
Step 2- Cloud storage auditor compute and generate integrity proof.
Step 3- cloud auditor server compute R from
Step 4- Cloud auditor server Verify signature of R output False then abort if not true.
Step 5-cloud auditor server Verify integrity of output true or false.

III. MATHEMATICAL MODEL

Let S be the Whole system which consists,
S= {I,P,O}
Where,
I-Input,
P- procedure,
O- Output.
I-{F,U}
F-Files set of {F1, F2,…., FN}
U- No of Users {U1, U2,……,UN}

A. Procedure(P):

P={POW, n ,POW_B, POW_F, □, i,j ,m , k}.
Where,

1. POW - proof of ownership.
2. n - No of servers.
3. POW_B –proof of ownership in blocks.
4. POW_F – proof of ownership in files
5. □ - tag.
6. i- Fragmentation.
7. j- No of server.
8. m-message
9. k- Key.

B. File Upload(FU):

Step 1: File level de-duplication

If a file duplicate is found, the user will run the PoW protocol POWF with each S-CSP to prove the file ownership.for the *j*-th server with identity *idj*, the user first computes

$\phi F;idj$= TagGen′(*F, idj*)

and runs the PoW proof algorithm with respect to ϕF, *idj*. If the proof is passed,the user will be provided a pointer for the piece of file stored at *j*-th S-CSP. Otherwise, if no duplicate is found, the user will proceed as follows:

First divides *F* into a set of fragments *{Bi}* (where $i = 1, 2, \cdots$). For each fragment *Bi*, the user will perform a block-level duplicate check.

Step 2: Block Level de-duplication
If there is a duplicate in S-CSP, the user runs PoWBon input:
$\phi Bi;j$= TagGen′(*Bi*, *idj*)
with the server toprove that he owns the block *Bi*. If it is passed, the server simply returns a block pointer of *Bi* to theuser. The user then keeps the block pointer of *Bi* and does not need to upload *Bi*.

C. *Proof of ownership(POW):*

Step 1: compute and send ϕ' to the verifier.
Step 2: present proof to the storage server that he owns *F* in an interactive way with respect to ϕ' The PoW is successful if the proof is correct
$\phi' = \phi(F)$

D. *File Download(FD)-*

To download a file *F*, the user firstdownloads the secret shares *{cij,mfj}* of the file from *k*out of *n* storage servers. Specifically, the user sends all the pointers for *F* to *k* out of *n* servers. After gatheringall the shares, the user reconstructs file *F*, *macF*by using the algorithm of Recover(*{·}*). Then, he verifies the correctness of these tags to check the integrity of
the file stored in S-CSPs.

E. *Output(O):*

User can upload, download, recover, share files on cloud server and provide data de-duplication and reliability.

IV. ADVANTAGES OF SYSTEM

1) File-level De-duplication
2) Block-level De-duplication
3) Improved reliability by distributed deduplication

V. SYSTEM ARCHITECTURE

In following figure we design a system which is useful for preventing data de-duplication with improved reliability.

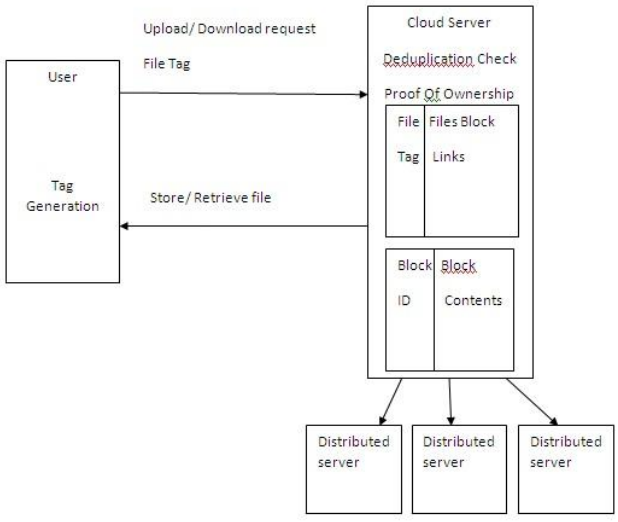

Figure1. Distributed De-duplication System

VI. CONCLUSION

We concentrated on the issue of evaluating if an untrusted server stores a customer's data. We presented a model for provable data possession (PDP), in which it is alluring to minimize the file piece gets to, the calculation on the server, and the client–server correspondence. Our answers for PDP fit this model: They cause a low (or even steady) overhead at the server and oblige a little, consistent measure of correspondence per challenge. Key parts of our plans are the backing for spot checking, which guarantees that the plans stay light weight, and the homomorphic verifiable labels, which permit to confirm data possession without having entry to the genuine data file. We likewise define the idea of hearty inspecting, which coordinates remote data checking (RDC) with for-ward mistake amending codes to moderate discretionarily little file debasements and propose a nonspecific change for adding vigor to any spot checking-based RDC plan. Examinations demonstrate that our plans make it down to earth to check possession of vast data sets. Past plans that don't permit testing are not commonsense when PDP is utilized to demonstrate possession of a lot of data, as they force a significant I/O and computational weigh on the server.

VII. REFERENCES

[1] J. R. Douceur, A. Adya, W. J. Bolosky, D. Simon, and M. Theimer, "Reclaiming space from duplicate files in a serverless distributed file system." in ICDCS, 2002, pp. 617–624.

[2] M. Bellare, S. Keelveedhi, and T. Ristenpart, "Dupless: Serveraided encryption for de-duplicated storage," in USENIX Security Symposium, 2013.

[3] ——, "Message-locked encryption and secure deduplication," in EUROCRYPT, 2013, pp. 296–312.

[4] S. Halevi, D. Harnik, B. Pinkas, and A. Shulman-Peleg, "Proofs of ownership in remote storage systems." in ACM Conference on Computer and Communications Security, Y. Chen, G. Danezis, andV. Shmatikov, Eds. ACM, 2011, pp. 491–500.

[5] J. S. Plank and L. Xu, "Optimizing Cauchy Reed-solomon Codes for fault-tolerant network storage applications," in NCA-06: 5th IEEE International Symposium on Network Computing Applications, Cambridge, MA, July 2006.

[6] P. Anderson and L. Zhang, "Fast and secure laptop backups with encrypted de-duplication," in Proc. of USENIX LISA, 2010

[7] J. Stanek, A. Sorniotti, E. Androulaki, and L. Kencl, "A secure data de-duplication scheme for cloud storage," in Technical Report, 2013.

[8] G. Ateniese, R. Burns, R. Curtmola, J. Herring, L. Kissner, Z. Peterson, and D. Song, "Provable data possession at untrusted stores," in Proceedings of the 14th ACM conference on Computer and communications security, ser. CCS '07. New York, NY, USA: ACM, 2007, pp. 598–609.

[Online].Available:http://doi.acm.org/10.1145/1315245.1315318

[9] R. D. Pietro and A. Sorniotti, "Boosting efficiency and security in proof of ownership for de-duplication." in ACM Symposium on Information, Computer and Communications Security, H. Y. Youm and Y. Won, Eds. ACM, 2012, pp. 81–82.

[10] W. K. Ng, Y. Wen, and H. Zhu, "Private data de-duplication protocols in cloud storage." in Proceedings of the 27th Annual ACM Symposium on Applied Computing, S. Ossowski and P. Lecca, Eds. ACM, 2012, pp. 441–446.

