Volume 5, Issue 7, July 2015 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

MHT Based Reliable and Fault Tolerant Cloud Storage System for a Stateful Data Upload and Retrieving Model to Correspond Efficient Data Consumption

Amit Kumar*
Student,
Department of Computer Engineering GHRCEM,
Savitribai Phule Pune University, Pune, India

S S Sambare
Associate Professor,
Department of Computer Engineering PCCOE,
Savitribai Phule Pune University, Pune, India

Abstract— cloud computing model has become immensely popular with its breakthrough advantages over other computational models. The massive computational power offered by cloud provides the ability of data storage on cloud servers. These being an advantage for cloud user, also raises concerns for data security. Users have to deeply rely on the existing offsite storage architecture for the security of data. The foremost constraints of data auditing are legitimate access of data, availability of data and integrity of stored data. Third party auditing is not reliable enough for the users and hence we propose a storage integrity and availability algorithm which can be implemented over the existing RAID model for better auditing and recovery of data. Our algorithm employs integration of Merkle Hash Tree and offers a supplementary advantage of resumable file retrieval, which can be beneficial for resource deficient terminals.

Keywords— cloud computing, cloud storage, merkle hash tree, RAID servers, data integrity

I. INTRODUCTION

Cloud computing with its numerous advantages has been well received within the IT community. However several challenges needs to be addressed for achieving user confidence and making the data auditing process more transparent. A major drawback being that users have no control on the integrity and availability of data once its uploaded over cloud for the fact that no copy is retained on the local machines. Cloud service providers may unknowingly corrupt or delete data for cost cutting or in the process of removal of stale data. The clean-up services can remove rarely accessed data from the cloud without the consent of users. With progressing research in the domain of secure data storage over cloud our scheme handles the data auditing more efficiently by making use of hashing, which ensures even the slightest corruption is detected and resolved with the least overhead. In our experimentation results, we have been able to detect smallest corruption on data and reduce the processing time of data preparation and recovery by avoiding parsing of undesirable data retrieval. The mechanism not only offers the assurance of a secure storage but also delivers error localization or the misbehaving server identification simultaneously. We also understand that cloud should be beneficial for all business domains and hence we have addressed the network bottlenecks which can be beneficial for smaller businesses operating behind slower connections. The proposed system integrates stateful data upload and retrieval model which can be valuable under network failure.

II. RELATED WORK

Being a major concern area, several models have been proposed in the past for ensuring data availability and integrity. One such model is "proof of retrievability" (POR) which ensures remote data integrity using spot-checking and error correcting code. The model proposed by Juels et al [1] and Dodis et al [3] employs Error Correcting Code (ECC) and Message Authentication Code (MAC). The model was further reworked and the optimized algorithm was presented by Shacham and Waters [2]. The algorithm suggests that validation is processed by generating authenticators which are stored on server. The users requests for these authenticators on multiple files and expects a valid response. However, these models worked only on static data structures. These limitation has been overcome by Erway et al [5] which employed dynamic data possession. Another breakthrough approach has been implemented by Cong Wang, Qian Wang, Kui Ren, Ning Cao and Wenjing Lou [4] which efficiently handles error localization. However data recovery process is an overhead and may be inefficient for smaller corruptions. Our proposed algorithm has considerably less overhead for error localization and data recovery in misbehaving servers as only the affected chunks can be extracted specifically using the hash key stored on the user side. Since we eliminated contiguous data structure, hence the file recovery time for the misbehaving servers has gradually decreased.

III. ARCHITECTURE

The generic cloud architecture consists of three layers, mainly the cloud server, user and a third party auditor [1-2]. The lack of trust arises for the fact that data integrity and availability checks are handled by the third party auditors,

which is an external agent for the user. Therefore cloud users can be more confident if the auditing process is further relative and connected to the data owners. Hence, the architecture can be redesigned by eliminating third party auditors as shown in Figure 1. Now, the users have the challenge key which can be targeted to cloud server for data accuracy checks. For every challenge c, the user will expect a response r. In-case of any corruption, the data will be recovered from other RAID diskettes.

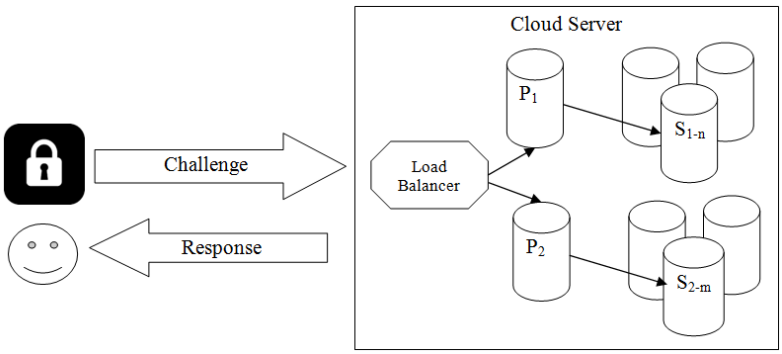

Fig 1: System Architecture

The system architecture will host m+n RAID diskettes across the cloud server. The user will hold the challenge key which will be generated at the time of file upload.

IV. IMPLEMENTATION

Our proposed system has three major components: data preparation and storage, file retrieval, data auditing. When a file is uploaded over the server, then our algorithm splits the file into chunks and organizes them in a tree structure. File hash is calculated for all the chunks which are also stored in the same order as that of chunks. Data redundancy is prevented for similar hashes which efficiently handles the memory occupancy. The chunk hashes are paired and for every pair, the hashes are concatenated to get a new hash which is further hashed. Moving a level up the tree, the concatenation and hashing is repeated till a 512 bit single hash value is generated for the entire file.

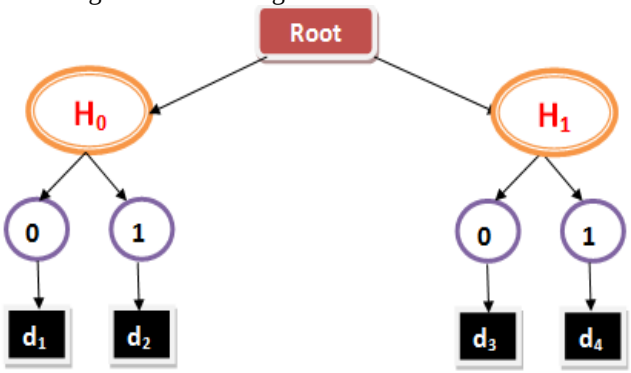

Fig 2: Data is split into chunks and structured in tree

The hash is stored in the server and a copy is provided to the user which has been represented in our system as follows:

```
<?xml version="1.0" encoding="utf-8" standalone="yes"?>
<FilesAuthorizationKey>
  <Files>
    <Position>1,0</ Position >
    <ParentFileEntityHash>99d7ebbaafffc236c0781eb66c6b5898a3a4cd03</ParentFileEntityHash>
    <NodeFileEntityName>test.mkv</NodeFileEntityName>
    <NodeFileEntityHash>950e0a0dd1365dea5c3ed0b94829d4136f995bc2</NodeFileEntityHash>
  </Files>
</FilesAuthorizationKey>
```

This approach smoothly works for data updating process without any overhead. Henceforth, the user has the privilege to perform data auditing to ensure data availability and data integrity. We have been able to successfully detect the smallest possible corruption on encrypted data chunks. When a challenge is processed by the user, the auditing process starts with the mapping of user provided file hash and chunk hashes with that of server stored hashes in the database. Challenge verification at node level makes error localization easy and resourceful. Once this verification passes, then the algorithm rapidly generates hashes and contests with the challenged hashes.

$ChallengeHash_i^{(j)} * W^{(j)}i \leftarrow (f\alpha_j(f_i), i \in \{1, \ldots, n\}) / w^{(j)}i,$

$UserHash_i ?= n(ChallengeHash_i^{(j)} * W^{(j)}i)$

$Hashi(j) = \sum \alpha ij * W(j)$

where: i = data block, j = server ID, α = precomputed data hash, W = database entity ID

In case of any corruption or unavailable data, a disparity is logged in the audit and the recovery process is triggered. One of the major advancement of our implementation is the speedy recovery process which runs in the background via worker threads. The thread only recovers the corrupted or unavailable chunks from the RAID servers as pictured in figure 3.

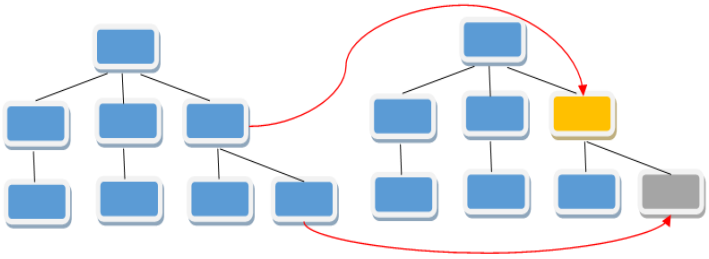

Fig 3: Minimal Data Traversing is required for recovery

We have also provided stateful file retrieval and upload model which can be beneficial for resource deficient terminals with slower connections. In the earlier implementations, users have to re-upload entire file in-case of power or network failure. But with the stateful file upload and retrieval model, users can resume the process as per the below pseudo code:

Step 1. User requests for file.
Step 2. Hash mapping is done and file is prepared for response
Step 3. Download Response

```
Check if user has the file stream available
        If available check if the hashes match
                if match, check if download is necessary
                else download the remaining bytes
        If not match then the file has been updated
                Delete old data and respond the new data
If no stream available then respond with the data
```

V. RESULTS

The proposed system is intended to deliver faster corruption detection and rapid data recovery with least data operations. These constraints will provide a consistent storage model with guaranteed data availability and security. When compared with the existing contiguous structured algorithm using our multimedia test data, we successfully established the prominent efficiency of our algorithm in terms of execution time and memory management. The execution time of individual modules of our algorithm is lesser then that of the existing algorithm [Figure 4].

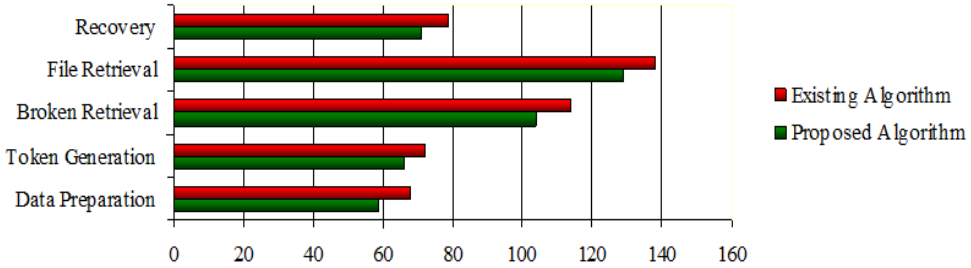

Fig 4: Time Comparison (in milliseconds)

The selection of Merkel hash tree for our model has been primarily due to its data rendering capabilities on large datasets. The hashes are obtained at node level which helps in swift error detection. In case of server failures or data corruption, unlike existing models which recovers entire file, only the effected nodes has to be recovered. Hence, in our test cases we were able to achieve rapid file recovery on every iteration [Figure 5].

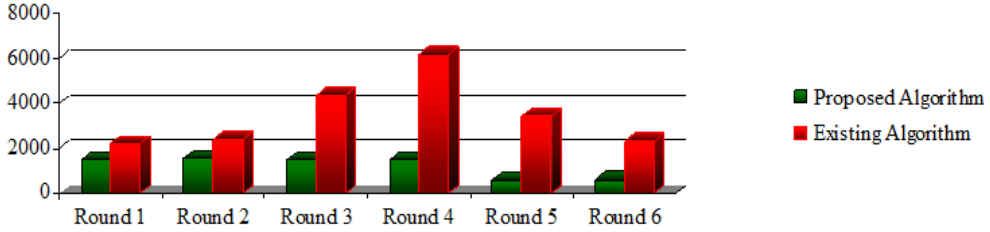

Fig 5: Data Retrieved for File Recovery (in bytes)

VI. CONCLUSIONS

Our algorithm deals with the critical parameters of data integrity and also provides a mechanism for fast error localization and error recovery. Usage of hash trees helps in error detection even at block levels and subsequently ropes data recovery with minimum overhead on the server. The proposed system provides a reliable cloud storage model by

ensuring high availability of data and corruption detection for every data request. We have also integrated algorithm for resumable file retrieval which can also be used while uploading data over cloud. This feature can be highly beneficial for resource deficient servers or in events of network failures.

REFERENCES

[1] Kevin D. Bowers, Ari Juels, and Alina Oprea, *Proofs of retrievability: theory and implementation*, In Proceedings of the 2009 ACM workshop on Cloud computing security, CCSW 2009, pages 43-54, New York,NY,USA,2009.ACM.

[2] Hovav Shacham and Brent Waters, *Compact proofs of retrievability*, In Proceedings of the 14th International Conference on the Theory and Application of Cryptology and Information Security: Advances in Cryptology, ASIACRYPT 2008, pages 90-107, Berlin, Heidelberg, Springer-Verlag

[3] Yevgeniy Dodis, Salil Vadhan, and Daniel Wichs, *Proofs of retrievability via hardness amplification*, In Proceedings of the 6th Theory of Cryptography Conference on Theory of Cryptography, TCC 2009, pages 109-127, Berlin, Heidelberg, 2009. Springer-Verlag.

[4] Cong Wang, Qian Wang, Kui Ren, Ning Cao and Wenjing Lou, *Towards secure and dependable storage services on Cloud computing*, IEEE Transactions on Services Computing, Vol. 5, No. 2, pp. 220-232, April-June 2012, 2012/07/12

[5] Chris Erway, Alptekin, Charalampos Papamanthou, and Roberto Tamassia, *Dynamic provable data possession*, in Proceedings of the 16th ACM conference on Computer and communications security, CCS 2009, pages 213-222, New York, USA, 2009 ACM.

