International Journal of Engineering and Advanced Technology (IJEAT)
ISSN: 2249-8958 (Online), Volume-8 Issue-6, August, 2019

# An Adaptive Methodology for Integrity Checking in Cloud Storage

**L Jambulingam, T V Ananthan, P S Rajakumar**

***Abstract*: Majority of the organization uses cloud for storage purpose in order to reduce the cost as well as maintenance. Due to increasing threat from internal and external sources, there would be possibility of corruption in the cloud storage files. Thus the storage must to be monitored periodically for integrity checking. Since most of the Data Owners have limited resources thus the responsibility of integrity checking goes to the Third Party Auditors (TPA). Usually the static way of deciding to use particular hash tree methodology to store the cloud storage meta-data, which is mainly used for integrity checking throughout is inappropriate for two main reasons, firstly, due to more fluctuated loss or corrupted cloud data, secondly, based on the variations in the number of files in the cloud users' directory; Initially the static approach would be good but it may not be optimal solution at the later period. therefore, in this paper, we have proposed Adaptive Integrity Checking method (AIC), which would lead a way for adaptive dynamic hash tree methodology for holding the cloud storage meta-data; which would drastically increases the performance of integrity checking in terms of both time and space complexity besides the benefits obtained in the EDHT-n version and HEDHT methodologies of handling the cloud storage integrity checking.***

***Keywords: Third Party Auditor (TPA), Adaptive Integrity Checking (AIC), Hybrid Enhanced Dynamic Hash Tree (HEDHT), Enhanced Dynamic Hash Tree (EDHT), Meta-Data, Cloud Service Provider (CSP), Microservice, API.***

## I. INTRODUCTION

Storage data is the asset to the client, to reduce cost; original file is being stored in cloud and deleted in the end-user. Cloud storage is a model of networked enterprise storage where data is stored in virtualized pools of storage which are generally hosted by third parties. Cloud storage provides customers with benefits, ranging from cost saving and simplified convenience to mobility opportunities and scalable service. These great features attract more and more customers to utilize and storage their data to the cloud storage: according to the analysis report, the volume of data in cloud is expected to achieve 40 trillion gigabytes in 2020. Even though cloud storage system has been widely adopted, it fails to accommodate some important emerging needs such as the abilities of auditing of cloud files by cloud clients. Thus integrity should be foremost requirement, any corruption had happened in CSP, it should be detected and corrected either during the file request given by the users or during auditing stage and made immediate recovery in order to maintain the integrity in the form of consistency across all of its data backup copies [14] by executing the recovery routines. Most importantly, the user may not aware about these processes are running in background, and thus Business Continuity Planning would be ensured always. Subsequently the trustworthiness towards the CSP would be improved. EDHT-N and HEDHT are the proposed Hash Tree Methodologies as advanced to MHT [2],[22] as shown in Figure 1 and Figure 2, where meta-data of the cloud file will be stored in static way, now we are proposed Adaptive Integrity Checking for the same purpose in a dynamic manner to speed up the audit and recovery process effectively compared to the earlier proposed paper.

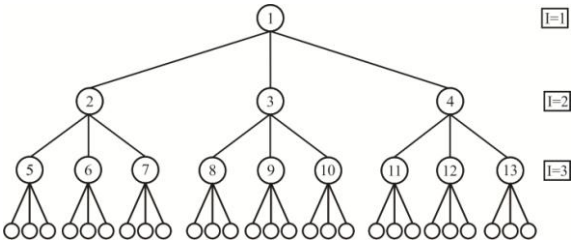



**Fig.1 Enhanced Dynamic Hash Tree – Version 3**

## II. PROPOSED TECHNIQUE

### Adaptive Integrity Checking (AIC)

Audit would be performed by TPA [5],[7],[10] on regular frequency with cloud storage to determine the faulty file in cloud, subsequently, TPA would record those error details in its database to ascertain the correct integrity checking methodology dynamically based on the Adaptive Integrity Checking (AIC) method and the same to be adopted. Initially, we say best methodology because it requires less number of hash computations for tree formation as well as recovery process compared to static EDHT-n/HEDHT methodologies.

Revised Manuscript Received on October 30, 2019.
* Correspondence Author
**L Jambulingam***, Research Scholar, Department of Computer Science and Engineering, Dr.M.G.R. Educational and Research Institute University, Chennai, India.
**T V Ananthan**, Professor, Department of Computer Science and Engineering, Dr.M.G.R. Educational and Research Institute University, Chennai, India.
**P S Rajakumar**, Professor, Department of Computer Science and Engineering, Dr.M.G.R. Educational and Research Institute University, Chennai, India.

TPA [12] would capture the error rates of each and every user directories available in cloud on a particular frequencies or whenever user access the files, hence based on these various error rate statistics details, we could pick the most occurring error rate

percentage or sometimes if the error rate percentage are not repeating then we should take an average of all those recorded value and finally decide which hash tree methodology to be adopted.

We also determine appropriate hash tree methodologies to store the cloud meta-data dynamically based on the variation in the number of files in the cloud users' directories. Thus it improves the performance of the auditing. AIC batch processing [11] would furthermore improve the speed of the auditing as well as recovery process.

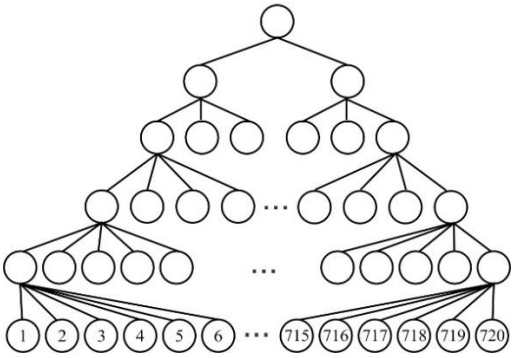



**Fig.2 Hybrid Enhanced Dynamic Hash Tree**

## III. ADAPTIVE INTEGRITY CHECKING (AIC) IN THIRD PARTY AUDITOR (TPA) USING API AND MICROSERVICE

Integrity Checking APIs are developed in a RESTful style as shown in the Table 1. These 10 APIs will have CRUD capabilities GET (get a single item or a collection), POST (add an item to a collection), PATCH (edit an item that already exists in a collection), DELETE (delete an item in a collection).

Table 1: APIs used for audit and error detection of corrupted cloud storage to ease the recovery process.

Also it were developed in a microservice -a software development technique - a variant of the service-oriented architecture (SOA) architectural style that structures an application as a collection of loosely coupled services, so each services are fine-grained and the protocols are lightweight.

As per Figure 3, User [3],[19],[21],[23] would request thru Cloud client for enquiry/delete/update operation of cloud file to the specific cloud service provider (CSP). API is a contract that provides guidance for a consumer (Cloud Client) to use the underlying service available in TPA, CSP, MDS, Recovery system[17]. API is usually a portion of a microservice, allowing for interaction with other microservices.

The broken out sections of the business logic, each encompassing a microservice. Thus complexity of the application is reduced, as the different services have well-defined interactions with each other.

Microservices architecture has become a common approach for any cloud integrity checking to achieve agility and the Continuous Delivery of applications to meet the growing demand of the cloud users.

Microservices are small, light, modular software programs, designed to fulfill one or a few purposes. They may be deployed independently, in small groups, in a container, or as part of a platform-as-a-service (PaaS) framework. They stand in contrast to more traditional monolithic programs, which are designed to fulfill several needs. Cloud Developers and users today are concerned with speedy development cycles, software scale, performance, and flexibility. Microservices offer all of these.

**Table.1 List of APIs for Cloud Integrity Checking**

| # | REST API | Provider | Consumer | Request Type |
|---|---|---|---|---|
| 1 | cloud-file | TPA | Cloud Client | PATCH; GET; DELETE |
| 2 | root-digest | CSP | TPA | GET |
| 3a | recover-file | Recovery System | TPA | GET |
| 3b | file-retrieve | CSP | TPA | GET |
| 4 | create-modified-file-digest | MDS | Cloud client | POST |
| 5 | file-push | CSP | Cloud client | POST |
| 6 | digest-push | TPA | Cloud client | POST |
| 7 | create-new-file-digest | MDS | Cloud client | POST |
| 8 | tpa-audit | CSP | TPA | GET |
| 9 | construct-edhtn construct-hedht | CSP | TPA | POST |

**Cloud-file:** User would request thru Cloud client [18],[21] for enquiry/delete/update/create operation of cloud file to the specific CSP thru TPA.

**Root-digest:** Only in case of enquiry and update operation, proof of ownership [1],[4],[6],[15],[16],[8],[9] will be determined, hence TPA would fetch the challenge corresponds to the user cloud file and send to CSP, which in turn send the root of EDHT-n/HEDHT for the received challenge. If root send by CSP matches with the root available in TPA then file reside in cloud is in intact.

**Recover-file**: Root of EDHT-n/HEDHT send by CSP does not matches with the root available in TPA then file reside in cloud is not intact. In case of file corrupt, invoke the recovery process.

**File-retrieve:** In case of file intact, invoke the actual file for view or modify. In case of modify, the file should be locked in serialiable mode to avoid dirty read, phaton read, non-repeatable read problems and sent to TPA.

**Create-modified-file-digest:** Fetched cloud file would be modified by the user/group users whose got privileges and subsequently sent to MDS.

**File-push:** In case of file update/create, the modified/created file will be pushed to CSP along with meta-data received from MDS to reconstruct the EDHT-n/HEDHT in case of existing, else it would create new EDHT-n/HEDHT for the newly created Cloud Directory file.

**Digest-push:** Cloud client send the meta-data received from MDS [24] to TPA to reconstruct EDHT-n/HEDHT in case of modified/created file or construct new EDHT-n/HEDHT in case of newly created cloud directory file. Thus CSP and TPA would always be kept in consistency the EDHT-n/HEDHT.

**Tpa-audit:** Audit would be performed by TPA on regular frequency with CSP to determine the faulty file in cloud, hence based on these values, TPA would record the errors in TPA database to ascertain the correct integrity checking methodology dynamically.

**Create-new-file-digest:** User can add new file to the cloud in particular directory by requesting the cloud client which subsequently request MDS to generate meta-data.

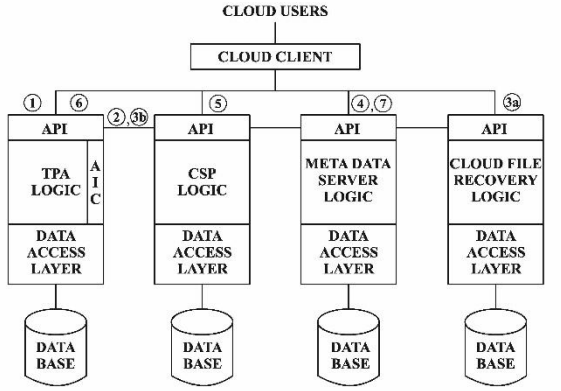



**Fig.3 Implementation of Cloud Integrity Checking using Microservice and API**

**Construct-edhtn or construct-hedht:** TPA send the same meta-data and methodology proposed by AICA to CSP directly instead of sending to cloud client, CSP uses these details to reconstruct the suggested EDHT-n/HEDHT.

**Aica-execute:** TPA would execute Adaptive Integrity Checking Algorithm (AICA) to arrive which optimal hash tree methodology to use for storing the cloud storage meta-data (either to use new or to retain the old methodology); by inputting the parameters obtained i.e., error percentage rate identified during TPA audit using tpa-audit API, which is noted as per Table 2 and variations in number of files in the cloud user directories identified during file-push to subsequently construct the appropriate suggested EDHT-n/HEDHT as per Table 3. Once the new hash tree methodologies is identified, it should be informed to CSP by invoking the construct-edhtn or construct-hedht API in order to synchronize between TPA and CSP of cloud storage meta-data to maintain consistency, which would ease the cloud storage auditing process efficiently as shown in Figure 4.

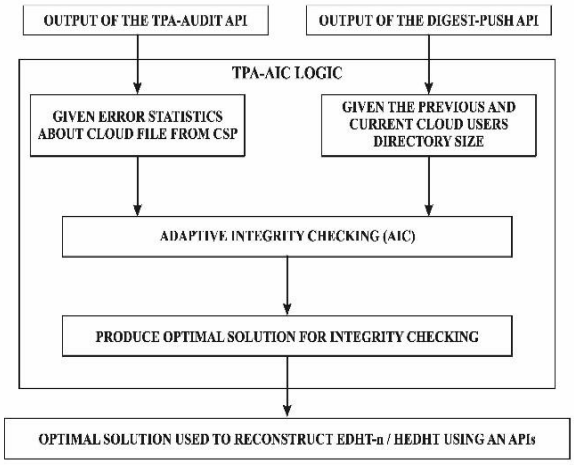



**Fig.4 Adaptive Integrity Checking Flow**

**Table.2 Record Error Details in TPA database for monthly frequency for Integrity Checking Purpose**

| Time Period | Error % for | | | | | | |
|---|---|---|---|---|---|---|---|
| | 50K Files | 200K Files | 300K Files | 700K Files | 1500K Files | 3000K Files | 7000K Files |
| January | 1.5 | 1 | 7.25 | 9 | 0 | 6 | 0.6 |
| February | 2.5 | 3 | 6.75 | 12 | 3 | 6 | 0.7 |
| March | 1.25 | 1 | 9 | 5 | 3 | 6 | 0.5 |
| April | 2.5 | 3 | 5 | 5 | 4 | 5 | 1.9 |
| May | 2.25 | 1.5 | 6 | 5 | 4.4 | 5 | 0.4 |
| June | 1 | 2.75 | 8 | 7 | 7 | 5 | 1.3 |
| July | 3 | 2 | 2 | 11 | 3.5 | 7 | 1.5 |
| August | 1 | 0.75 | 12 | 0 | 3.5 | 7 | 1.1 |
| September | 2.75 | 2.5 | 5 | 7 | 1.6 | 5 | 1 |
| October | 1.75 | 1.25 | 2 | 7.25 | 3 | 4 | 1 |
| November | 2.5 | 2.5 | 9 | 6.75 | 0 | 8 | 1 |
| December | 2 | 2.75 | 12 | 9 | 3 | 8 | 1 |

**Table.3 Best Hash Tree Methodology proposed to store meta-data in cloud storage for dynamic integrity checking of different Minimum-Maximum File Range combination in CSP end with varied Error rate %**

| Best Hash Tree Methodologies | Min-Max Range (NIL Error) | Best Hash Tree Methodologies | Min-Max Range (10% Error) | Best Hash Tree Methodologies | Min-Max Range (15% Error) |
|---|---|---|---|---|---|
| HEDHT | 1-120 | HEDHT | 1-120 | HEDHT | 1-120 |
| EDHT-5 | 121-125 | EDHT-5 | 121-125 | EDHT-5 | 121-125 |
| MHT | 126-128 | MHT | 126-128 | MHT | 126-128 |
| EDHT-6 | 129 216 | EDHT-6 | 129-216 | EDHT-6 | 129-216 |
| EDHT-4 | 217-256 | EDHT-4 | 217-256 | EDHT-3 | 217-243 |
| EDHT-7 | 257-343 | EDHT-7 | 257-343 | EDHT-4 | 244-256 |
| EDHT-5 | 344-625 | EDHT-5 | 344-625 | EDHT-7 | 257-343 |
| HEDHT | 626-720 | HEDHT | 626-720 | EDHT-5 | 344-625 |
| EDHT-3 | 721-729 | EDHT-3 | 721-729 | HEDHT | 626-720 |
| EDHT-4 | 730-1024 | EDHT-4 | 730-1024 | EDHT-3 | 721-729 |
| EDHT-6 | 1025-1296 | EDHT-6 | 1025-1296 | EDHT-4 | 730-1024 |
| EDHT-7 | 1297-2401 | EDHT-3 | 1297-2187 | EDHT-6 | 1025-1296 |
| EDHT-5 | 2402-3125 | EDHT-7 | 2188-2401 | EDHT-3 | 1297-2187 |
| EDHT-4 | 3126-4096 | EDHT-5 | 2402-3125 | EDHT-7 | 2188-2401 |
| HEDHT | 4097-5040 | EDHT-4 | 3126-4096 | EDHT-5 | 2402-3125 |
| EDHT-6 | 5041-7776 | HEDHT | 4097-5040 | EDHT-4 | 3126-4096 |
| MHT | 7777-8192 | EDHT-3 | 5041-6561 | HEDHT | 4097-5040 |
| EDHT-5 | 8193-15625 | EDHT-6 | 6562-7776 | EDHT-3 | 5041-6561 |
| EDHT-7 | 15626-16807 | MHT | 7777-8192 | EDHT-6 | 6562-7776 |
| EDHT-3 | 16808-19683 | EDHT-5 | 8193-11599 | MHT | 7777-8192 |
| HEDHT | 19684-40320 | EDHT-4 | 11600-16384 | EDHT-4 | 8193-16384 |

## IV. RESEARCH METHODOLOGY

After a series of attempts and summaries, we put forward our solution as follow:

**Table.4 Notations used in AIC Algorithm**

| | |
|---|---|
| find($£_{hashtre}$) | To find height of hash tree of particular methodology |
| $£_{hashtree}$ | Height of hash tree |
| $N_s$ | Number Of Siblings required in a particular methodology |
| $VN_S$ | Various Number of Siblings i.e., EDHT-3 is 3; EDHT-4 is 4 and so on |
| $N_f$ | Number of files |
| $N_{fc}$ | Number of files currently |
| $N_{htcc}$ | Number of hash tree Defect calculation to correct it |
| $N_{htc}$ | Number of hash tree computation for each methodology |
| $find_{htc}$ | Find hash tree defect correction |
| $d_p$ | Defect percentage |
| $min_r$ | Minimum range |
| $max_r$ | Maximum range |
| $f_{aic}$ | Final Adaptive Integrity Checking Method |
| $c_{aic}$ | Current Best Integrity Checking Method |
| $temp_r$ | Temporary range |
| $f_d$ | Defected files |
| $£_{hvs}$ | Height of hash tree for various number of siblings |
| $TA_{hcvs}$ | Total average hash tree calculation for various number of siblings |
| $TW_{hcvs}$ | Total Worst hash tree calculation for various number of siblings |
| $TB_{hcvs}$ | Total Best hash tree calculation for various number of siblings |
| $f_{mhcfc}$ | To find minimum hash tree calculation from the collection |
| $hc_{ph}$ | Hash calculation at particular height |
| $i_{nf}$ | Initial Number of files |

**Algorithm 1** find($£_{hashtree}$)

**Input:** $N_s$, $N_f$
**Output:** $£_{hashtree}$
$hc_{ph} \leftarrow 1$;
$£_{hashtree} \leftarrow 0$;
if$N_f$ ==1 then
return 2;
If ($N_s$> 0)
**while**$N_s^{£}{}_{hashtree} < N_f$
$£_{hashtree} \leftarrow £_{hashtree} + 1$
else
while(true)
$h_{cph} \leftarrow h_{cph} * £_{hashtree}$;
if$h_{cph}$>= $N_f$ then
break;
$£_{hashtree} \leftarrow £_{hashtree} + 1$;
end if
return $£_{hashtree}$ ;

Algorithm 1 is used to find out a height for Hybrid EDHT and EDHT-n versions, basically it takes the number of files as an input as well as the number of sibling to represent the version number only in case of EDHT-n methodology and return the height.

**Algorithm 2** $N_{htcc}$

**Input:** $N_s$, $£_{hashtree}$
**Output:** $N_{htcc}$
$N_{htcc} \leftarrow 0$
if ($N_s$> 0)
$N_{htcc} \leftarrow (£_{hashtree} - 1) * N_s$;
else
fori ←2; i<= $£_{hashtree}$; i++
$N_{htcc} \leftarrow N_{htcc} + i$;
end for
end if
return$N_{htcc}$;

Algorithm 2 is used to calculate the hash computation required for the hybrid EDHT and EDHT-n version to identify one defected file and thus it will be multiplied with number of defected files to compute total hash computation to find out all the faulty files.

**Algorithm 3** hashTreeComputation

**Input:** $N_s$, $£_{hashtree}$
**Output:** $N_{htc}$
$N_{htc}$ 0;
in$hc_{ph} \leftarrow 1$
If ($N_s$> 0)
**for** j ← 0; j< $£_{hashtree}$; j++
$N_{htc} \leftarrow N_{htc} + N_s^j$;
end for
Else
forint j ←1; j<= $£_{hashtree}$; j++
$hc_{ph} \leftarrow hc_{ph} * j$;
$N_{htc} \leftarrow N_{htc} + hc_{ph}$;
end for
**return**$N_{htc}$;

Algorithm 5 is used to compute total hash computation required for hybrid EDHT and EDHT-n version for the given height of it and number of siblings only in case of EDHT-n version.

**Algorithm 4** adaptiveIntegrityChecking

**Input:**$d_p$, $N_f$
**Output:** $f_{aic}$, $min_r$, $max_r$
for$N_{fc} \leftarrow i_{nf}$; $N_{fc}$<=$N_f$; $N_{fc}$ ++
$f_d \leftarrow N_{fc}$ /100 * $d_p$;
$£_{hvs} \leftarrow$ find($£_{hashtree}$)($VN_S$, $N_{fc}$);
**Best**
$TB_{hcvs} \leftarrow 1 + find_{htc}(VN_S, £_{hvs}) *$
($f_d/VN_S$ + ($f_d$mod $VN_S$ ==0 ? 0:1)) +
$N_{htc}(VN_S, £_{hvs})$;

**Worst**
$TW_{hcvs} \leftarrow 1 + find_{htc}(VN_S, £_{hvs}) * f_d +$
$N_{htc}(VN_S, £_{hvs})$;

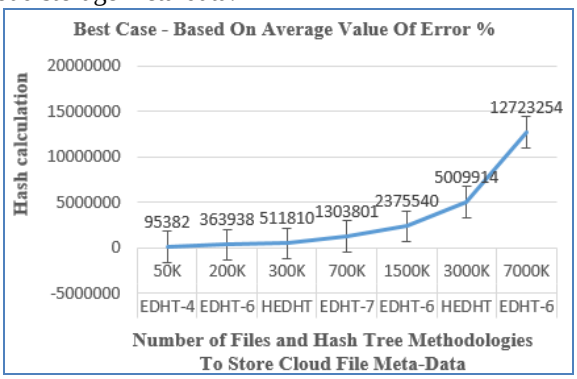
**Average**

$TA_{hcvs} \leftarrow TB_{hcvs} + TW_{hcvs}$
$c_{aic} \leftarrow f_{mhcfc}(TB_{hcvs} or\ TW_{hcvs} or\ TA_{hcvs})$;
iff$_{aic}$!= $c_{aic}$then
$temp_r \leftarrow max_r + 1$;
$f_{aic} \leftarrow c_{aic}$;
else
$max_r \leftarrow N_{fc}$;
$f_{aic} \leftarrow c_{aic}$;
$min_r \leftarrow temp_r$;
end if
end for
print $f_{aic}$+" "+$min_r$+"-" + $max_r$;

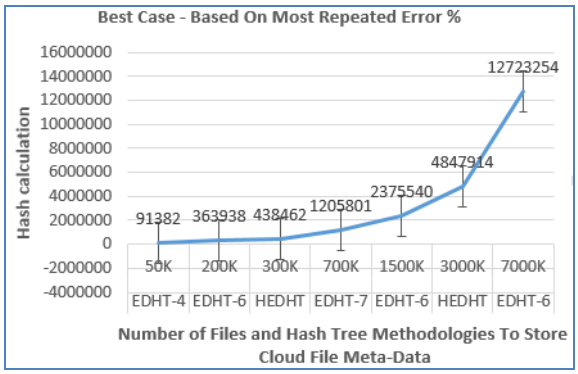
Algorithm 4 is used to determine the optimal hash tree methodology in dynamic way in order to store the cloud file meta-data, which is a primary aim for Adaptive Integrity Checking. We have used all the above algorithms to identify the cloud file range to fit into a specific Best, Worst and Average case possible hash tree methodology as well as to dynamically change the cloud file to store its meta-data in hash tree based on the error pattern, which were recorded periodically by the TPA system, in order to do the speedy recovery by reducing the hash computation process once we identify the file reside in cloud is faulty, subsequently it reduce space as well as time complexity.

## V. RESULTS AND DISCUSSION

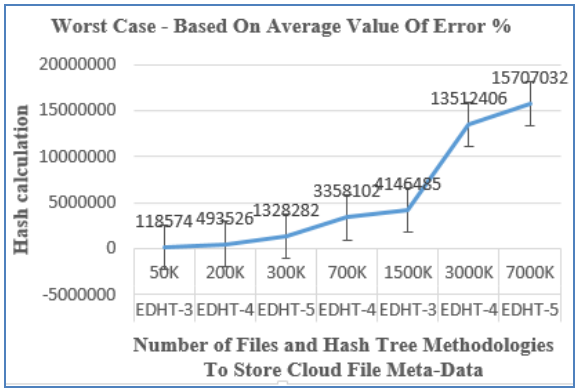
From Graph 1 to 6, it is clearly denotes the best, worst and average Hash Tree Methodologies has been generated dynamically to store cloud file meta-data for auditing purpose based on the repeated value of error percentage as well as average value of the error percentage noted in Table 3 for various number of cloud file storage, thus Adaptive Integrity Checking (AIC) would efficiently recover the file lost as well as auditing process with less number of hash computation and memory space compared to static data structure for storing the cloud storage meta-data.



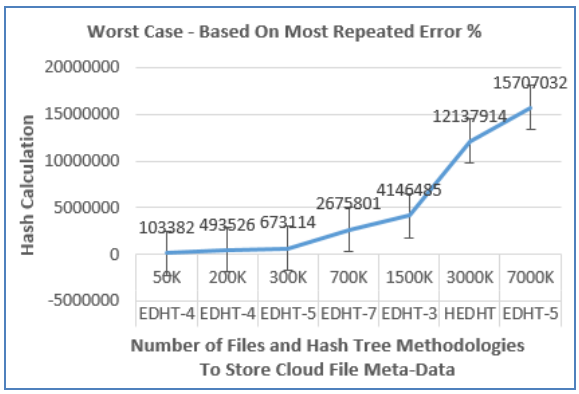

**Graph 1**



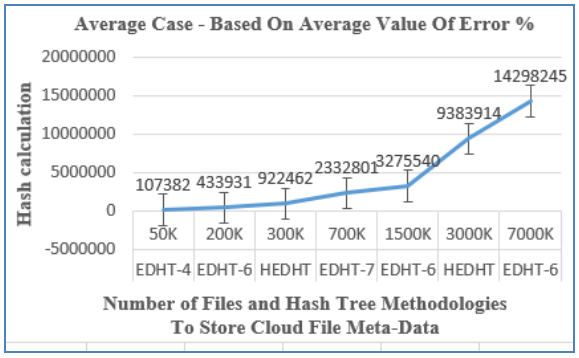

**Graph 2**



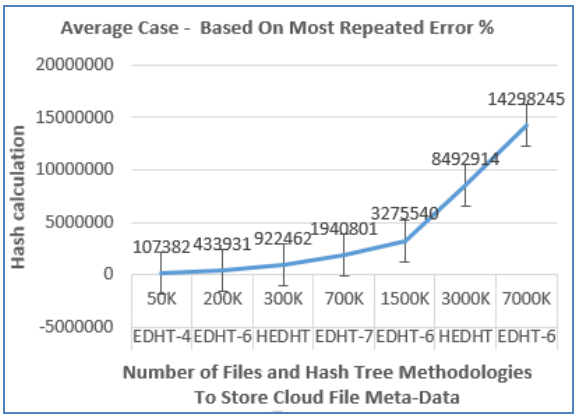

**Graph 3**




**Graph 4**




**Graph 5**




**Graph 6**

## VI. CONCLUSION

Cloud Storage is an Infrastructure as a Service (IaaS) which is a cost effective to an organizations with an integrity challenges, thus storage intactness to be checked periodically. In this paper we have done the research work to find the dynamic hash tree methodology to store the cloud file meta-data to perform the audit process for the cloud storage as well as speedy recovery process for the lost or corrupted file stored in the CSP to maintain integrity thru the efficient proposed EDHT-n and HEDHT data structure rather static way. Also, AIC batch processing has implemented to speed-up both the processes.

## ACKNOWLEDGMENT

In this paper, we proposed Adaptive Integrity Checking based on these methodologies cloud storage integrity is achieved optimally by batch auditing and subsequently batch recovery in case, if file not intact in the cloud storage which has the remarkable future in Cloud Integrity. I solemnly thank my Guide and Computer Science and Engineering Department Professors and University for preparing and implementing this paper.

## REFERENCES

1. A. Juels and J. B. S. Kaliski, “PORs: Proofs of retrievability for large fildoes,” in Proc. 14th ACM Conf. Comput. Commun. Security, Alexandria, USA, 2007, pp. 584–597.
2. Chang Liu, Rajiv Ranjan, Chi Yang, Xuyun Zhang, Lizhe Wang, Senior Member, IEEE, and Jinjun Chen, Senior Member, IEEE, “MuR-DPA: Top-Down Levelled Multi-Replica Merkle Hash Tree Based Secure Public Auditing for Dynamic Big Data Storage on Cloud”, IEEE Trans. Computers, vol. 64, no. 9, Sep. 2015.
3. WentingShen, Jing Qin, Jia Yu, RongHao, and Jiankun Hu, Senior Member, IEEE, “Enabling Identity-Based Integrity Auditing and Data Sharing with Sensitive Information Hiding for Secure Cloud Storage”, IEEE Transactions on Information Forensics and Security, vol. 14, no.2, Feb. 2019.
4. Yuan Zhang, Student Member, IEEE, ChunxiangXu, Member, IEEE, Xiaohui Liang, Member, IEEE, Hongwei Li, Member, IEEE, Yi Mu, Senior Member, IEEE, and Xiaojun Zhang, “Efficient Public Verification of Data Integrity for Cloud Storage Systems from Indistinguishability Obfuscation”, IEEE Transactions on Information Forensics and Security, vol. 12, no.3, Mar. 2017.
5. Dan Gonzales, Member, IEEE, Jeremy Kaplan, Evan Saltzman, Zev Winkelman, Dulani Woods, “Cloud-Trust - a Security Assessment Model for Infrastructure as a Service (IaaS) Clouds”, IEEE Transactions on Cloud Computing, vol 5 , no 3, July-Sept. 1 2017.
6. G. Ateniese, R. Burns, R. Curtmola, J. Herring, L.Kissner, Z. Peterson, and D. Song, “Provable data possession at untrusted storages,” in Proc. 14th ACM Conf. Comput. Commun. Secur., 2007, pp. 598–609.
7. Yujue Wang, Qianhong Wu, Member, IEEE, Bo Qin, Wenchang Shi Robert H. Deng, Fellow, IEEE, Jiankun Hu, “Identity-Based Data Outsourcing with Comprehensive Auditing in Clouds”, IEEE Transactions on Information Forensics and Security, vol. 12, no.4, Apr. 2017.
8. 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. Inform. Syst. Secur., vol. 14, no. 1, pp. 1–34.
9. G. Ateniese, R. Di Pietro, L. V. Mancini, and G. Tsudik, “Scalable and efficient provable data possession,” in Proc. 4th Int. Conf. Secur. Privacy Commun. Netow., 2008, pp. 1–10.
10. KaipingXue,Peilin Hong, 2014, 'A Dynamic Secure Group Sharing Framework in Public Cloud Computing', IEEE Transactions on Cloud Computing, vol. 2, no. 4, pp. 459-470.
11. JinxiaWei,RuZhang,JianyiLiu,JingLi,XinxinNiu,Yuangang Yao, 2017, 'Dynamic data integrity auditing for secure outsourcing in the cloud', Concurrency and Computation: Practice and Experience, vol. 29, no. 12, p. e4096.
12. C. Erway, A. Kupcu, C. Papamanthou, and R. Tamassia, “Privacy-Preserving Public Auditing for Data,” in Proc. 16th ACM Conf. Comput. Commun. Secur., 2009, pp. 213–222.
13. P. SyamKumar,R. Subramanian, 2012, 'RSA-based dynamic public audit service for integrity verification of data storage in cloud computing using Sobol sequence', International Journal of Cloud Computing, vol. 1, no. 2/3, p. 167.
14. XinTang,YiningQi,Yongfeng Huang, 2016, 'Reputation Audit in Multi-cloud Storage through Integrity Verification and Data Dynamics', 2016 IEEE 9th International Conference on Cloud Computing (CLOUD).
15. S. Venkatesan,AbhishekVaish, 2011, 'Multi-agent Based Dynamic Data Integrity Protection in Cloud Computing', Computer Networks and Information Technologies, pp. 76-82.
16. HyunjooKim,YoungsooKim,IkkyunKim,Hyuncheol Kim, 2018, 'Dynamic Information Extraction and Integrity Verification Scheme for Cloud Security', Mobile and Wireless Technologies 2017, pp. 424-429.
17. P. SanthoshKumar,LathaParthiban,V. Jegatheeswari, 2018, 'Secured data storage and auditing of data integrity over dynamic data in cloud', International Journal of Internet Technology and Secured Transactions, vol. 8, no. 4, p. 528.
18. ChunluWang,ChuanyiLiu,BinLiu,Yingfei Dong, 2014, 'DIV: Dynamic integrity validation framework for detecting compromises on virtual machine based cloud services in real time', China Communications, vol. 11, no. 8, pp. 15-27.
19. Santhosh Kumar, Sathyabama University, Latha Parthiban, Pondicherry Community College, 2017, 'Cloud Data Integrity Auditing Over Dynamic Data for Multiple Users', International Journal of Intelligent Engineering and Systems, vol. 10, no. 5, pp. 239-246.
20. Jingwei Li, Jin Li, DongqingXie, and Zhang Cai, “Secure Auditing and Deduplicating Data in Cloud”, IEEE Trans. Computers, vol. 65, no. 8, Aug. 2016.
21. AotingHu,RuiJiang,Bharat Bhargava, 2018, 'Identity-Preserving Public Integrity Checking with Dynamic Groups for Cloud Storage', IEEE Transactions on Services Computing, pp. 1-1.
22. JianMao,YanZhang,PeiLi,TengLi,QianhongWu,Jianwei Liu, 2017, 'A position-aware Merkle tree for dynamic cloud data integrity verification', Soft Computing, vol. 21, no. 8, pp. 2151-2164.
23. Tao Jiang,XiaofengChen,Jianfeng Ma, 2016, 'Public Integrity Auditing for Shared Dynamic Cloud Data with Group User Revocation', IEEE Transactions on Computers, vol. 65, no. 8, pp. 2363-2373.
24. John Zic,ThomasHardjono, 2013, 'Towards a cloud-based integrity measurement service', Journal of Cloud Computing: Advances, Systems and Applications, vol. 2, no. 1, p. 4.