IJCSE International Journal of Computer Sciences and Engineering Open Access
Research Paper Volume-5, Issue-6 E-ISSN: 2347-2693

ASID: Application Specific Time Efficient Inline Deduplication on Cloud Storage

Jyoti J. Malhotra[1*], Jagdish W. Bakal[2]

[1]Dept. of Computer Science and Engineering, G. H. Raisoni College of Engineering, Nagpur, India
[2]Dept. of Computer Science and Engineering, G. H. Raisoni College of Engineering, Nagpur, India

*Corresponding Author: jyoti.j.malhotra@gmail.com

Available online at: www.ijcseonline.org

Received: 10/May/2017, Revised: 26/May2017, Accepted: 17/Jun/2017, Published: 30/Jun/2017

***Abstract*—** As the third party cloud storage services provide fewer maintenance facilities, various enterprises and organizations are attracting towards them. This results in the huge amount of data outsourcing over cloud storage servers. Uncontrolled data proliferation is the huge issue. This increasing backup data volume needs better data management technique to deflate the storage space for cloud servers. Data deduplication is one of the most popular data management approaches, which does not allow storing duplicate data over the storage space. This paper presents the application specific inline data deduplication system on cloud server side along with the efficient and optimized file upload and download operations. The system frames and compares utility based and object map based duplicate content searching techniques on the file and chunk algorithmic levels. Map object plays an important role in quick searching for the duplicates as it evades read operations of the existing files. For downloading the file, the system also provides the functionality of data integrity checking at server side for cloud users to verify the originality of file. The performance of the system is evaluated on random files in the form of flat files, structured files, and unstructured files. The experimental results prove the performance of deduplication system in terms of time and memory usage.

***Keywords*—** *cloud, application, deduplication, integrity, performance, inline*

I. INTRODUCTION

The increasing amount of data across multiple platforms has drastically increased the demand of Cloud storage requirements. Storage on the cloud is the data representation where the digital data is stored in virtualized logical pools, more specifically saying, it is stored in multiple physical storages which span multiple servers and the physical environment is owned, administered and managed by a hosting company. The Cloud Storage Providers are responsible for monitoring data availability, data accessibility, and data security. Cloud computing involves both software and hardware which are distributed to users as a service by the service providers.

The cloud storage service providers allow the service consumers to store their large quantity of data while considering the factors of the memory space and network bandwidth requirements. With this huge data, arises the problem of memory space; to resolve the problem of memory storage space and manage the data efficiently, data deduplication plays a significant role in cloud computing. This technique manages the scalability and consistency of data.

Data deduplication [1] is the smart compression method which is used for removing the repeated copies of files or data by keeping a unique copy in the storage system to reduce the space occupation. It also enhances the time efficiency of search results. It is one of the fastest rising segments in the storage ecosystem. Deduplication identifies unwanted duplicate or similar files which are replicated over the data repository; deletes the additional copy and stores the single instance of the file by assigning the "reference pointer" for the duplicates. Nowadays deduplication has taken a strong position in the data world and is implemented for databases [2], [3], media and non-media files [4],[5],[6],[7],[8]. It is widely used for rapid, consistent, and cost-efficient data backup and recovery. Deduplication process is categorized into multiple levels namely; algorithm level, source (client) level, and target (server) level [1].

With respect to algorithmic level, duplicate removal process goes through major two elimination pipelines namely, File Level Approach (FLA), Chunk Level Approach (CLA). FLA is the traditional way of checking duplicates on the basis of their hash values. We have preferred using file byte contents to find the duplicates over their hash values. As stated in Equation (1), files F are read by the backup process, to obtain

their bytes <B>, which is transferred to the storage layer, ST for deduplication processing <DD> to eliminate the redundant copies and finally updating the results into the Metadata, MD.

$$F \xrightarrow{<B>} ST \xrightarrow{<DD>} MD \quad (1)$$

In CLA, files F are transported to chunking layer CHUNK, for applying intermediate chunk transformation <i_chT> where the file is separated into static or dynamic size chunks; these chunks are then forwarded to storage layer ST. ST extracts the bytes of the chunks <chB> and performs Deduplication <DD> to eliminate redundant copies in order to update the results into the Metadata, MD as shown in Equation (2).

$$F \rightarrow CHUNK \xrightarrow{<i\_chT>} ST \xrightarrow{\langle chB|DD\rangle} MD \quad (2)$$

There are a few challenges of deduplication system as listed below:

1. The tradeoff between duplicate removal and metadata overhead: This is the biggest challenge of deduplication system. For example, we consider that the file is divided into a number of small sized chunks for deduplication checking, which results in a better saving of memory space as more duplicates are detected. On the other hand, it is not cost effective because it requires a huge metadata index to keep records of each file chunks. Normally metadata is loaded on RAM, but if its size is large, it needs to store on disk, which increases the I/O operations and its cost.

2. The tradeoff between duplicate removal and scalability: More duplicates are removed if more similar chunks are found. But these chunk comparison increases the performance complexity as a number of files are growing. Sometimes it leads to bottleneck on deduplication results.

3. Reliability: Deduplication on cloud storage system should be reliable. Such system must be adaptable to the problems of a node failure, loss of data, corruption of files etc. To recover the loss of data, some replications should be maintained in the system.

4. Security and privacy: In the cloud, various types of data are shared by the users which arise the problem of security and privacy. For security and privacy, some confidentiality and authentication techniques should be applied to cloud storage system.

In this paper, we propose the following features-

1. Application-specific, server-side inline deduplication (ASID) with file and chunk level algorithmic strategy.
2. Backup window, an important asset for calculating the time required for the duplicate check is minimized by storing metadata in the serialized map objects.
3. File integrity is achieved by security tags. Security tags are the combination of user and file details encoded into 4-digit keys and are shared between client and server (cloud) for verifying the file veracity.
4. Successful unique file upload and download operations.

Here, we present the study of previous related work done on the deduplication techniques for backup storage in next section II. The details of algorithmic level and target level deduplication implementation are discussed in Section III, in which we can see system architecture and different modules description. In section IV, we have included the implementation and evaluation results and overall conclusion of the work is stated in section V.

II. RELATED WORK

There are several Proof of Storage (PoS) systems which are used to validate the data files which are stored on the cloud server. Merely all the developed techniques are designed for the singular user environment, while for a multi-user environment, there is no such provision available. A specific multi-user cloud framework must have the secure deduplication at the user end. In the paper [9] authors developed the idea of duplicate dynamic proof of storage as well as implemented an efficient construction known as DeyPoS for getting dynamic PoS as well as secure cross-user deduplication concurrently. In paper [10] cryptographically assured as well as the effectual system has been designed by authors for customers to challenge their ownership of file using three algorithms namely, the key generation between client and server, proof generation by setting a challenge and verification of client owning the file. Developed system [10] uses the system on spot verification where the customer just needs to get to small parts of the first document, dynamic coefficients as well as randomly selected lists of the original files for verifying the file ownership.

Present hierarchical authorized deduplication framework allows the cloud service providers to sort the users based on their privileges. In paper [11] authors developed a new system which secures hierarchical deduplication that maintains the file, fingerprint and query privacy. The developed system also backs authorized searching and dynamic privilege modifications. Inline deduplication is concentrated on secondary storage or on cloud storage, for storing the data as well as for providing some simple application interfaces. The application cannot directly access these interfaces. Despite the fact that the file system provides various application interfaces as well as a number of applications, developing the file system for in-line deduplication has significant issues in I/O path and read operations. File read operation involves activities such as obtaining fingerprints in file recipes gets the address by verifying fingerprint index as well as obtaining related block

of information in the disk drive. This can maximize the latency in a read operation. In paper [12] to minimize the latency, authors have developed Low-Read-Latency File System (LRLFS) for the in-line deduplication. After conducting a number of tests authors claim that LRLFS has a low read latency in reading path.

With respect to deduplication and its scalability, in paper [13] authors have concentrated on the time as well as space needs for the data deduplication. Authors also elaborate the parallel version of chunk level deduplication along with metadata summary. Each node holds the fingerprint summary of all other nodes in memory. At the time of chunk verification, each node verifies the data with chunk metadata. If same chunks are found, the system deletes the repeated chunks. The issue of false positives in the duplicate check is not addressed here. Another high-performance inline deduplication over the cloud is proposed by [14] where authors have integrated the concepts of block level deduplication, multi-cache structure, and bloom filter indexing on solid state drives (SSD). A short stint of encryption, decryption and verification server is illustrated in [15]. With respect to integrity and reliability in the cloud, authors in [16] propose the utilization of public key authentication mechanism to perform auditing without referring the local copy and content of data, thus reducing the communication overhead.

III. SYSTEM OVERVIEW

The proposed system, ASID consists of following two main entities:

1. Client: Client is the registered and authenticated user, who can access the storage system of cloud server to store their data files. This offloading of data files will reduce the space complexity at the user side.

2. Cloud server: Server is responsible for storing and serving the data files of clients. It performs data deduplication to liberate the storage space. The server also maintains the log table for all entries of files uploaded by the clients. The server performs data deduplication to enhance the storage capacity.

A. Architectural view

The system consists of two deduplication enabled modules namely; file upload and file download. The system also provides the file integrity check.

1. File Upload

Figure 1 describes the graphical architecture of file upload procedure with data deduplication processing at cloud server side.

At client side, the files which are to be uploaded are filtered based on their file type. This step identifies the type of files such as flat text, pdf, word, XML or any other type. This file filtration is done to achieve efficient content based deduplication as every file type has its own metadata and content structure for storing the data. In this system, the server performs FLA, where a whole file is processed to byte to byte deduplication check for pdf, doc, XML and various other types of file. While it performs CLA deduplication check for the flat text files. In CLA, equal sized chunks are obtained from the text file and client module launches these chunks of flat files to the server. This reduces the chunking overhead at the server side.

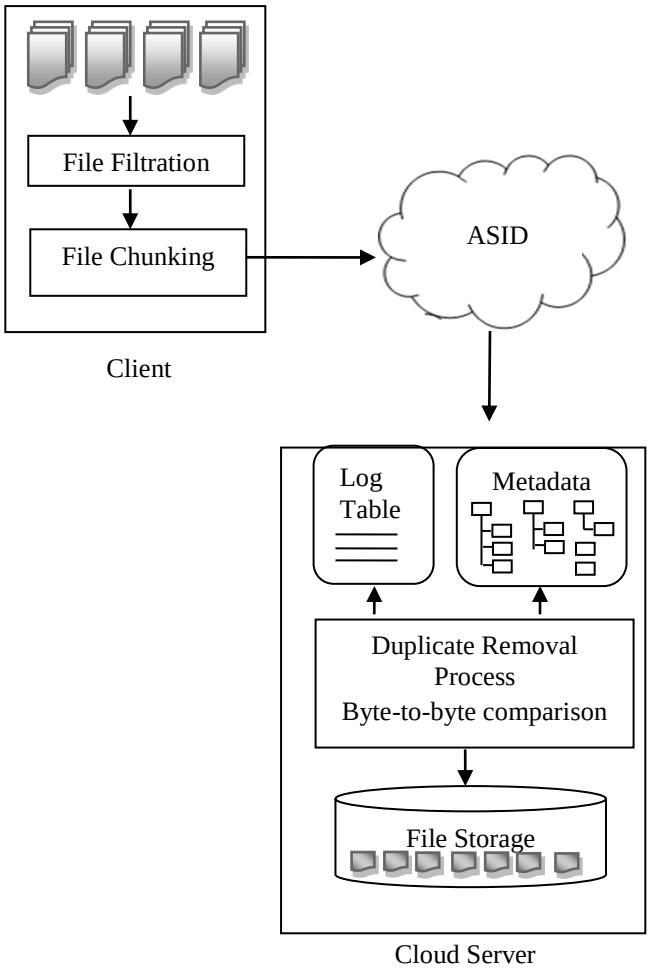

Figure 1: Architectural view for file uploads

At the time of file upload request, a user tag (*userTag*) and file verification tag (*fileTag*) is computed at the user side. Initially, username and password are bounded together to get a 4-digit unique (*userTag*). For *fileTag*; username, password, file name and its hash are bound in a string. After that, the code of this string is computed and further converted to get a 4-digit unique number. This 4-digit *userTag* and *fileTag* acts

as verification codes and are used at the time of file download to verify the authenticity and integrity of the file.

At Cloud server side: After receiving the request for file upload from the client, the server performs duplicate removal process before saving it at its storage directory. Deduplication helps in saving the space of cloud storage.

The possibility of removing the duplicates by deleting the duplicate file can be seen from the probability distribution among four state variables capturing the facet of FB (File bytes match), CB (Chunk bytes match), DUP (Duplicate) and DEL (Delete) as illustrated in Figure 2.

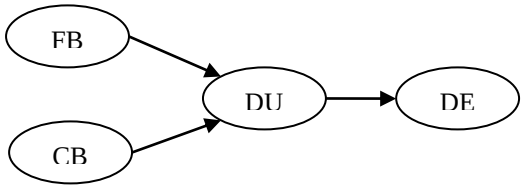

Figure 2: Probability distribution states for duplicate removal

The probability P(fb) denotes that there is a file level byte match. P(cb) denotes that there is a chunk level byte match. P(dup|fb,cb) denotes that there is a probability of getting a duplicate match, given both FB and CB matches; stating the fact that two files are duplicates depends only on the fact that their contents are duplicates and/or their chunks are duplicates. P(del | dup) denotes the probability of deleting a file given that those files are duplicates.

The joint probability of all variables is the product of the individual probabilities is as stated in Equation (3) and (4).

$$P(fb, cb, dup, del) = P(fb)\,P(cb)\,P(dup \mid fb, cb)\,P(del \mid dup) \quad (3)$$

i.e.

$$P(fb, cb, dup, del) = P(fb)\,P(cb)\prod_{i=(fb,cb)} P(dup \mid i)\,P(del \mid dup) \quad (4)$$

We have considered CB for flat text files type and FB for any other file type.

In previous system [17], a check for the duplicate file was based on the hash values [18] of the file content. This method did not give fair justice to the structured type of files such as doc, pdf, etc.; where file content includes the metadata header, storing the details of last accessed time, last modified time, and author details. The hash method produces different hash values even if the file is opened followed by a few changes and then saving back to the original form. This change is because of the header details which updates the modification time on every alteration. Hence, this type of hash value transformation makes the structured files unsuitable candidates for hash comparison. The hash method also does not support the text highlighting and metadata change feature. The working of duplicate removal process with the non-hashed content based method is briefed in Algorithm (1).

We have tested the redundancy of the content using two methods namely, utility based deduplication (UD) and map based deduplication (MD). The utility based method uses the library which compares the documents in text and image mode; however, every check using this method requires a number of file activities such as, opening the file, reading the contents, checking the contents, etc. Though it gives an appropriate result, there is a downside of this method; every time application needs to perform I/O operations which increase the time complexity of the system and resource utilization. To overcome this drawback, we put forward to use the content bytes for duplicate checking at the server side.

Algorithm 1: File upload duplicate removal process

Input: File *F*, File bytes *fbytes*, Client details, *key* for tag generation
Output: Unique file storage
Begin FileUpload
  fileType ← extract_Filetype <*F*>
  userTag ← generateTag <*client details, key*>
  fileTag ←generateTag <*client details, F, key*>
  if (*FileMap* is empty**) then**
    Store <*F, fileType*> //Store file to respective file storage
    Update <*FileMap, fileType*> object instance
    Update user logs;
  else
    FileMap[] ← get existing metadata details (as per file type)
    for each (*fileMapBytes* present in the *FileMap*)
      if (*fbytes == fileMapBytes)* **then**
        File is duplicate; Do not store the copy
        Update *ReferencePointers*;
        Update the user logs;
      else
        New file; Store <*F, fileType*>
        Update <*FileMap, fileType*> object instance
        Update user logs
      end if
    end for
  end if
End FileUpload

Once the file is received at the server, metadata is scanned for some previous values in order to make a lookup. If metadata is empty, the file is directly stored in the cloud without a duplicate check. With non-empty metadata; byte contents of the file are compared with the existing files metadata contents. *ReferencePointers* are created for

duplicate files and unique files are stored in the respective file storage directory based on its file type. On every upload, user's log files are updated with the details such as- file name, its reference, date and upload time. This log file is referred at the time of file download for data integrity and for assuring that the client is downloading the original file which is not modified or tampered on the cloud.

Optimized metadata search is accomplished by implementing the serialized object map instances for every file type. This metadata file object map, *FileMap* contains the details of files along with the file bytes and is created at the time of first file upload. When a new file upload request arrives, this object map is de-serialized for the duplicate check. If there are any new entries, this object map is updated with the new values. Use of object map helps to achieve fair duplicate detection without any read complexities and unnecessary I/O access.

2. File download and Integrity check

Algorithm (2) and (3) computes few steps for the file download process along with data integrity checking. On the file download request, cloud server performs *userTag* authentication. Later, server ensures the username and the list of files uploaded by that user in the log table. According to file download request, the server retrieves the number of files associated with the requesting user name from log table. The server checks the file reference- *fileRef* in *ReferencePointers* data structure if *fileRef* is found then the particular file or block is sent to the user by accessing its respective reference pointer(s) and the entries of *FileMap* . Otherwise, server searches *FileMap* for the corresponding file based on its type and the file is sent for the successful download.

Algorithm 2: File download process

Input: Filename to be downloaded *fname*, *userTag*, and client username *uname*
Output: Respective file
Begin FileDownload
flag ← authenticateUserTag <*userTag*>
if (*flag* is true**) then**
fileRef ← searchReference (*ReferencePointers)*
if (*fileRef* is found) **then**
Send file client *(FileMap[], fileRef, fileType*)
else
Send file to client *(FileMap[], fName, fileType*)
end if
else
Invalid client for file download
end if
End FileDownload

The integrity of the file validation is illustrated below in Algorithm (3).

Algorithm 3: File Integrity check

Input: Filename *fname, fileTag* and user Logs
Output: Boolean value {true, false} on file modification
Begin FileIntegrity
status ← checkModificationStatus (*userLogs*)
if (*status* is same**) then**
flag ← verifyTags (*fileTag*)
if (*flag* is true) **then**
File is not modified
else
File is modified
end if
else
File is modified
end if
End FileIntegrity

On receiving the file from the server, the user can ensure the integrity of the file by sending a request to the cloud server. In response, cloud server accepts the request and checks user log record. This record contains the details of file upload such as user name, file name, last upload time, last modification time and other file attributes. The system reads the modification time of user's log and modification time of the file at cloud server. If it matches a second level integrity check is done by accessing *fileTag* of both the files. A tag and attribute match conveys that file is not tampered at the server side and the user is provided with the original file.

IV. IMPLEMENTATION AND EVALUATION

Standard benchmarks for deduplication efficiency are not available; as DD performance depends on the amount of duplicate data content. To evaluate the performance of ASID, we have conducted the experiment with CloudZone storage. Cloud framework was simulated with CloudZone, which is the cloud-based storage system. It allows the users to store data, retrieve data along with simultaneous read and write operations of data. To maintain the integrity of data, we have imposed *userTag* and *fileTag* verification. Data files are stored on a cloud on different volumes of data for different types of file.

To evaluate the performance of the system, we conducted operations like file upload, file download with different type and size of files, and file integrity check. The work was tested with four sets of data-D1, D2, D3, D4. In the first set, we uploaded unique and duplicate pdf files of specific sizes of KBs and MBs, followed by word files, text files and other types of file in the second, third and fourth dataset. An inline check on the duplicates was done before saving the files on the storage. In Figure 3, we see the amount of duplicate detection in terms of DD ratio (pre_DD size / post_DD size) and DD percentage (amount of duplicate detection and its removal). DD ratio is bounded between the scale 1 to 4 for

DD percentage range 0% to 70%. For dataset D1, the recorded DD ratio was 1.37 with 27% duplicate removal. The documented ratio and percentage for data sets D2, D3, and D4 were 2.42, 1.24, 1.52 and 58.6%, 20%, and 34.25% respectively.

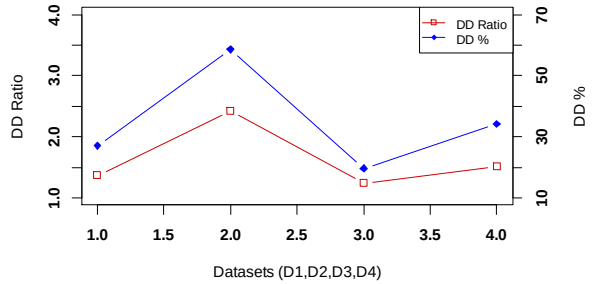

Figure 3: Deduplication ratio and percentage for given four sets of data

Figure 4 plots the runtime memory usage of the ASID for both the methods UD and MD. It is tested on various file upload instances (I_1 to I_{10}). The current runtime of the file upload instance is measured and is presented in the logarithmic form. The runtime occupied by both the methods is almost same. UD and MD methods occupy almost the same runtime with a slight variation in the memory usage. This variation is bounded in the range from 5% to 20% and is also dependent on the file sizes and the external processes and resources such as CPU, disk and memory usage.

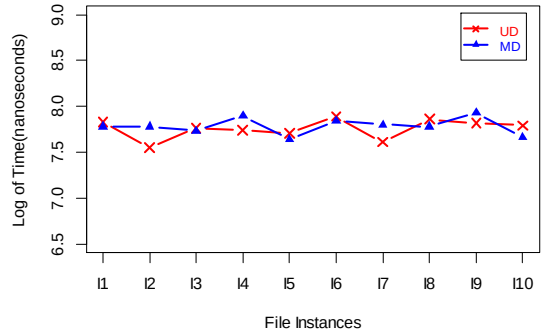

Figure 4: Memory usages for UD and MD for file upload instances from datasets

Figure 5 shows the file upload time comparison of UD versus MD-based system. Required time to identify duplicates at the time of file upload with both the methods is measured in nanoseconds and the time is transformed to log scale to avoid the high skewed distributions of the time required for uploading various files of different type and sizes. Map based deduplication method MD takes less time as compared to UD method. UD performs duplicate checking with various tasks such as opening a file, reading the contents and perform a duplicate check based on the data content. The time required for all these operations increase the overall time for UD method. While on another side, MD performs an optimized search by encapsulating the file details in the map object; which is referred for the duplicate verification and removal. For given file instances (I_1 to I_{10}).; with MD, required time falls in the logarithmic range of 6 to 7.5 and for UD it is in the range of 9 to 10.

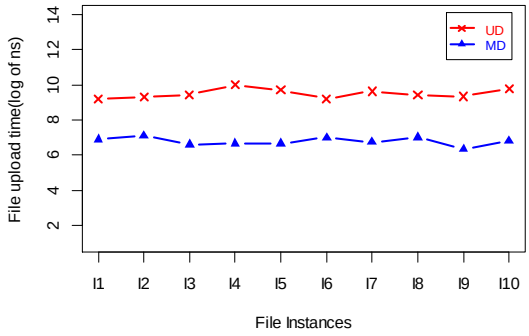

Figure 5: Logarithmic upload time for UD Vs MD on file instances from datasets

The UD method shows the logarithmic range in 9 to 10 and MD method's time range goes in 6 to 7.5 with a percentage difference of 25%. Therefore, it can be distinguished that the MD approach works better over the UD approach in the ASID; thereby reducing the backup window which is one of the essential inline deduplication challenges.

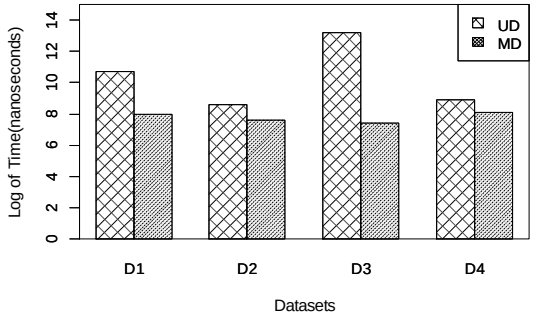

Figure 6: Logarithmic upload time for UD Vs MD on the datasets

Overall time required to upload the dataset files is shown in Figure 6. Files are uploaded with the FLA and CLA algorithmic pipelines. We see that time required with MD is less as compared to the upload time with UD method for all the datasets. The time difference percentage between two methods for four datasets is 25.47%, 11.63%, 43.47% and 9.1% respectively.

Along with file upload time, we have also measured the average time required for file download and the file integrity check. As can be referred from Algorithm 2 and 3, we use 4-digit *userTag* and *fileTag* for verifying the authenticity and integrity of the files to be downloaded. Figure 7, plots the cumulative distribution of time in milliseconds needed for generating the tags. The time required for *userTag* generation

is less than 50ms. *fileTag* generation varies between the range 10 ms to 130 ms. When compared to *userTag* generation, *fileTag* generation takes little additional time because it involves the hash calculation of the file content along with other details of the user.

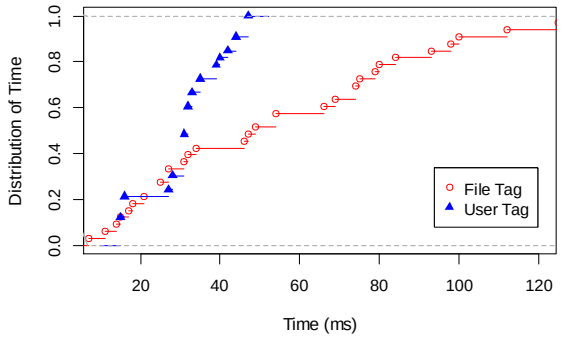

Figure 7: Cumulative distribution of time required for user and file tag generation for file integrity check

Figure 8 shows the graphical sketch of the spread of time required for file download and the file integrity check. Required time for various file instances is shown in the enclosed range of milliseconds. Time with respect to file download and file integrity are represented in the left and right whiskers respectively. Median time required for file download is 6000ms with lower whisker below 2000ms and higher whisker near to 10000 ms. This means for some files download time was below 2000ms and for other files, it was in the range from 2000 to 10000 ms. On the left whisker, we can depict an outlier towards 14000ms for downloading files of larger size. File download time depends on the factors considering file size, metadata size, metadata search and number of file references.

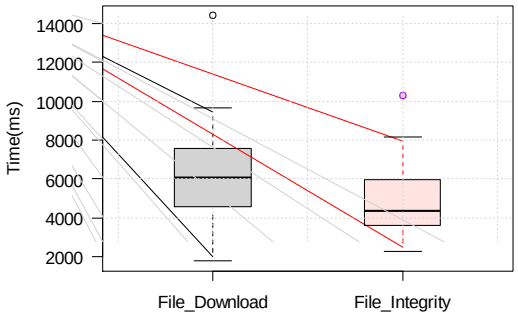

Figure 8: Box plots distribution of quartile values of time required for file download and integrity check

Median time required for file integrity check is slightly above 4000ms with lower whisker near to 2000ms and higher whisker near to 10000 ms. This means for some files download time was below 2000ms and for other files, it was in the range from 2000 to 8000 ms. On the right whisker, we can see an outlier towards 10000ms an exceptional value required for large size files.

V. CONCLUSION

This paper presents the time and memory efficient, inline data deduplication system on the cloud using map based object for quick duplicate detection. Experiments are performed on the files using FLA and CLA approaches. The evaluation analysis illustrates that for file upload operations, the proposed MD approach demonstrates 8% to 12% improved performance over UD approach, thus reducing the backup window time required for performing deduplication storage. As file maps are stored in the form of the object there are less file read operations and disk hits. The system is also enhanced with the data integrity checking for file downloads where 4-digit security tag computations are used to verify the original content. The experimented results confirm the duplicate removal percentage of 20%, 27%, 34% and 58.6% for the respective datasets across the users. In addition to that, the integrity of the file is confirmed in a minimum amount of time duration which includes the verification of user and file tags.

ACKNOWLEDGMENT

We wish to show our sincere gratitude to everyone involved in the progress of this paper.

REFERENCES

[1] N. Mandagere, P. Zhou, M.A. Smith, S. Uttamchandani, "*Demystifying data de-duplication*", In the Proceedings of the ACM/IFIP/USENIX Middleware'08 *Conference Companion*, ACM, Belgium, pp. 12-17, 2008.

[2] Y. Jiang, C. Lin,W.Meng, C. Yu, A. M. Cohen, N. R. Smalheiser, "*Rule-based deduplication of article records from bibliographic databases",* Database(Oxford)-The Jouornal of Biological Databases and Curation, 2014.

[3] M. Carvalho, A. H. Laender, M.A. Goncalves, A. S. da Silvaet, "*A genetic programming approach to record deduplication*." IEEE Transactions on Knowledge and Data Engineering, Vol. 24, Issue 3, pp.399-412, 2012

[4] Y. Li, K. Xia, "*Fast Video Deduplication via Locality Sensitive Hashing with Similarity Ranking"*. In the Proceedings of the 2016 International Conference on Internet Multimedia Computing and Service, ACM, China, pp.94-98, 2016.

[5] O. Murashko, J. Thomson, H. Leather, "*Predicting and Optimizing Image Compression*." In the Proceedings of the 2016 ACM on Multimedia Conference, Amsterdam, The Netherlands, pp. 665-669, 2016.

[6] D. Kim, S. Song, B.Y. Choi, "*SAFE: Structure-aware file and email deduplication for cloud-based storage systems*". In Data Deduplication for Data Optimization for Storage and Network Systems. Springer International Publishing. pp.97-115, 2016.

[7] X. Du, W. Hu, Q. Wang, F. Wang, "*ProSy: A similarity based inline deduplication system for primary storage."* In the proceedings of 2015 IEEE International Conference on Networking, Architecture, and Storage (NAS) Boston, USA, pp. 195-204, 2015.

[8] A. S. Agrawal, J. Malhotra, "*Clustered Outband Deduplication on Primary Data*" In the proceedings of 2015 IEEE International Conference on Computing Communication Control and Automation (ICCUBEA 2015), Pune, India, pp. 446-450, 2015.

[9] K. He, J. Chen, R. Du, Q. Wu, G. Xue, X. Zhang, "*DeyPoS: Deduplicatable Dynamic Proof of Storage for Multi-User Environments*," IEEE Transactions on Computers, Vol. 65, Issue. 12, pp. 3631-3645, 2016.

[10] C. Yang, J. Ren, J. Ma, "*Provable ownership of file in de-duplication cloud storage*," Security and communications network journal, Vol. 8, Issue. 14, pp. 2457-2468, 2013.

[11] X Yao, Y. Lin, Q. Liu, Y. Zhang, "*A secure hierarchical deduplication system in cloud storage*," In the proceedings of IEEE/ACM 24th International Symposium on Quality of Service (IWQoS), Beijing, China, pp. 1-10,2016.

[12] Y. Zhou, Y. Deng, Y. Li, J. Xie, "*Reducing the read latency of in-line deduplication file system*," In the proceedings of IEEE 34th International Performance Computing and Communications Conference (IPCCC), Nanjing, China, pp. 1-2, 2015.

[13] G. Wang, Y. Zhao, X. Xie, L. Liu, "*Research on a Clustering Data De-Duplication Mechanism Based on Bloom Filter*," In the proceedings of IEEE International Conference on Multimedia Technology(ICMT, 2010), Ningbo, China, pp. 1-5, 2010.

[14] J. Wang, Z. Zhao, Z. Xu, H. Zhang, L. Li, Y. Guo, "*I-sieve: An inline high performance deduplication system used in cloud storage.*" IEEE transactions, Tsinghua Science and Technology, Vol. 20, Issue. 1, pp. 17-27, 2015.

[15] Z. Wen, J. Luo, H. Chen, J. Meng, X. Li, J. Li, "*A Verifiable Data Deduplication Scheme in Cloud Computing*," In the proceedings of International Conference on Intelligent Networking and Collaborative Systems, Salerno, Italy, pp. 85-90, 2014.

[16] M.S. Sulthana, T. Samatha, V. Sravani, A. Mahendra, "*Multiple Auditing Schemes with Integrity and Reliability in Cloud Computing*". International Journal of Computer Sciences and Engineering (IJCSE) Vol. 5, Issue.5, pp. 1-6, 2017.

[17] J. Malhotra, J. Bakal "*FiLeD: File Level Deduplication Approach*". International Journal of Computer Trends and Technology (IJCTT) Vol. 44, Issue. 2, pp.74-79, 2017.

[18] Network working group, RFC 3174 - US Secure Hash Algorithm-1, September 2001.

Authors Profile

Jyoti J. Malhotra is M.E. in Computer Science and Engineering from the SPPU University Pune in 2005. She has 15+ years of teaching experience. She is pursuing her Ph.D. degree in Computer Science and Engineering; RTM Nagpur University under the guidance of J. W. Bakal. Her research interest lies in Data Storage patterns, Big Data, Software Testing and Theory of Computation. She has worked in C, Java, and Linux Programming. She is the life member of Computer Society of India. She has publications in National, International conferences and Journals like IEEE Conference, Springer Conference, etc.

Jagdish W. Bakal received M. Tech. (EDT), from Dr. Babasaheb Ambedkar Marathwada University, Aurangabad. Later, He completed his Ph.D. in the field of Computer Engineering from Bharati Vidyapeeth University, Pune. He is presently working as Principal at the S.S. Jondhale College of Engineering, Dombivali (East) Thane, India. In the University of Mumbai, he was on honorary assignment as a chairman, board of studies in Information Technology and Computer Engineering. He is also associated as chairman or member of Govt. committees, University faculty interview committees, for interviews, LIC or various approval works of institutes. He has more than 27 years of academics experience including HOD, Director in earlier Engineering Colleges in India. His research interests are Telecomm Networking, Mobile Computing, Information Security, Sensor Networks and Soft Computing. He has publications in journals, conference proceedings in his credit. During his academic tenure, he has attended, organized and conducted training programs in Computer, Electronics & Telecomm branches. He is a Professional member of IEEE. He is also a life member of professional societies such as IETE, ISTE INDIA, CSI INDIA. He has prominently contributed in the governing council of IETE, New Delhi India.

.

