VOL. 10, NO 20, NOVEMBER, 2015 ISSN 1819-6608
ARPN Journal of Engineering and Applied Sciences

www.arpnjournals.com

AN INTRICATE STRINGENT CHUNKING (ISC) DE-DUPLICATION FOR SPACE OPTIMIZATION IN PRIVATE CLOUD STORAGE BACKUP

M. Shyamala Devi[1], Yella Sravya[2] and Mounika Yarasi[3]
[1]R. M. D Engineering College, Computer Science and Engineering, Chennai, India
[2]System Engineer Trainee, Infosys, Mysore, India
[3]Advanced Java Trainee, NIIT, Tirupati, India
E-Mail: shyamalapmr@gmail.com

ABSTRACT

Cloud Computing has evolved as a service as well as deployment model over the decade in catering data storage and data access technology. Many private data owners prefer utilizing cloud storage service model due to flexibility in maintaining infrastructure, cost and ability to access data over internet. The cost of cloud utilization is determined by the amount of data stored on the cloud environment. It is imperative that optimization of cloud storage for effective data usage enables saving cost, space and effective data utilization. Recent advancements have established digital data in storage medium are redundant and data compression is effective in eliminating data redundancy. Deduplication techniques have been devised to identify and eliminate identical data in cloud. As private cloud storage has limited hardware resources and infrastructure, it is essential to optimally utilize the storage space to be able to hold maximum data. In this paper, we discuss the limitations of the existing de-duplication methods and propose a new scheme for Data De-Duplication. The proposed method of Intricate Stringent Chunking (ISC) De-duplication which is the enhanced File level de-duplication provides dynamic space optimization in private cloud storage backup as well as increases the throughput and enhances the efficiency of deduplication.

Keyword: cloud computing, private storage cloud, cloud backup, data de-duplication, chunking, redundancy.

1. INTRODUCTION

Cloud Computing has emerged as a delivery model in providing internet based applications, web services and IT infrastructure using utility pricing model. Public clouds are administered by third party service providers and applications from different data owners are managed together on the cloud servers, storage systems, and networks. Private clouds are designed for the limited use of one client and and managed by the individual organization's own administrator. Hybrid clouds are the combination of both public and private cloud models.

A. Cloud storage

Cloud storage is a service model that stores data from data owners. The cloud storage is managed and backed up remotely thus made available to users over a network administered by the data owners. Cloud storage facilitates users with storage space and responds data users to handle user friendly and data requests on the acquired data which is the underlying principle of all kinds of cloud applications [3]. Public cloud storage such as Amazon's Simple Storage Service (S3) provides a multi-tenant storage environment [19]. Private cloud storage services provide a dedicated infrastructure protected behind an organization's firewall. Private clouds are suitable for data owners who require customization and sophisticated control over their data as shown in Figure-1. Hybrid cloud storage is the combination of at least one private cloud and one public cloud service model. Cloud storage backup [3] is the service model to back up data that involves responding to data requests offsite to a managed service provider for protection.

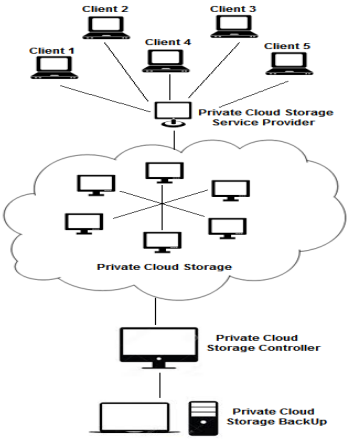


Figure-1. Private cloud storage.

B. Overview of De-duplication

Data Deduplication is the technique that identifies duplicate data to eliminate redundancy in the stored data and increases the capacity of data transferred and stored on the storage medium. De-duplication is also called as "intelligent compression" or "single-instance storage" to mean that it eliminates redundant data thus reducing the

space required or data storage [10]. For example, if an organization's email management data contains 100 instances of a same file attachment. If the email platform is backed up or archived, all 100 instances are saved, requiring all of the same attachment is stored in the storage medium as 100 files. With the advent of data de-duplication techniques, only one instance of the file attachment can actually be stored. Each subsequent instance would be marked to reference back to the one saved file.

C. De-duplication techniques

Several optimization techniques are in place to handle de-duplication of the stored data. The optimization of backup storage is shown in figure 2. The Data de-duplication can be handled at the whole file, block, and bit level [1, 2, 5]. Whole file de-duplication identifies the hash value for the entire file which is the file index. If the new file being added to the cloud storage matches with the file index, then it is considered as duplicate and a pointer is added to reference to the existing file index. If the new file does not match the file index then it is added to the cloud storage as a new file index.

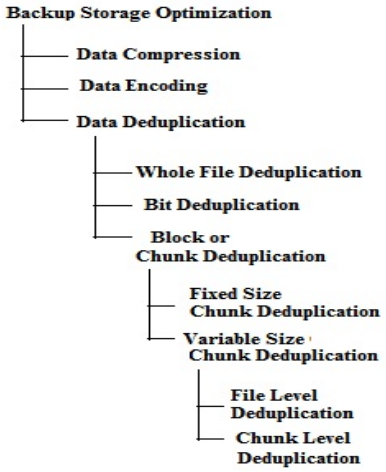


Figure-2. De-duplication methods.

Block De-duplication [4, 5] is a method that divides the files into fixed-size block or variable-size blocks. In Fixed-size chunking, a file is partitioned into fixed size chunks to occupy certain bytes of space. In Variable size chunking, a file is partitioned into chunks of different size. Both the fixed and variable size chunking creates unique identifier for each block using a hash algorithm such as MD5 or SHA-1 or MD5. The unique identifier is then compared with a central index. If the identifier exists, then it would be deciphered as data block has been processed and stored earlier. Hence, the method will just need to save a pointer to the already stored data. If the identifier is new, it means the block is unique. A unique identifier is added to the index and the unique chunk is stored. Block de-duplication and Bit de-duplication searches within a file and saves unique iterations of each block or bit.

The rest of the paper is organized as follows. In Section II, we analyze the existing methods of de-duplication with its advantages and disadvantages. In Section III, we discuss about our proposed system and its functions. In Section IV, we conclude our design of Intricate Stringent Chunking and prove that our scheme greatly increases the de-duplication efficiency. We show our implementation analysis in Section V.

2. ANALYSIS OF EXISTING METHODS

A. Advantages of existing methods

i) The indexes for whole file de-duplication are significantly smaller which takes less computational time and space when determining the duplicates. Backup performance is less affected by the de-duplication process.

ii) Fixed-size chunking is conceptually simple and fast as it requires less processing power due to the smaller index and reduced number of comparisons.

iii) In variable size chunking, the impact on the system performing the inspection and recovery time is less. The efficiency of identifying the duplicate is high.

iv) Bit De-duplication performs bit level de-duplication and it is more efficient since it eliminates redundancy at bit level.

B. Disadvantages of existing methods

i) The efficiency of Whole File de-duplication is not good as a little change within the file makes the whole file to be saved again. For example, if 100 identical attachments are sent by a telecom provider, this method identifies all those 100 attachments that are exactly same in size, but it would not find the exact duplicate copies that are saved (i.e.) Bill. Feb, Bill. Mar, Bill. Apr etc. This de-duplication searches only the size of the file irrespective of the content on each file.

ii) When a small amount of data is inserted into a file or deleted from a file in Fixed-size chunking, a completely different set of chunks is generated from the new file.

iii) In both fixed and variable size chunking, the indexes are large resulting in larger index table and higher number of comparisons. This leads to low throughput and higher processing time to identify the duplicate bit.

C. Methods of block level de-duplication

The block level de-duplication partitions the incoming file into fixed size chunks or variable size chunks. Depending on the duplicate detection of incoming chunk, the variable size chunk de-duplication can be

divided into Chunk level de-duplication and File level de-duplication.

D. Chunk level De-duplication - DDDFS

When a file is received and determined to be written in cloud server, every chunk of the file is verified for duplicate with chunks of all files. This is the methodology used in Chunk level de-duplication to identify duplicates in the file. Data Domain De-duplication File System (DDDFS) is a file system which performs chunk level de-duplication [5] which supports multiple access protocols. This is managed by the interfaces such as Network File System (NFS), Common Internet File System (CIFS) or Virtual Tape Library (VTL) to a generic file service layer.

File service layer manages the file metadata using Namespace index and sends the file to the content store. Content store partitions the file into chunks of variable size. Secure Hash Algorithm SHA-1 or MD5 identifies the hash value for each variable size chunk which is called as ChunkID. Content store maintains the File Reference Index (FRI) which contains the sequence of ChunkID constituting that file. Chunk store maintains a chunk index that is used for duplicate chunk detection. Chunk index is the metadata which usually contains ChunkID and the address of actual chunks in cloud storage. Unique chunks will be compressed and stored in the container.

E. File level De-duplication - extreme binning

When a file is received and determined to be written in cloud server, every chunk of that file is checked for duplicate with all the chunks of the similar files. This is the approach of identifying duplicates called as File level de-duplication. Extreme Binning uses this approach by dividing the chunk index into two tiers namely Primary index and Bin [4]. Primary Index contains the representative ChunkID, whole file hash and pointer to bin. The disk includes bin, Data chunks and the File recipes. The file recipes contain the sequence of chunked for that file. Bhagwat, et al., [4] explained the logic for knowing the structure of a backup node in extreme binning de-duplication. When a file has to be backed up, it performs variable size chunking and finds the representative ChunkID and the hash value for the entire file. The Representative ChunkID is verified in the primary index. If it is not present in primary index, then the incoming file is identified as a new one and a new bin is created with all ChunkID, chunk size and a pointer to the actual chunks which are added to the disk. Representative ChunkID, file hash value and the pointer to bin of the newly created bin are added to the primary index. If the representative ChunkID, file hash of the incoming file is already present in the primary index, then the file is a duplicate and it is not loaded into disk and the bin is not updated. If the representative ChunkID of the incoming file is already present in the primary index but the hash value of the whole file does not match, then the incoming file is considered to be nearly similar to the one that is already on the disk. Most of the chunks of this file will be available in the disk. The corresponding bin is loaded to RAM from the disk, and now searches for the matching chunks of the incoming file. If the ChunkID is not found in the bin, then its metadata of the chunk is added to the bin and the corresponding chunk is written to the disk. The whole file hash value is not modified in the primary index and the updated bin is written back to the disk. Here every incoming chunk is checked only against the indices of similar files, this approach achieves better throughput compared to the chunk level de-duplication. Since non-traditional backup workload demands better de-duplication throughput, file level de-duplication approach is more suited in this case.

3. OUR CONTRIBUTION

A. Proposed system

Generally the backup of the private storage cloud belongs to the non-traditional backup. Traditional backup contains data streams with locality of reference. But the non-traditional backup contains the individual files that owns by the individual users of the organization with no locality of reference. The storage of the private cloud should be optimized as there is physical limitation on the storage space. Here we try to enhance the File level de-duplication since it provides high de-duplication throughput. However a single primary index is used for de-duplication that takes more time in merely checking the representative ChunkID of the file. This leads to low de-duplication throughput. So we try to refine file level de-duplication further to increase the throughput and de-duplication efficiency. So we propose a new method for de-duplication namely Intricate Stringent Chunking (ISC) File De-duplication which is the modified File Level de-duplication that provides grouping of files of individual users. Our System architecture has four components as users, Cloud Storage Controller, Intricate Stringent Chunker and is shown in Figure-3.

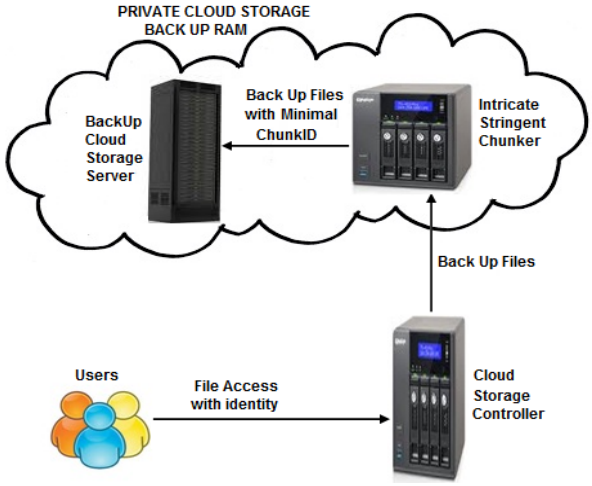


Figure-3. Intricate Stringent Chunking (ISC) File De-duplication – system architecture.

B. Intricate Stringent Chunking (ISC) File De-duplication

The existing File Level de-duplication (Extreme Binning) is shown in Figure-4. The single Primary index contains representative ChunkID, whole file hash and bin pointer which points to the bin of the backup node which is used for finding the duplication regardless of the users of the private cloud which leads to low de-duplication throughput.

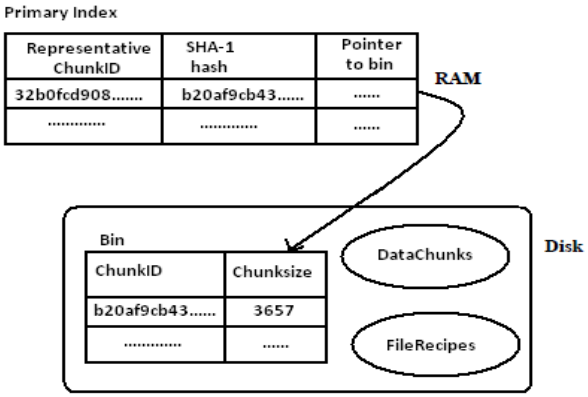


Figure-4. Backup node in extreme binning.

The single Primary index contains representative ChunkID, whole file hash and bin pointer which points to the bin of the backup node which is used for finding the duplication regardless of the users of the private cloud which leads to low de-duplication throughput. Private storage cloud consists of personal documents of the individual users belonging to organization. If we use Extreme Binning, then there will be only one primary index for all user files. So all the incoming files that belong to the different user's merely waste time for checking the representative chunkID of the single primary index that reduce the throughput and de-duplication efficiency. In our Intricate Stringent Chunking (ISC) File De-duplication, the users accessing the private cloud storage are identified by their unique user-id. Here the Chunk File index is divided into Intricate Index, Intricate Stringent Index, Chunk Index and Bin. We create separate Intricate Index, Intricate Stringent Index, Chunk Index and Bin for all the files of each user and each file belonging to an individual user is grouped with their folders and is shown in Figure-5. With this method, it is possible to group the files of each users of the organization.

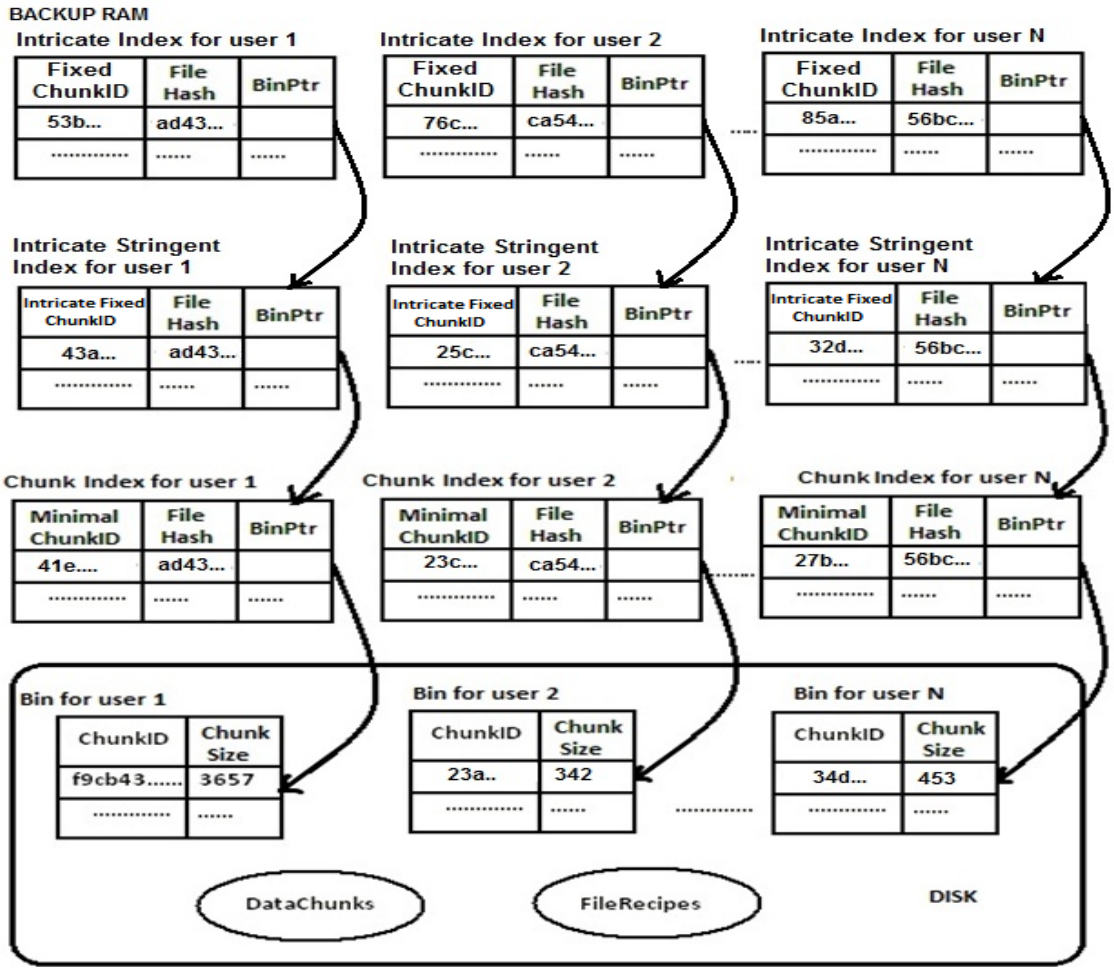


Figure-5. Chunk file index of intricate Stringent Chunking (ISC) File De-duplication.

4. DESIGN OF INTRICATE STRINGENT CHUNKING (ISC) FILE DE-DUPLICATION

Before we start our design of ISC De-duplication, we have the following assumptions,

- Users of the Private Cloud are provided with the separate Userid.
- The files of the individual users are collected in separate folders in the cloud backup.
- Bin of Backup Cloud storage server contains the files of each users as separate folders

A. Function definition of ISC

Our new Intricate Stringent Chunking (ISC) File De-duplication scheme has the following functions,

Functions of data users

a) Provide Cloud Service( )

Functions of Cloud Storage Controller:

b) Initiate Cloud Storage Service( )

c) Control Cloud Storage Service( )

Functions of Intricate Stringent Chunker:

d) Intricate Stringent Chunking( )

e) Deduplicate Cloud Backup( )

The Cloud Service Providing module is executed by data users. The Cloud Storage Initiation module and Cloud Storage Controller module are executed by Cloud storage controller. The Cloud Storage Controller groups all the user files as separate folders and is sent to intricate stringent chunker. The Intricate stringent chunking module and Cloud Backup Deduplication module is executed by intricate stringent chunker.

B. Cloud service providing module

The user authentication is done in this module. If the user is new, then the registration process is done in this module and is shown in Algorithm 1.

Algorithm 1: Provide cloud service ( )

Input: Login Request from user

Output: Authenticated user details

1. Receive the login user name
2. Get the login password
3. Verify the authenticity of the user
4. if (user = = existing user)
5. then provide useraccess with CSC
6. else
7. Request for new user registration
8. call InitiateCloudStorage( )

C. Cloud storage initiation module

After the user authentication is done in the private cloud, then he / she can start viewing, editing and saving their personal files into their folders and it is shown in Algorithm 2. In this module, the authenticated user can perform their own work and they may also try to upload the files from online

Algorithm 2: Initiate cloud storage service ( )

Input: New User Registration and Authenticated User request from Cloud Service Providing module.

Output: Enabling viewing, editing and saving the file.

1. Receive the Authenticated user request.
2. Get the user login and password
3. Verify the authenticity of the user
4. if (user = = existing user)
5. then provide access for viewing, editing and saving their personal files
6. else
7. Request for new user registration
8. call ControlCloudStorageService( )

D. Cloud storage controller module

This module performs the function of integrating the files of the individual users. This module groups the files of all users and is shown in Algorithm 3.

Algorithm 3: Control cloud storage service ( )

Input: Files of the User from Cloud Storage Initiation Module

Output: User Grouped files in separate folders

1. Receive the files from user.
2. Separate the files of each users
3. Create individual folders for each users
4. Group the files in folders of respective users.
5. call Intricate Stringent Chunking ( )

E. Intricate stringent chunking module

The Intricate Stringent Chunking module is executed by Intricate Stringent Chunker. The Chunk File index for each user is created in this module and it is shown in Algorithm 4.

Algorithm 4: Intricate stringent chunking ( )

Input: User Grouped files in separate folders from Cloud Storage Controller Module.

Output: Creating Chunk File Index in three forms as Intricate Index, Intricate Stringent Index, Chunk Index for all User files

1. Receive the file folders of all the users
   / * creating Intricate Index with three field's namely Fixed ChunkID, FileHash and Binptr for each user */
2. Find the size of the file as FileHash.
3. Divide the files equally into fixed sized chunks.
4. Each Fixed Sized chunks is updated as Fixed ChunkID in Intricate Index.
5. Update FileHash value in Intricate Index.
6. Link the BinPtr of Intricate Index to their respective Intricate Stringent Index of each user.
   / * creating Intricate Stringent Index with three field's namely Intricate Fixed ChunkID, FileHash and Binptr for each user */
7. Divide each Fixed ChunkID in Intricate Index into fixed sized chunks as Intricate Fixed ChunkID.
8. Update FileHash value in Intricate Stringent Index.
9. Link the BinPtr of Intricate Stringent Index to their respective Chunk Index of each user.
   / * creating Chunk Index with three field's namely Minimal ChunkID, FileHash and Binptr for each user */
10. Find the Minimal ChunkID of each file of Intricate Stringent Index
11. Update Minimal ChunkID in Chunk Index.
12. Update FileHash value in Chunk Index.
13. Link the BinPtr of Chunk Index to their respective Bin of each user in Backup Cloud storage Server.
14. Send the Chunk File index to Backup cloud Storage Server.
15. call DeduplicateCloudBackup( )

Here three indexes are created for all the user files as Intricate Index, Intricate Stringent Index, Chunk Index. The Intricate Index has three field's namely Fixed ChunkID, FileHash and Binptr. The Intricate Stringent Index has three field's namely Intricate Fixed ChunkID, FileHash and Binptr. The Chunk Index has three field's namely Minimal ChunkID, FileHash and Binptr. First, The Filehash value is found for all the files. Then the files for each user are divided into fixed sized chunks as fixed chunkID. Then each fixed chunkID is palced in the Intricate Index with their filehash value. Now each fixed chunkID is again divided into fixed sized chunks as Intricate Fixed ChunkID and is placed in the Intricate

Stringent Index along with the FileHash value. Now the minimal fixed chunkID of that Intricate Fixed ChunkID is found for all the files. The Intricate Fixed ChunkID with minimum hash value is choosen to be the minimal ChunkID for the chunk index. The minimal ChunkID is palced in the Chunk index along with the FileHash value with binptr pointing to the corresponding bin in the Backup Cloud storage server.

The minimal chunkID is found using Broder's theorem [16]. The purpose of finding the minimal ChunkID is that according to Broder's theorem, the probability that the two sets S1 and S2 have the same minimum hash element is the same as their Jaccard similarity coefficient [17]. In other words, if two files are highly similar they share many chunks and hence their minimum chunk ID is the same with high probability. The file hash is found by SHA-1 or MD5. The Binptr provides pointer to the corresponding bin. Each bin contains two fields as chunkID and the chunksize. The hash value of the chunk is named as ChunkID.

F. Cloud backup De-duplication module

This module performs the function of de-duplication detection by comparing the incoming Chunk file index with the backup node Chunk file index. It starts by checking the minimal ChunkID of chunk index of the Backup server with the minimal ChunkID of chunk index retrieved from intricate stringent chunker. If both the minimal ChunkID of chunk index are the same, then the file is a duplicate one. If the file is identified as duplicate, then it is not saved into the disk of Backup RAM. If the existing minimal ChunkID does not match with intricate stringent chunker, then it corresponding file is assumed as new file and unmatched chunks of the file are updated into backup node. So here the file is assumed to be duplicate if and only if both the intricate stringent chunker and Backup cloud storage server share the same minimal ChunkID in their chunk index thereby increasing the de-duplication efficiency by executing algorithm 5.

Algorithm 5: De-duplicate cloud back up ( )

Input: Chunk File Index for all User files

Output: Backing up the respective chunks.

1) Receive the Chunk File Index in three forms as Intricate Index, Intricate Stringent Index, Chunk Index for all User files
2) Declare the retrieved chunk file index as New Chunk File Index
3) Declare the chunk file index of Backup Cloud Storage Server Old Chunk File Index.
4) If (minimal Chunk ID of New Chunk File Index = = minimal Chunk ID of Old Chunk File Index && File Hash of New Chunk File Index= = File Hash of Old Chunk File Index) then
5) Report the detection of Duplicate File.
6) Create the pointer with the previous file
6) else
7) Extract the Files having unmatched minimal Chunk ID
8) Create a new bin with two fields Chunk ID, Chunk Size for the unmatched minimal ChunkID.
9) Update the fields ChunkID, Chunk Size of unmatched minimal Chunk ID in new bin
10) Update the Chunk Index of New Chunk File Index related to unmatched minimal Chunk ID
11) Update the Intricate Index of New Chunk File Index related to unmatched minimal Chunk ID
12) Update the Intricate Stringent Index of New Chunk File Index related to unmatched minimal Chunk ID
13) Save the new File of unmatched minimal Chunk ID into Back up Disk.
14) Update the data chunks and file recipes in to the Back Up disk
15) End

5. IMPLEMENTATION

We have implemented this by creating the Cloud storage Controller, Intricate Stringent Chunker and Backup Cloud Storage Server and multiple clients on WINDOWS platform. Any number of clients can be registered to the cloud server. The coding is done by using visual studio.Net and back end as Microsoft SQL server. The cloud server node is executed followed by the users' registration. All the users can have their individual username and password. They can upload any type of files. Our Intricate Stringent Chunking De-duplication is compared with the Extreme Binning file de-duplication. Our analysis is showing that our proposed system will have de-duplication efficiency based on the number of files being stored in the backup node. Our result analysis is shown in the Figure 6 and 7.

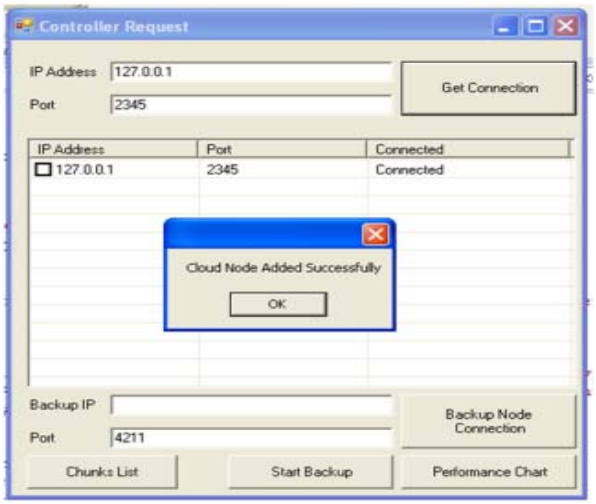


Figure-6. Registering clients to cloud storage controller CSC.

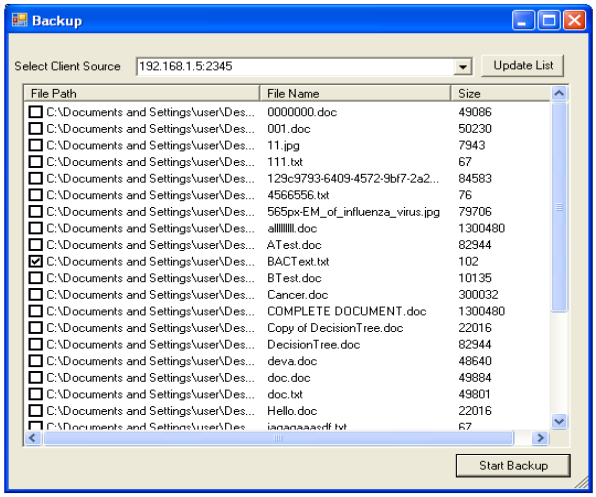


Figure-7. Making backup for cloud users.

The performance analysis of De-duplication File efficiency is analyzed by the number of files stored in the Backup Cloud Storage Server and it is shown in the Table-1 and Figure-8.

Table-1.

Number of files given for Backup	Number of files stored in ISC De-duplication	Number of files stored in Extreme binning
500	12	150
550	8	125
600	15	110
650	7	90
700	19	200
750	25	230
800	15	270
850	29	180
900	11	270
950	20	290
1000	35	320

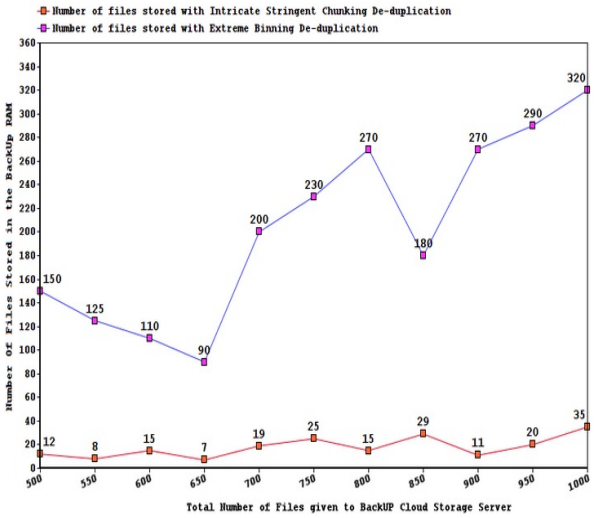


Figure-8. De-duplication file efficiency.

The performance analysis of De-duplication Time efficiency is analyzed by the time taken to store the de-duplicated file in the Backup Cloud Storage Server and it is shown in the Table-2 and Figure-9.

Table-2.

Number of files given for backup	De-duplication Time with ISC De-duplication (ms)	De-duplication time with extreme binning (ms)
500	9	95
550	15	130
600	13	170
650	19	110
700	24	85
750	12	105
800	17	135
850	22	156
900	14	175
950	12	125
1000	23	180

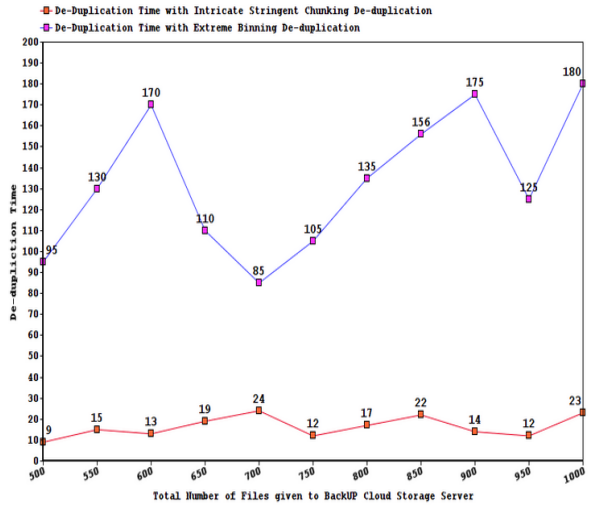


Figure-9. De-duplication time efficiency.

6. CONCLUSIONS

This paper describes a novel design namely Intricate Stringent Chunking File De-duplication that effectively removes duplication. It is highly advantageous to improve the private cloud backup storage efficiency by reducing the de-duplication time and the number of files to be backed up. Our future enhancement is to use chunk level de-duplication in the private cloud storage by overcoming the negative factors in its efficiency.

REFERENCES

[1] Jaehong Min, Daeyoung Yoon, and Youjip Won. 2011. Efficient De-duplication techniques in modern backup operation. IEEE Transactions on Computers. 60(6).

[2] Wei *et al.* Mad2: A scalable High-throughput exact de-duplication approach for network backup services, Mass Storage Systems and Technologies, IEEE / NASA Goddard Conference.

[3] Abe *et al.* Towards better integration of parallel file systems into cloud storage. In Cluster Computing Workshops and Posters (CLUSTER WORKSHOPS). IEEE International Conference.

[4] Bhagwat D., Eshghi K., Lillibridge M. 2009. Extreme binning: Scalable, parallel de-duplication for chunk-based file backup.

[5] Zhu B., Li K. and Patterson H. 2008. Avoiding the disk bottleneck in the data domain de-duplication file system. In: Proceedings of the 6th USENIX Conference on File and Storage Technologies, FAST'08, Berkeley, CA, USA. USENIX Association. pp. 18: 1-18: 14.

[6] P. Kulkarni, F. Douglis, J. La Voie and J. Tracey. 2004. Redundancy Elimination within Large Collections of Files. Proc. USENIX Ann.Technical Conf., General Track. pp. 59-72.

[7] B. Hong and D.D.E. Long. 2004. Duplicate Data Elimination in a San File System. Proc. 21st IEEE / 12th NASA Goddard Conf. Mass Storage Systems and Technologies (MSST), pp. 301-314.

[8] C. Dubnicki, L. Gryz *et al.* 2009. HYDR Astor: a Scalable Secondary Storage. In: Proceedings of the 7th USENIX Conference on File and Storage Technologies (FAST), San Francisco, CA, USA, Feb. 2009.

[9] C. Policroniades and I. Pratt. 2004. Alternatives for Detecting Redundancy in Storage Systems Data. Proc. Conf. USEXNIX '04, June.

[10] W.J. Bolosky *et al.* 2000. Single Instance Storage in Windows 2000. Proc. Fourth USENIX Windows Systems Symp. pp. 13-24.

[11] L.L. You, K.T. Pollack and D.D.E. Long. 2005. Deep Store: An Archival Storage System Architecture. Proc. Int'l Conf. Data Engineering (ICDE '05). pp. 804-8015.

[12] M. Lillibridge *et al*. 2009. Sparse Indexing: Large Scale, Inline Deduplication Using Sampling and Locality. Proc. Seventh USENIX Conf. File and Storage Technologies (FAST '09).

[13] D.R. Bobbarjung, S. Jagannathan and C. Dubnicki. 2006. Improving Duplicate Elimination in Storage Systems. ACM Trans. Storage. 2(4): 424-448.

[14] Zeng W, Zhao Y, Ou K and Song W. 2009. Research on cloud storage architecture and key technologies, ICIS '09: Proceedings of the second International Conference on Interaction Sciences. pp. 1044-1048.

[15] Policroniades C. and Pratt I. 2004. Alternatives for detecting redundancy in storage systems data, ATEC '04: Proceedings of the annual conference on USENIX Annual Technical Conference, A. Z. Broder. On the resemblance and containment of documents. In SEQUENCES '97: Proceedings of the Compression and Complexity of Sequences. 1997, pp. 21–29.

[16] P. Jaccard. 1901. Etude comparative de la distribution orale dans une portion des Alpes et des Jura. In Bulletin del la Soci´et´e Vaudoise des Sciences Naturelles. 37: 547-579.

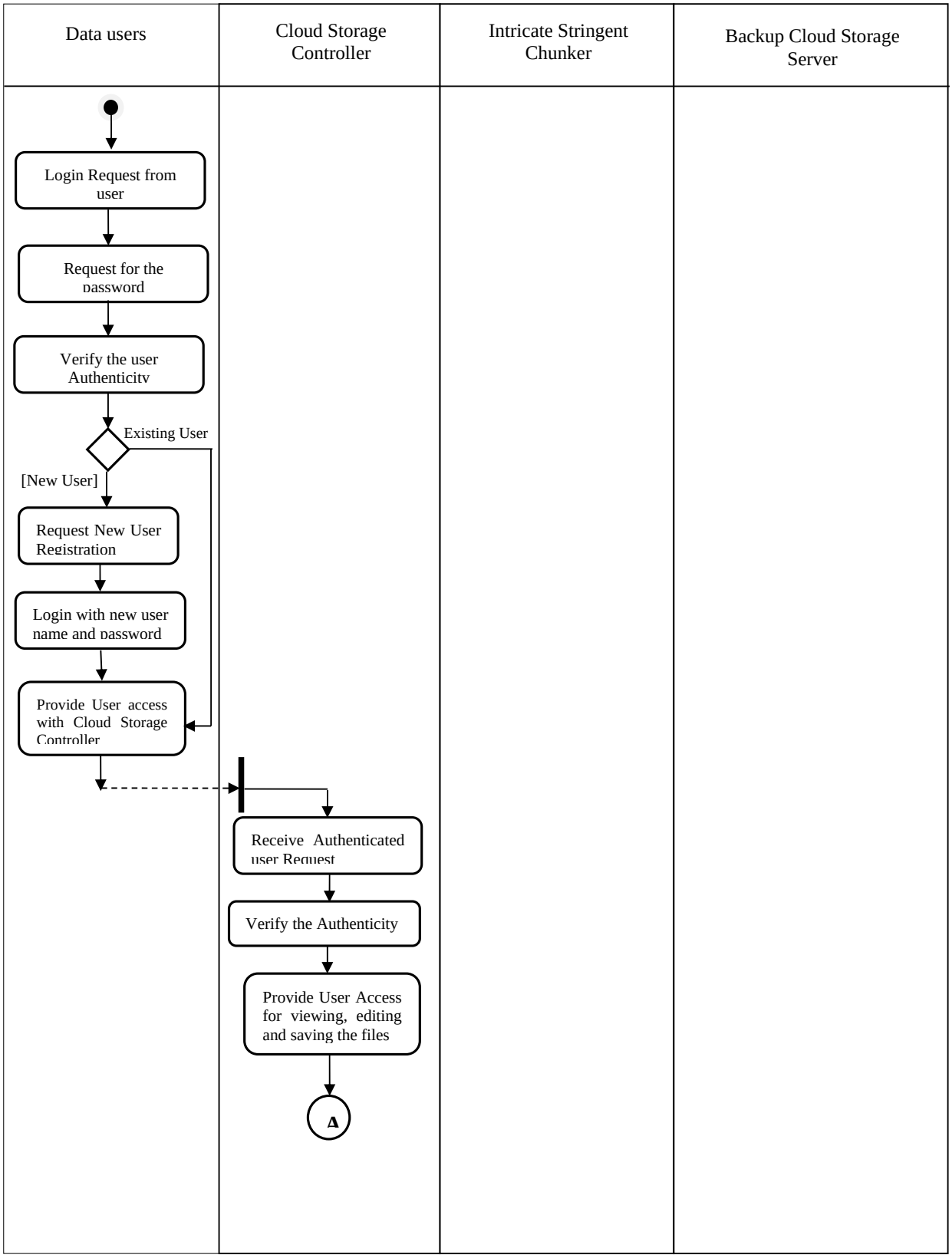
[17] G. Forman, K. Eshghi and S. Chiocchetti. 2005. Finding similar files in large document repositories. In KDD '05: Proceeding of the Eleventh ACM SIGKDD International Conference on Knowledge Discovery in Data Mining. pp. 394-400.

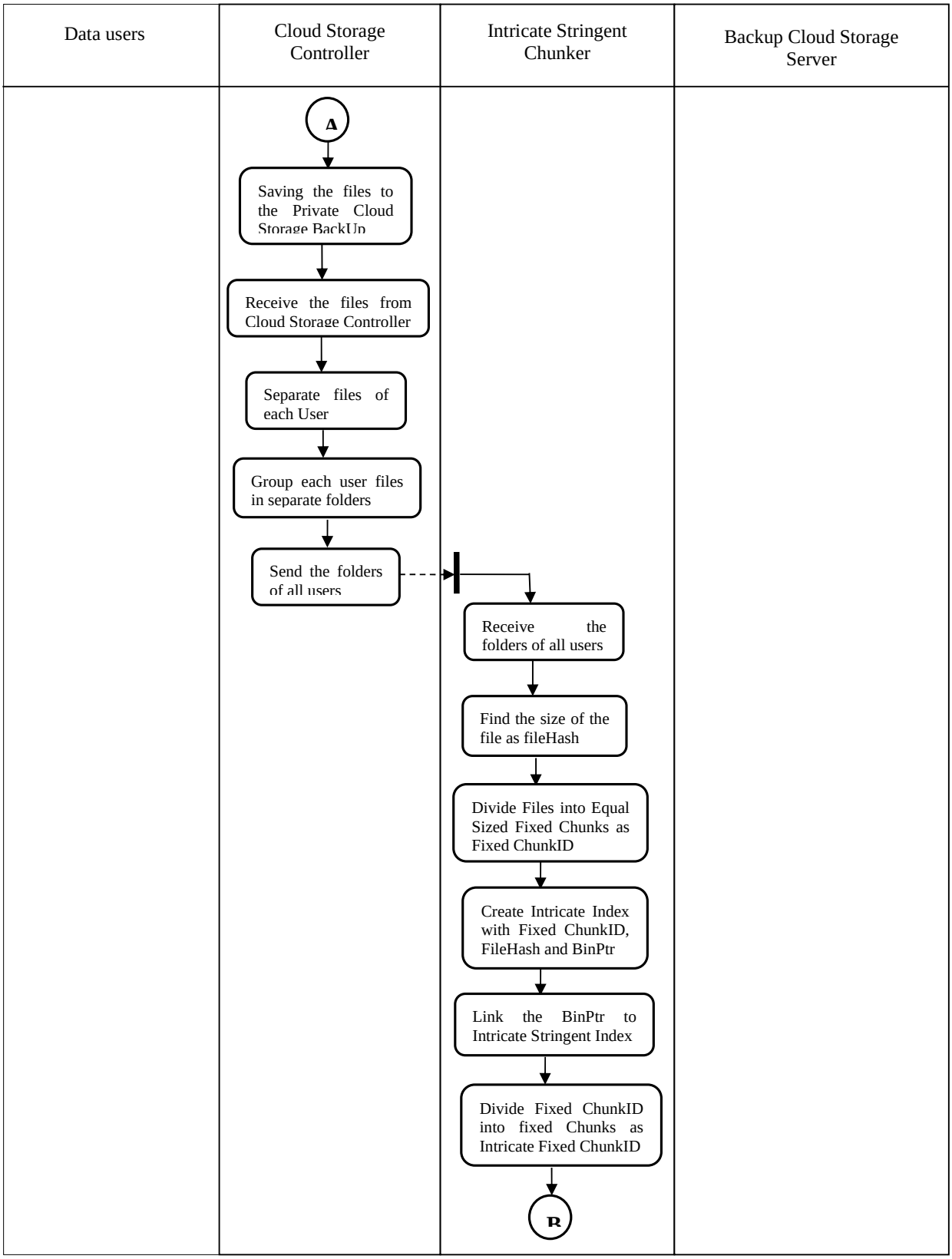
[18] Amazon Web Services LLC. 2009. Amazon Simple Storage Service. http://aws.amazon.com/s3/, 2009.

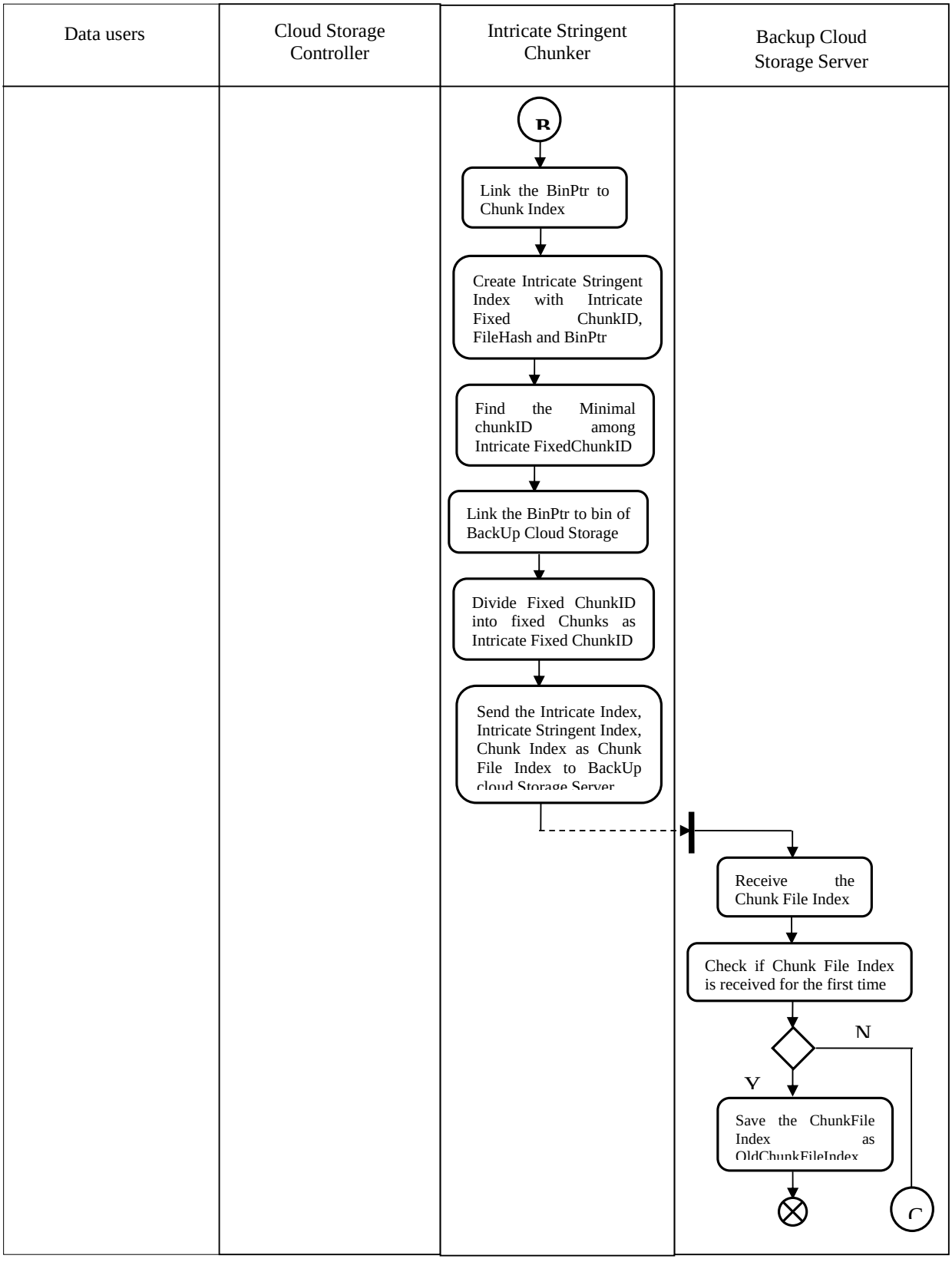
[19] Amazon's Elastic Block Storage. Elastic Block Storage, [Online] Available: http://aws.amazon.com/ebs/.

APPENDIX A

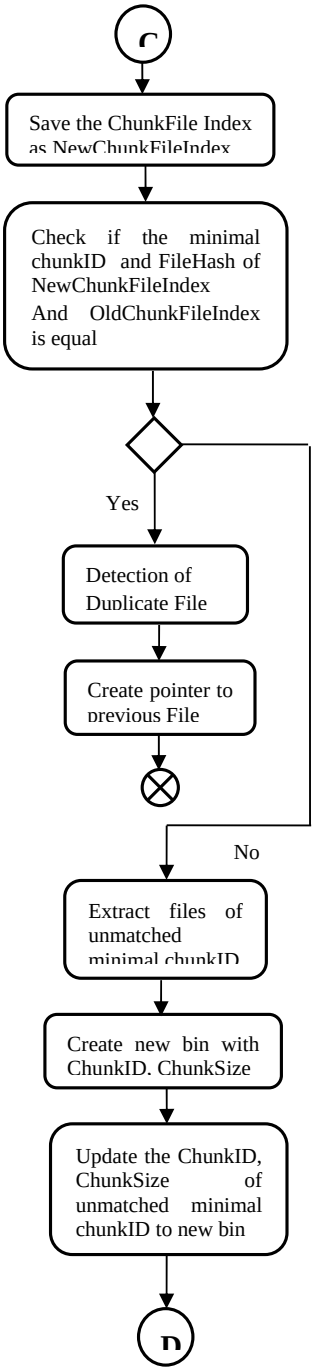
Activity diagram for the intricate stringent chunking file De-duplication:







Data Users	Cloud Storage Controller	Intricate Stringent Chunker	Backup Cloud Storage Server



Data Users	Cloud Storage Controller	Intricate Stringent Chunker	Backup Cloud Storage Server