CUCKOO HASHING BASED FINEGRAIN AUDITING IN CLOUD STORAGE

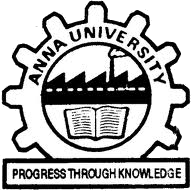
­ **CS8712 SOFTWARE DEVELOPMENT LABORTAORY**

*Submitted by*

**SRIDHARAN J**

**NIHIL KULASEKARAN L**

**SENTHAMIZHAN S**



**MADRAS INSTITUTE OF TECHNOLOGY, CHENNAI**

**ANNA UNIVERSITY, CHENNAI 600 044**

**OCTOBER 2017**

**ANNA UNIVERSITY: CHENNAI 600 025**

**BONAFIDE CERTIFICATE**

Certified that this project report forCS8712 Software Development Laboratory titled “CUCKOO HASHING BASED FINEGRAIN AUDITING IN CLOUD STORAGE” is a bonafide work done by SRIDHARAN J(2014503056),NIHIL KULASEKARAN L(2014503029) and SENTHAMIZHAN S(2014503050) under my supervision, in partial fulfilment for the award of the degree of Bachelor of Engineering in Computer Science and Engineering. Certified further, that to the best of my knowledge the work reported here in does not form part or full of any other thesis or dissertation on the basis of which a degree or award was conferred on an earlier occasion to this or any other candidate.

Laboratory in-charge Supervisor

Signature with Seal Signature with Seal

TABLE OF CONTENTS

|  |  |  |
| --- | --- | --- |
| SERIAL NUMBER | CONTENT | PAGE NUMBER |
| I | INTRODUCTION | 4 |
| II | LITERATURE SURVEY | 5 |
| III | PROPOSED WORK | 10 |
| IV | ALGORITHM | 14 |
| V | PROOF OF CUCKOO HASHING | 16 |
| VI | ARCHITECTURE DIAGRAM | 18 |
| VII | IMPLEMENTATION DETAILS AND RESULT | 18 |
| VIII | CONCLUSION | 20 |
| IX | REFERENCES | 20 |

**1 ) INTRODUCTION**

Today Cloud computing is growing as popular in terms of computing capability and storage. Computing capability in cloud came into existence after cloud storage. Millions of users are now saving their data in cloud environment. Cloud storage is a service model in which data is maintained, managed, backed up remotely and made available to users over a network. Cloud computing[1] provides service on three levels: Infrastructure (IaaS), Platforms (PaaS), Software (SaaS). The user has to rely on the cloud service provider for reliability and performance. Cloud usage is also associated with high data security and privacy risks related to data storage and management. Clouds can also be classified based on the type of owners as: Private Clouds, Public Clouds, Hybrid Clouds. Cloud data centers are awash in digital data, easily possessing petabytes and even exabytes of information, and the complexity of data management escalates in big data. There are lot of issues to be focused on the modern cloud storage which include security to data, performance measures and providing user-friendly environment. Data integrity has to be ensured and at the same time data attributes should be preserved.

It becomes important for the user to get the service he requests in a convenient manner. While ensuring the user-convenience the cloud provider must also look for saving his data. User must be aware of the current user-trends and requirements so that he must also balance the trade-off between performance and storage capacity. The Storage should be saved as much as possible in the service provider point of view as well as the client must be satisfied with the service he is getting. The problem of exhausting storage capacity in cloud servers is discussed here. Duplication of data causes more storage to be used for the same data that is stored again. There is chance that data saved by one user is saved by other users in cloud. The data that is stored may get duplicated in various forms such as a part of a user's data may get replicated again and again or another user may store the same data which is stored by some other user or may even by the same user again knowingly or un-knowingly.

Deduplication has become a key component in modern backup systems due to its demonstrated ability of improving storage efficiency. Depending on the data size, we can save lot of storage space in cloud if we find de-duplicated data. The data size can be dealt with in terms of chunks or entire files i.e., a file taken as an entire chunk. These have occurred again in the same data store and the cloud provider loses the advantage of more storage if he doesn’t solve this problem. This serious issue have been solved using certain methods with some level of accuracy but not have been entirely eliminated. Although it can’t be done to perfection many people have tried minimizing duplication in cloud storage. Data de-duplication is a form of compression in which we identify repeated chunks of information and store them once to save memory space in cloud environment.

Data is chopped into chunks of smaller data and de-duplication algorithm is applied. A unique way for re-collecting the chunk which have occurred earlier is found and is used. The first occurrence of the chunk must be stored optimally in the cloud so that the next occurrence can be easily matched with the first occurrence. Deduplication is mainly done through hashing. Cuckoo hashing[5] is a hashing scheme in computer programming for resolving hash collisions of values of hash functions in a table, with worst-case constant lookup time. A hash based method is implemented in this paper and performance is calculated. This problem is not the only problem that has to be taken care of. Along with de-duplicating cloud data, fine grained auditing must be done to preserve the attributes of the data.

Attributes are the properties of the data. When the data had been created, modified or deleted. If in the process the original information about the file which was uploaded by the user gets deleted then there is no meaning in removing duplicates at the cost of losing file information. The fine grained details should be preserved. Most of the Fine Grain Auditing method monitors the data access and ensures data integrity based on the information retrieved or modified. With the help of this method, it becomes easier to focus on security-relevant issues in cloud computing areas that are very important. De-duplication[3] sometimes may lead to high threat security issues. If a server tells a client that it doesn’t need to send the file reveals the fact that some other client has the same file. Data Reliability becomes a problem when only one copy of the file is shared by all the owners.

**2) LITERATURE SURVEY**

Cuckoo hashing was first described as simple and efficient dictionary with constant lookup time[4]. Cuckoo hashing usually is implemented with O(1) retrieval time and constant time for insertion. But cuckoo hashing suffers from infinite loops during insertion. To overcome infinite loops during data insertion an approach is proposed in which the data is selected from un-busy kicking-out routes[5]. Un-busy kicking-out data units are determined by the frequency it is accessed. For this they have used a counter for each value in the hash table. They have also implemented fine-grained locks in their system to improve the concurrency. Space overhead is incurred as every value uses a counter variable[5]. This method is called as Min Counter method as they have selected data unit based on the frequency of elements which is minimum. Cuckoo hashing is now being used in modern switches[6] to find the path to route in constant time. Cuckoo hashing is used in applications where there are more retrieval information is present rather than updating information because of O(1) guaranteed retrieval. Single Double Cuckoo Hashing [6] is other method in which each button in the table has the capacity of storing at most 2 values. Based on values, a bucket can be empty or party filled(single element) or filled completely(Double elements)[6]. A method for placing variable length elements is discussed. Using generic cuckoo hashing insertion method with few modifications to fit the variable length values inside cuckoo hashing table. The main drawback of this alternative is that now the memory width will not match that of the values for all the tables it stores, leading to memory waste.

De-duplication assisted of Cloud-of-Cloud(DAC)[7] supports the following features in client side, data deduplication, data distribution, performance evaluation and cost evaluation. Data deduplication takes of care of identifying duplicate data using SHA1 or MD5 hash value. Data distribution ensures that data is distributed among multiple servers. Performance and Cost evaluation modules take care of performance of service and cost metric associated with it. Distributes data among different cloud service providers(Cloud of Clouds), thus by ensuring the high availability of data in any point of time[7].

Only fixed length block level deduplication is used which reduces the probability in finding the duplicates. Data deduplication is special form of compression[8]. For identification of identical chucks a single pointer is sufficient whereas for chunks that are similar different approach has to be followed. For identifying similar chunks a cubic time complexity method with quadratic space[8] is proposed for storing and retrieving the information. Based on the information that is already present, new data and its pointer length are decided. This ensures dynamic way of producing block level deduplication. The Cloud Data Management Interface[9] (CDMI) defines the functional interface that can be used by applications to create, retrieve, update and delete data elements from the Cloud. Client, Front-end node, Database, Adaptor node and Storage node are the components used. File level duplication is adopted in this method, so deduplication probability for large files will fail.

Attribute based sharing is widely used in environments where data providers outsource their data to cloud service providers in encrypted format[10]. In this model, data provider then shares the data to users processing specific attributes. Data provider can confidentially provide data to users as data is encrypted and key is only known to data provider. It achieves the semantic security for data confidentiality. Based on the applications, data are grouped thus increasing the chance for data deduplication. In this application aware deduplication strategy[11], dedupe storage nodes are responsible for storing raw data. Director provides the logical separation among data. Clients data is separated using director and stored in dedupe storage node. Cloud storage is one of the solutions for limited mobile storage. Energy consumption plays a vital role in mobile devices. Whenever a new file is uploaded, Smart Deduplication for Mobile(SDM)[12] cloud storage systems chooses either file level or block level deduplication based on the learning system to achieve optimal data deduplication and high energy consumption. Based on the learning, SDM will choose whether to go for file or block level deduplication technique when a new file is to be uploaded to cloud. Bloom Filter technique is used for hashing which reduces the time that takes to load big databases. Though this method is useful of mobile systems, but it requires additional processing time to decide which type of deduplication technique to be used.

In content defined chunking, files are split into fixed and variable length blocks[13]. For variable length blocks, cut points are decided based on features of file. Computational complexity of deciding cut points are high. A new method called Rapid Asymmetric Maximum (RAM)[13] based on asymmetric chunking algorithm was discussed. Instead of using hashes, RAM uses bytes value to declare the cut points. The high chunking throughput comes at the cost of higher chunk variance.

Data deduplication can also be applied over encrypted data and it ensures the security of the data that is being stored in cloud. Proof of Ownership(PoW) is an interactive algorithm in which data owner is required to prove the ownership of data. Different approaches for implementing data deduplication over encrypted data are discussed in [14]. A sliding block algorithm in which data deduplication is found by sliding the pointer ¼ and ½ times left and right is said to improve data deduplication rate by 6.5%[15]. Using weak hash function, data chunks are compared, if they match, file in the left of ¼ blocks will be searched, if they too match then ½ of left block is searched, otherwise it will exist and save data value[15]. Additional time is required to check for each data around the new de-duplicated data. The time cost of achieving this algorithm depends on division of chunk size which increases with increase in chunk size.

The first privacy-preserving deduplication protocol capable of efficient ownership management in fog computing. It achieves fine-grained access control by introducing user-level key management and update mechanisms[16]. Fog provides the storage service at network edge and acts as a proxy. This Requires high bandwidth and suffers from storage overheads. In-line deduplication is done at server side where users uploaded data will be checked while it is being moved to server. Other ways of classifying deduplication algorithm is based on the location of deduplication data. If deduplication happens at the place where data is created it is called source deduplication. When de-duplication happens after it is transmitted it is called target deduplication.

Global deduplication is other deduplication strategy in which deduplication is performed across multiple devices to save the data space. Global deduplication is often followed by Cloud Service Providers to save the storage across users. A separate copy of entire file will never be saved. In post processing deduplication algorithm, users uploaded file will be saved in server and then subjected deduplication algorithm. Using cache knowledge[17] in in-line deduplication, deduplication procedure time can be decreased. History aware In-line Deduplication algorithm (HIDC)[17] proposes how encoding and decoding can be applied over compressed data. Data deduplication at virtual machine level can be handled. Improved K-means algorithm is used to find the deduplication at virtual machine images and eliminate redundant copies of the file[18]. This method uses preprocessing technique for processing stored VM images and fingerprints to improve the performance during deduplication procedure. Valid only with only with runtime and backup storage systems. Doesn’t scale up with many-to-one runtime and backup storage systems.

A new method for uploading media content alone to user is discussed. Media content is first encrypted and then checked for deduplication. An agency server which checks for offline attacks is active[19]. Scalable Video Content(SVC) enables different versions of the video to be stored in a single video thus saving space. Method for media content alone works with this approach[19]. Requires a separate agency server to monitor offline attacks. Inline processing of data deduplication is carried out which reduces the bandwidth required to upload the data to the server if there is already de duplicated available[19]. Requires time for processing in client side. Degrades the performance of client computer when a large file needs to be uploaded. Key Generation Center (KGC) ,Cloud Service Provider(CSP) ,Authorized Party(AP) and Data Holders[20] play vital role in maintaining de-duplicated data in cloud. AP is trusted by all other parties for sharing data. Certificate issuance system parameter generation is handled by KGC. CSP offers to provide storage space. Authorized party is the delegate for data holder and also maintains data integrity across cloud storage. Success ratio depends on the keys generated by KGC.

Graph is used with constraints as edges and data value as nodes. So connected nodes logically form a file, thus increasing the probability of finding deduplication and link the data correctly[21].Fragmentation should enforce the protection by dividing sensitive information into separate fragments. This method Requires extra memory for storing the data in graph format. An efficient inner-product predicate encryption system(indistinguishable under chosen plain-text attack for attribute-hiding). It uses privacy preserving predicate encryption with fine-grained searchable capability for cloud storage[21].

A Third Party Auditor (TPA) is used to validate the security and integrity of files in the cloud. Normal message flow will happen between client and Cloud Service Provider(CSP)[22]. Unforgeable metadata generated from individual data blocks is called homomorphic authenticators[22]. Using the homomorphic authenticators, we can ensure data integrity of file is maintained. Failure of TPA will hugely impact the security and data integrity of all the clients available. An improved NTRU(Number Theory Research Unit) cryptosystem is used to present a secure and verifiable access control and to protect the outsourced big data stored in a cloud[23]. An adequate threshold “t” needs to be selected by the data owner before deployment because an unsuitable threshold would decrease the security of the data if “t” is too small or have a harmful effect on the flexibility of access control if “t” is too large. Several works related to data deduplication are discussed[23]. Also data owner can dynamically update the data policy that is defined for his data. Frequent updates will is an overhead to the system.

Private cloud is used for encrypting the incoming data from client, whereas public cloud stores the data[24]. Each data is associated with tag and label and data is encrypted with the set of attributes. Private cloud will save the tag. A separate service provider is required to determine the set of attributes used for encrypting. Clients, dedupe storage nodes and director are used for maintaining deduplication. Two tired data routing is used which does the file-level application aware routing decision in director and the super chunk level similarity aware data routing in clients. Requires application knowledge(extension) of the file.

**3) Proposed Work:**

**Data de-duplication** has been a problem in Cloud storages where same or different users upload the same data pattern to their storage. Duplication of the same data again and again just in different names of storage space or different file names is simply waste of valuable storage space. This project intends to provide a suitable method to de-duplicate cloud storage such that the performance is enhanced and storage space is saved.

This model is initially done in JAVA platform on Windows Operating System. Since JAVA has better support for file level handling and supports modifying the attributes of the files and folders. JAVA provides lots of other advantages as like compatibility in code. It can be run on many platforms so creating the model in java makes it compatible to Linux based distribution too. To accomplish the task a common data dump is required. Since all the users may provide duplicated data there arises a need to store all the data together. A separate folder is visualized as the drive in cloud where a particular user saves his files. The user stores all his file in this folder and during the process the data from all the files irrespective of the file type should be stored centrally so that the data can be de-duplicated. The de-duplication part is focused inside the folder alone. That is once the user uploads all the files to his folder the de-dup process is run once for the entire file bundle and duplicated data is removed. The process supports all types of files and is entirely done on file-level. Since all type of files can be treated as binary all files can be de-duplicated using this process. The process is entirely done for each files whereas the actual de-dup algorithm runs for each chunk.

De-duplication can be done both in file level as well as on a chunk level basis. Here we do it on a chunk level basis as it provides better de-duplication as the level becomes finer and finer. Fixed sized chunks of data are used as data de-duplication units rather than going for the entire file. As this process goes on the data is read from the user file and processed while the user file is completely deleted as it is no longer required. A retrieval mechanism is also required. As a user uploads a file of 24MB, the file is divided into 6 if the chunk size is of 4MB and the rest of the process is done. The entire chunks are stored in a centralized file as it is present in the user file. This file is a large file as it stores all the data of the user files. The data in this file should not be duplicates. That is this file store should only contain unique data so that the process is achieved. In order to achieve that hashing based scheme is adopted. The duplicated chunks are to be identified among all the chunks. The chunks are to be matched with the fore coming chunks so that we may know which part is duplicated. But comparing large chunks of say size 4MB every time is a tedious process and is not encouraged given the performance should be higher and not much time can be placed in the de-duplication process as user-satisfaction is more important. So the chunks are hashed and a 16 bit hash is generated for each chunk of the specified size. A good key generation algorithm is to be chosen so that the keys do not collide much. The entire process revolves around hashing then. As the hash is generated a hash map needs to be maintained so that the hashes generated by the hash function are stored separately. The hash table must be so efficient and use optimal space. Cuckoo hashing is preferred in this case as it provides O(1) time complexity. As each hash is generated by the function it is to be compared with the previous value right away. A simple look up in the hash table for the hash value generated for the current chunk will tell if the data is already available or not. Thus as the user uploads his files and the de-duplication process is on progress the data is hashed and then compared with previous values in the hash map constructed in O(1) time complexity and the de-duped data is decided. So as the user files are getting de-duped by the process the original data is being stored in the central file store where as the hashed file is store in the corresponding file with a different extension. The central file store doesn’t have any duplicates and is common for all the user files whereas the corresponding file which is created to store all the hash values of the chunks in that file does have duplicate hash values stored in it if duplicates exist in the file.

Choosing the optimal chunk size is a necessity as the de-duplication process entirely depends on the this. The granularity of the chunk decides how efficient this algorithm is going to be. If the chunk size is chosen to be very small then the de-dup efficiency will be higher as for each smaller chunk size definitely a large part of the original data may be found duplicated. But taking large chunk size may or may not give better results as compared to finer chunks. The problem of varying chunk sizes should also be dealt with as it can’t be said sure every time that the duplicate chunk exists from a starting point and a ending point in the file. Two chunks may be different in the way comparing the entire data but may have over-lapping data which may not be getting detected if the proper size consideration for that particular chunk is being met with.

The chunks should be recognized as belonging to each user file. This should be done else all the chunks will be stored in a centralized data store with no way to identify the file which the chunk belongs to. So a separate file for storing the contents of the file is required. But this would be again nothing and provide same results as the original file. So the hashed data is stored in the corresponding files after running the de-dup process. The process of checking the hash with the previous values of hash map is only for storing non-duplicated data in the central store whereas each corresponding file must store all the hash values of the chunk in that file. During File Retrieval it becomes necessary that we know what data is in the user file. So the chunk hashes are to be stored in the same file name with a different file format so that the original data doesn’t look tampered. There is also a need for compression of data which can be done using certain libraries.

As the centralized data store is being used to store data there is a need to know where the chunks are stored in the central file. The hash is stored in the separate file corresponding to the original file. But after this during file retrieval there is no way to re-construct the original data. There has to be some way to retrieve the original data from the hash value. But the original data is in the central file and the hashes are in their corresponding files. So there has to be a mapping between the original data and the hash values of them. The mapping can be done using a hash map in java with the necessary parameters taken into consideration. The mapping should be done in such a way that the position of the data chunk in the central file along with the offsets required to locate it in the big file are stored along with the hash value. Then during the File Retrieval process the original file can be again retrieved using the mapped values. The mapped values are to be stored in a separate file so that the positions of the unique data chunks are easily accessed. The file has to store objects as it can be seen the hash values stored as objects are easy to access rather than storing them as text files and getting accessing issues within the file. During Retrieval the each hash value is taken and the corresponding chunk’s location in the central file is identified from the object file and the data is written to the new original user file.

Fine-Grained Auditing is an important concept to be dealt with. As long this process is being done there is also a concern for the original user file’s properties. A user may have created it in some date and would have given specific permissions to the file and may even have modified it at a later time. But storing the entire data in central file doesn’t mind about any of the properties above mentioned and only works on raw data not store any information about the file attributes. A mechanism is needed to retrieve the attributes of the original files and apply them onto the new user files that are created during the time of file retrieval. JAVA provides special classes to work with the file attributes. The file attributes can be altered with the methods. The files with the hash values created in the place of each original user file can have the properties of the original user files written into them and then they can also be copied from them into the new user files at the time of file retrieval thereby preserving the user file properties.

The obvious approach will be hashing. The hashing technique to be used is important. It must be highly efficient and optimal as the de-duplication part should not be the critical process consuming much time. A hash method is chosen and proper enhancements are done to make it more efficient. The entire chunk cannot be put into the hash tables, so we use a suitable hash function to get the hash value of the chunk and the value is then put into the hash table. Likewise the duplicated chunk can be identified.

As the duplicated data chunk is identified, the original data has to be stored somewhere. Somewhere where all the users who had stored the same duplicated data can access it. Therefore a centralized data store is used where all the data either duplicated or non-duplicated are put in. The data in the centralized data store is not categorized or classified, its simply raw data of all users. Whenever an user requests his file the centralized data store is referenced and his data alone is given to him. This is done using a separate method or a file where each user’s data and their position in the central file is stored along with the hash value to identify the data. Thus data retrieval is done using this information. The hash values then generated using the hash function are then to be directly checked in the map structure in JAVA. If the hash value of the data chunk is already available in the hash tables then the user’s data is not copied onto the central store again, instead just the previous reference made to the same data is stored, thereby saving lot of storage space. Thus in case of duplication there is no extra space occupancy required as the old data has been pointed out.

A suitable technique for compression of data is also to be looked after as some drive space can be saved. Thus the data stored in the central space is compressed and is not stored as the same data as it is uploaded by the user. Sometimes the chunk size may be an issue as it may not meet with the complete size requirements. A typical chunk may not be of the perfect chunk size say 2.3 MB which is very smaller than 4 MB, which is the fixed size. This issue should be taken care of Fine-grained auditing is also a major concern here and so we also look to maintain the user level of perception of his file. The file details corresponding to each user should be preserved as though data may get de-duplicated each file is different in its permissions, ownership, creation time etc., Therefore a metadata preserving mechanism is also to be dealt with.

**4) ALGORITHM**

For identifying duplicates in file we use Chunk Hash algorithm. To retrieve file we use Retrieve file algorithm. We discuss both the algorithm using mathematical notations.

**I)CHUNKHASH ALGORITHM**

Let fr be the data read from the file. M represents the hash map object. si, di, sti, li, represent the hash string of the chunk i, di represents the file where the chunk i is being stored, sti be the starting position in the file, li represent the offset of the chunk i. N is the folder name where the files to be de-duped are stored. n is the size of file divided by 4 i.e. the number of chunks. fd represent every file inside the folder. Cf represents the chunk of 4MB size. Hf is the hash function used (MD5 Hash). hf is the hash value generated for the chunk Cf.

fr read(dedupe.txt)

x, M(x) exists, M convert ( fr )

x, M(x) does not exist,

M Create Hash Map(si,di,sti,li)

N Get Folder Name

fd Make src file

Read

hf Hf ( Cf )

hf M write hf to src file

( hf M ) M hf , write hf to src file

**II)RETRIEVE FILE ALGORITHM**

Let M represent the hash map which contains the hash values generated for each chunk. f is the file read. The JSON file f is read and then the file in which the chunk is present in is accessed. The hash key associated with each chunk is hx. The src file is read into fr and the output user original file is written into fw. ­  M(hx) is the mapped pairs in the hash map for the hash value hx. The corresponding attributes stored in the hash map as a pair with the hash values are represented as di the file number, si the starting index and li the offset of the chunk i. dr represents the dedupe file and attrr  and attrw represents the attributes of the src file and the user file respectively.

f read(“dedupe.txt”)

Convert f into map M

Read input Inp

fw  write original file

fr read src file

read M(hx)

M(hx) read( dr, si, li)

fr read dedupe dr file

write ch(sx, sx+lx) to fw

write attrr to attrw

**Time Complexity**

* Cuckoo Hash Recollection
* uses two hash functions h1,h2:{0,1}∗→[r] to map items to slots
* lookups This is O(1) worst-case
* Deletions are O(1) in the worst-case.
* Insert(x)
  + Put x in position h1(x); if that slot is taken, eject the occupying item to its 'other position', recurring until we find an empty slot or failing if we try to relocate any element xx more than once.

* + For converting a file into it’s corresponding “.src” format, complexity is O(size of file)=>O(n). For compressing a file it will take O(n) time.
    - Let “X” be the number of character to be read from file. H(X) is the time taken to read entire file. One character can be read at 1 unit of time thus reading X character takes X units of time which is H(X)=O(X).
  + For converting an object to its JSON representation time complexity is O(number of attributes in class) => O(n)
  + Overall time complexity =>O(1+n+n+n)=> O(n) which is same as reading the file.

**Space Complexity**

* Takes extra memory to store “.src” which is in order of bytes.
* Size is exactly O(ceil(size of file/(48\*4MB))) which is negligible as this method guarantees to save space.

**5) PROOF FOR CUCKOO HASHING**

Consider the following experiment: insert nitems into a cuckoo hash table.

Let tI(n)*tI(n)* be the time to insert the last element. We aim to show that

Eh[tI(n)] = O(1) => *(1)*

Note here that while the probability is over *h*, this is averaging over all n insertions.

Framing the analysis

The Cuckoo Graph

To assist us in reasoning about displacements during insertion, we define *the cuckoo* graphG whose vertices correspond to slots in the table, and whose edges correspond to items to be inserted; i.e. V=[r], E={ex=h1(x),h2(x)/ x € S}, where S is the set of items.

We also need to define the concept of *the bucket of xx*, which we denote B(x)*B(x)*; this is the set of all positions reachable in the Cuckoo Graph starting from h1(x)*h1(x)* or h2(x)*h2(x)*; said differently, B(x) is the component of G with edge ex.

Notes

* Though we may think of the set of items S as fixed, the values of hash function are random variables over [r], and so this also makes G a random variable (i.e. a random graph). So to is B(x) for any item x.
  + If G encodes a table, each component of G will never have fewer vertices than edges. Further, if the addition of edge ex would violate this property, then the addition of x would cause a complete rehash of the table.

The following lemma justifies these definitions

To insert a new item x, we must shuffle, in the worst case, the elements in all the slots B(x) (before the insert succeeds or we abort). Thus

tI(x)≤|B(x)| =>(2)

Thus it will suffice, for (1) to show that

E|B(x)|≤O(1) =>(3)

Technical Details

First note that

E|B(x)|=∑y ∈ S Pr [ey ∈ B(x)]

=n Pr[ ey ∈ B(x)]

Thus if we can show that ∀x≠y,

Pr[ey ∈ B(x)]=O(1/r) =>(4)

then we immediately establish [(3)](http://www.cs.toronto.edu/~wgeorge/csc265/2013/10/17/tutorial-5-cuckoo-hashing.html#mjx-eqn-eqgoal_restated). [(4)](http://www.cs.toronto.edu/~wgeorge/csc265/2013/10/17/tutorial-5-cuckoo-hashing.html#mjx-eqn-eqlemma_2) is easy consequence of the following lemma:

What remains are technical details. We work them out here for completeness, trying to break down the reasoning in such a way that each line follows obviously from the proceeding. The only fact about G we use is that each edge is drawn independently; the only fact about probabilities we use repeatedly is the union bound.

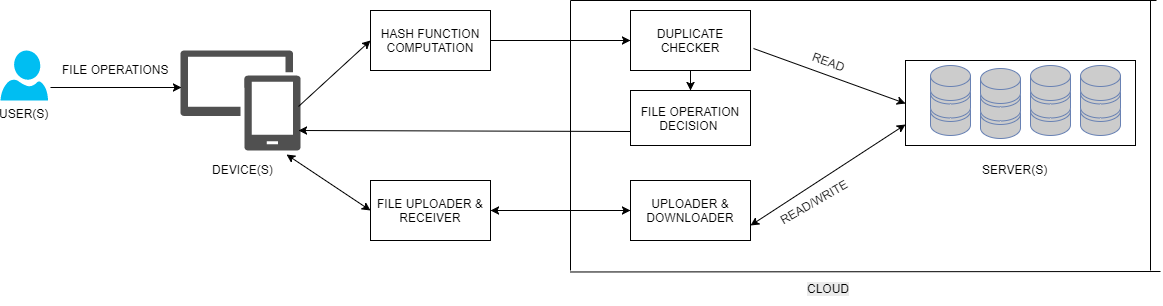
Proof of (4)

ey ∈ B(x)⟺∃h1(y)→h2(x) path

⟺ Thus we have

Pr[ ey ∈ B(x)] ≤ ∑ℓ≥1c−ℓ/r = O(1/r)

**6) Architecture Diagram :**

****

**Figure 1. Architecture diagram for proposed system.**

**7) Implementation Details and Result**

Project is being implemented in Java. As in Figure1, User can give the folder name to perform deduplication. All the subfolders in the given folder will also be de-duplicated. After deduplication cuckoo hash map object will be converted to JSON format. For converting object to JSON format we have used google GSON library which is the one of the most efficient library for JavaScript Object Notation(JSON). It will be compressed and stored as text file. For retrieving the file back, user will give the file name he wants, once corresponding “.src” file is found, original file will be formed back and provided to user. From the “.src” the original file’s extensions will be found using user defined attribute. Then corresponding file will be formed and retrieval procedure starts. File Retrieval implementation is also successful which is also linear in time. Metadata information like last accessed time, last modified time, created time, read/write permissions and owner of the file are preserved. is maintained and included in file thus making it possible retrieve files correctly.

We tested the implementation using the dataset provided in lemurproject.org/clueweb12/ClueWeb12-RemoveduplicateRecords.php. When unzipped the given dataset has 4 text files, each ranging around 20GB of size. We tested our deduplication algorithm on these files. We copied these files into a Test folder. Before deduplication test folder size is 81.4 GB. After applying deduplication procedure to Test folder, disk space is 18.84 GB. Thus the percentage of space that is consumed is given as

% of space consumed =

So percentage of space that is consumed is given as 23.14%. The amount of space that is saved is given as

% of space saved =)

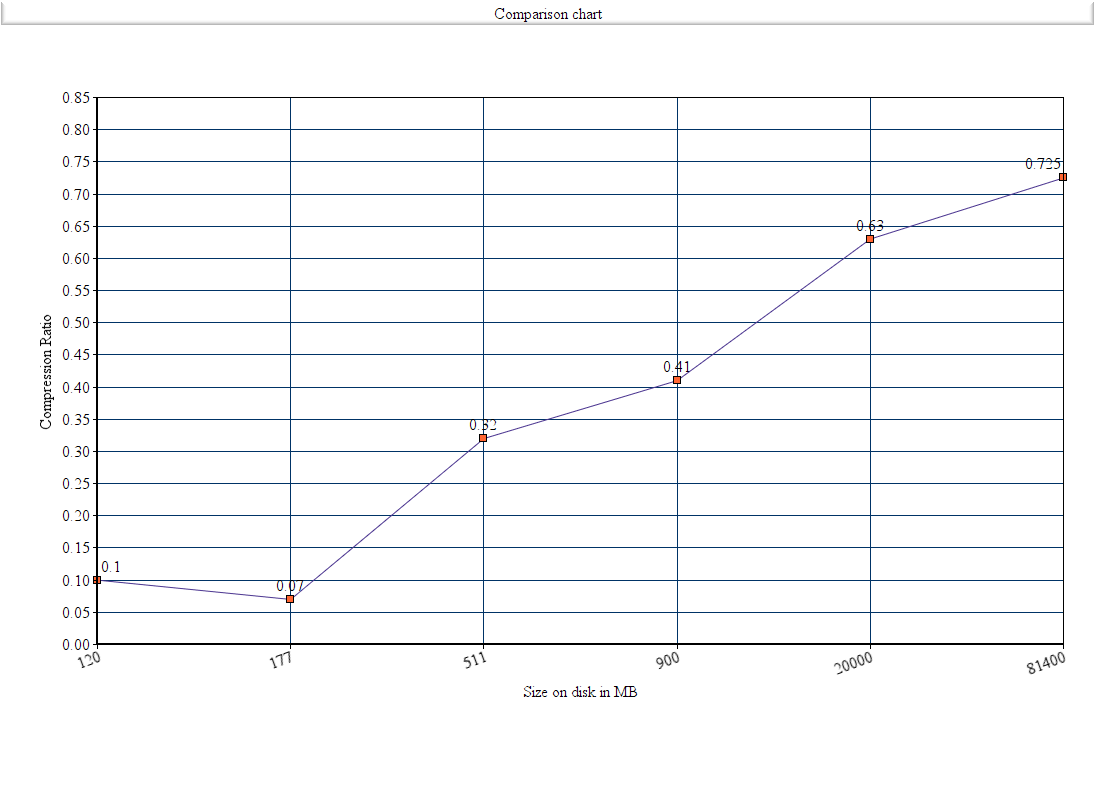


Figure 2. Comparison chart for different data sets.

The given dataset saves space around 76.86% which is very much higher that other deduplication algorithms used previously. Also the proposed system uses Cuckoo hashing to retrieve data which makes it efficient in terms of time. Time complexity described above shows that deduplication algorithm can be applied linearly to store and retrieve data. Figure 2 shows the result obtained and plotted in graphs for different types of data set. Higher the compression ratio, more is the data saved.

**8) CONCLUSION**

Thus the proposed system will save data storage space to a great extent. Further system will be extended to Client-Server architecture implemented as a simulation for cloud. Deduplication across users will be applied. Ensuring integrity of the file is important and it has to be accomplished. Security of the data has to be ensured by validating each command user gives for upload or retrieval as data is common for all users.

**9) REFERENCES**

[1] S. Patidar, D. Rane and P. Jain, "A Survey Paper on Cloud Computing," *2012 Second International Conference on Advanced Computing & Communication Technologies*, Rohtak, Haryana, 2012, pp. 394-398.

[2] Babaso D. Aldar, et al., "*A Survey on Secure Deduplication of Data in Cloud Storage*", International Journal of Innovations in Engineering and Technology (IJIET), Volume 6 Issue 1 October 2015.

[3] Meghana Vijay Kakde et al, "*Survey Paper on Deduplication Data and Secure Auditing in Cloud,*" International Journal of Computer Science and Information Technologies (IJCSIT), Vol.7 (1), 2016, pp 94-95.

[4] R. Pagh et at., “*Cuckoo hashing,*” in Springer Berlin Heidelberg, 2001.

[5] Y. Sun et al., "*A Collision-Mitigation Cuckoo Hashing Scheme for Large-Scale Storage Systems*," in IEEE Transactions on Parallel and Distributed Systems, vol. 28, no. 3, pp. 619-632, March 1 2017.

[6] G. Levy et al., “*Flexible Packet Matching with Single Double Cuckoo Hash*” in IEEE Communications Magazine, Vol. 55, No. 6, pp. 212-217, May 2017.

[7] Suzhen Wu et al., “ *DAC: Improving storage availability with Deduplication-Assisted Cloud-of- Clouds*”, Future Generation Computer Systems, Vol. 74, pp. 190-198, Sep 2017.

[8] M. Hirsch et al., “*Optimal partitioning of data chunks in deduplication systems*”, Discrete Applied Mathematics, Vol. 212, pp. 104-114, Oct 2016.

[9] Xiao-Long et al., “*A file-deduplicated private cloud storage service with CDMI standard*”, Computer Standards; Interfaces, Vol. 44, pp. 18-27, Feb 2016.

[10] H. Cui et al., “*Attribute-Based Storage Supporting Secure Deduplication of Encrypted Data in Cloud*” in IEEE Transactions on Big Data , Vol. PP, No. 99, pp.1-1, Jan 2017.

[11] Y. Fu et al., “*Application-Aware Big Data Deduplication in Cloud Environment*”, in IEEE Transactions on Cloud Computing  , Vol. PP, No.99, May 2017.

[12] Ryan N.S. et al., “ *SDM: Smart deduplication for mobile cloud storage*”, Journal in Future Generation Computer Systems, Vol. 70, pp. 64-73, May 2017.

[13] Ryan N.S et al., “*A new content-defined chunking algorithm for data deduplication in cloud storage*”, Future Generation Computer Systems, Vol. 71, pp. 145-156, June 2017.

[14] K. Akhila et al., “*A Study on Deduplication Techniques over Encrypted Data*”, Procedia Computer Science, Vol. 87, pp. 38-43, May 2016.

[15] GuiPing Wang et al., “*A sliding blocking algorithm with backtracking sub-blocks for duplicate data detection*” Expert Systems with Applications, Vol. 41, pp. 2415-2423, Apr 2014.

[16] DongyoungKoo et al., “*Privacy-preserving deduplication of encrypted data with dynamic ownership management in fog computing*,” in Future Generation Computer Systems, January 2017.

[17] R. A. Fegade et al., “*Cloud iDedup: History aware in-line Deduplication for cloud storage to reduce fragmentation by utilizing Cache Knowledge*”, 2016 International Conference on Computing, Analytics and Security Trends (CAST), Pune, pp. 244-249, May 2017.

[18] Jiwei Xu et al., “*Clustering-based acceleration for virtual machine image deduplication in the cloud environment*”, Journal of Systems and Software, Vol. 121, pp. 144-156, Nov 2016.

[19] Y. Zheng et al., “*Toward Encrypted Cloud Media Center With Secure Deduplication*”, in IEEE Transactions on Multimedia, Vol. 19, No. 2, pp. 251-265, Feb 2017.

[20] Z. Yan et al., “*Heterogeneous Data Storage Management with Deduplication in Cloud Computing*”, in IEEE Transactions on Big Data , Vol. PP, No.99, pp.1-1, May 2017.

[21] S. De Capitani et al., “*Fragmentation in Presence of Data Dependencies*”, in IEEE Transactions on Dependable and Secure Computing, Vol. 11, No. 6, pp. 510-523, Jan 2014.

[22] Q. Wang et al., “*Enabling Public Auditability and Data Dynamics for Storage Security in Cloud Computing*”, in IEEE Transactions on Parallel and Distributed Systems, Vol. 22, No. 5, pp. 847-859, May 2011.

[23] C. Hu et al., “*A Secure And Verifiable Access Control Scheme For Big Data Storage In Clouds*”, IEEE Transactions, Vol. PP, No. 99, pp.1-1, Feb 2017.

[24] Wang, X et al., “*Efficient privacy preserving predicate encryption with fine-grained searchable capability for Cloud storage*”, Electrical Engineering, Vol. 56, pp.871- 883, Nov 2016.