

PRIVACY PRESERVATION ALGORITHM USING EFFECTIVE DATA LOOKUP ORGANIZATION FOR STORAGE CLOUDS

Amar More¹ and Sarang Joshi²

¹Department of Computer Engineering, Pune Institute of Computer Technology,
Maharashtra, India
amarmore2006@gmail.com

² Department of Computer Engineering, Pune Institute of Computer Technology,
Maharashtra, India
sarang.joshi2002@gmail.com

ABSTRACT

In the era of cloud computing, many cloud service providers like Amazon, Microsoft, Google, etc are offering cloud storage as a service. We can migrate our data to the storage offered by them and can retrieve it back at any point of time or can share it with other people by making it available publicly through Internet. Since the size of the data stored in storage clouds could be very huge, searching and retrieving a desired data by maintaining its privacy is one of the important issues. In this paper, we present a new data storage organization for storage clouds which would be helpful for faster data lookup through privacy preservation.

KEYWORDS

Cloud Computing, Storage Clouds, Storage Organization, Data Privacy

1. INTRODUCTION

Storage clouds allow users to upload the data on their servers and offer high availability, security and reliability services. At its most basic level, a cloud storage system needs just one data server connected to the Internet. A client (e.g., a computer user subscribing to a cloud storage service) sends copies of files over the Internet to the data server, which then records the information. When the client wishes to retrieve the information, he or she accesses the data server through a Web-based interface. The server then either sends the files back to the client or allows the client to access and manipulate the files on the server itself. Amazon S3 [1], Microsoft Windows Azure Blob Storage [2], Nivranix [6], EMC Atmos [4], Mezeo [7], Google Storage [3], Rackspace Cloud Files [8], and Eucalyptus[5] are some of the cloud storage providers.

Since the data stored in cloud storage could be very large, retrieving a data to which a user is authorized to is a major challenge. Even though the client can specify the list of people who are authorized to access the data, all the cloud storage providers provide direct access to the private data (the data owned and stored by the client) only. But even if the client is authorized to access the data stored by some other client, access to such data is not provided directly.

If a client wants to access a data which is not a private data but is authorized to access it, then he or she either has to know the detailed information like name, location of such data or the data access URL should be available with that client. The data access URL is a URL about the data to which an authorization is provided to some client or group of clients by the owner of the data. Hence owner of the data has to perform an additional task of generating and managing URLs of the data. Also there is no search facility available to the client locating the non private data to which he or she is authorized to access.

In this paper, a new data storage organization for storage clouds is proposed, which would be helpful for effective data lookup. It would not only help the users to locate the private or non private data in a more efficient manner, but also maintain the privacy of the data. We have used Walrus storage service provided by Eucalyptus [10] an open source cloud platform for the deployment of cloud storage infrastructure and implementation of new storage organization.

The remainder of the paper is organized as follows. In Section II, we describe Eucalyptus Walrus architecture and challenges in current Walrus implementation in Section III. Mathematical model for the algorithm is described in Section IV. Section V provides Implementation Details and finally we conclude in Section IV.

2. EUCALYPTUS WALRUS ARCHITECTURE

Walrus is a storage service included with Eucalyptus which is interface compatible with Amazon's S3 [9], [11], [12]. Walrus allows users to store persistent data, organized as buckets and objects. It supports REST, SOAP and Bit Torrent protocols for data access [13], [14]. Figure 1 shows the internal architecture of Walrus. The Walrus storage is organized as a group of buckets. User can create the buckets and the data would be stored inside those buckets. Every storage cloud provider will have limitation on the number of buckets those could be created by every user. For example Amazon S3 allows maximum hundred buckets per user.

2.1 Walrus Buckets and Objects

A bucket is a container for data objects. Along with the objects it contains, a bucket has a name, owner, access control policy, logging information and a location as shown in Figure 1. Nesting of buckets is not allowed as buckets cannot contain other buckets. Each bucket within Walrus must have a unique name and it cannot be changed after it is created, but we can make a new bucket and copy the contents of new bucket into it.

A Walrus object is a container for data. Walrus has no knowledge of the contents of the objects. It just stores it as a bunch of bits. As shown in Figure 1, every object has key, value and access control policy attributes. Every Walrus object must belong to a bucket. There is no limit on the number of objects that can be placed in a bucket.

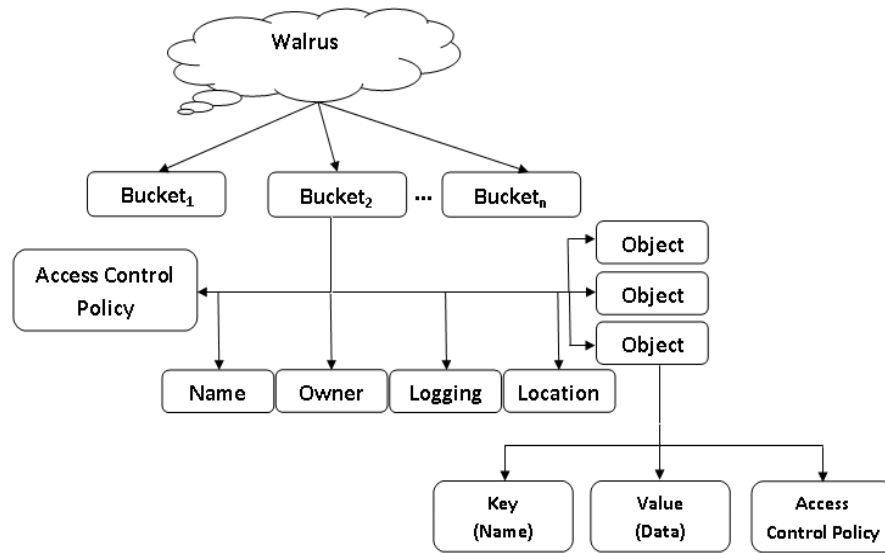


Figure 1: Eucalyptus Walrus Architecture

2.2 Walrus Access Control Policy

The access control policy could be set for each bucket and object stored in Walrus. By default the owner of the bucket or object will get the full permission to store, read, delete and modify the data. Owner can set the access control policy for his own bucket or object according to which all other users will be given an access to it. We can set READ, WRITE, FULL CONTROL, READ ACP, WRITE ACP permissions to any bucket or object:

The permission can be granted to three different categories of users as follows:

1. Group Grantees: In this category, we can grant the permission to the group of users. The groups can further be classified into three different types viz. AllUsers, AuthenticatedUsers and LogDelivery.
2. Canonical User Grantees: If the owner wants to give the permission to some known users to access the data, and these users do not belong to the same group, then it can be done by setting the permission to each user id (also called as canonical id). In this case we can assign different permissions to different users, which is not possible in case of group grantees.
3. Email Address Grantees: In this category, users with specific email address will be granted the permissions.

3. CERTAINTY ANALYSIS OF DATA ACCESS BEHAVIOR IN WALRUS STORAGE

The buckets stored in Walrus are classified into three categories viz. private, public and protected buckets. For the owner the bucket is a private bucket. If the permission for accessing the bucket is granted by owner to some other user then the bucket would be treated as a protected bucket for that user and if the permission is given to all users then the bucket would be treated as a public bucket.

When the user tries to fetch the list of accessible buckets using existing tools like s3cmd, s3fs, s3curl, or jets3t which are used to access Walrus Storage Service; always the list of private i.e. certain buckets is provided. But the list of non private buckets will not be provided directly since Walrus do not maintain such list separately like private buckets. In this case Walrus expects the user to provide identity of the bucket like its name or access URL.

Suppose user U wants to access a set of buckets B . If B is a set of private buckets of U then the query would always be certain since the probability $P(B \cap U) = 1$ and hence no searching would be required to check whether U is authorized to access B . On the other hand, if B is a set of non private buckets then $P(B \cap U) < 1$ since the bucket may not belong to the user. This yields an uncertain decision and hence access control list of the bucket would be searched to check whether U is authorized to access B .

3.1 Challenges Identified In Existing Implementation

The existing Walrus implantation by default provides the list of private buckets of the users. The access to protected or public buckets would be provided on demand. For accessing protected or public buckets, the user has to either know the name of the bucket or the access URL for the bucket in advance. This imposes an additional overhead of generation, maintenance and deletion of bucket access URLs.

When the access URL is generated for the bucket, one of the parameters to be provided is the time in hours for which the URL will be valid. In this case, revoking the access before the specified time is not possible. In such circumstances even after revoking the access the bucket may be accessible to the user. This may compromise the privacy of the buckets.

The time required to locate the private buckets is less than that of protected or public buckets. When an attempt is made to access a protected or public bucket, the Access Control List of the bucket is searched to check whether or not the user is authorized to access it. Due to this searching, the time required to access such buckets increases. To illustrate this, the sample test run is conducted with 100 registered users with Walrus Storage. For the test, 5 buckets are created per user and out of those 5 buckets one bucket is made protected and one bucket is made public. The time required to access the buckets by five random users is recorded. Table 1 shows the time required to access the buckets.

4. MATHEMATICAL MODELING OF THE SYSTEM

Let B represents a system representing the buckets stored in Walrus such that,

$$B = \{n, s, f, d, m \mid F(d), F(m), F(f)\} \quad (1)$$

where,

n = name of bucket

s = size of bucket

f = set of flags which will indicate whether the bucket is private, protected or public

d = data present in the bucket

m = address of machine at which the bucket is present

and for every bucket $b \in B$,

$$F(d) : n \rightarrow d$$

$$F(m) : n \rightarrow m$$

$$F(f) : n \rightarrow f$$

Also let U be a system represents the set of users such that,

$$U = \{u, l_{private}, l_{protected}, l_{public} \mid F(private), F(protected), F(public)\} \quad (2)$$

where,

u = user id

$l_{private}$ = list of private buckets

$l_{protected}$ = list of protected buckets

l_{public} = list of public buckets

and for every user $u \in U$ and for every bucket $b \in B$,

$$F(private) : u \times b \rightarrow l_{private}$$

$$F(protected) : u \times b \rightarrow l_{protected}$$

$$F(public) : u \times b \rightarrow l_{public}$$

In order to facilitate locating the private and non private buckets efficiently, we have reorganized the metadata of the buckets in an in memory data structure. This in memory data structure is designed using two different strategies viz. linked organization and hashed organization.

Private Buckets in Nanoseconds	Protected Buckets in Nanoseconds	Public Buckets in Nanoseconds
13541356	17716525	20751198
12771076	15149817	22186371
27476716	32278136	27666525
22445588	25973608	23591333
27673747	32644019	29809070

Table 1: Time Required for Locating Buckets

4.1 Representing Internal Data Structure Using Linked Organization:

Let $C\langle i,j,k \rangle$ be the index organization of buckets for each user $u \in U$, such that $1 \leq i \leq 3, 1 \leq j \leq 62$ and $1 \leq k \leq n$, where i represents the list li in which the bucket is present such that $\{0$ private list, 1 protected list, 2 public list $\}$, j represents the index of sub list lj into which li is partitioned on the basis of starting character of the bucket name which may be from the set $S = \{a, b, c, \dots, z, A, B, \dots, Z, 0, 1, \dots, 9\}$, where $|S| = 62$ and k represents the appropriate index of the bucket in j^{th} list. The storage organization is shown in Figure 2 and Figure 3.

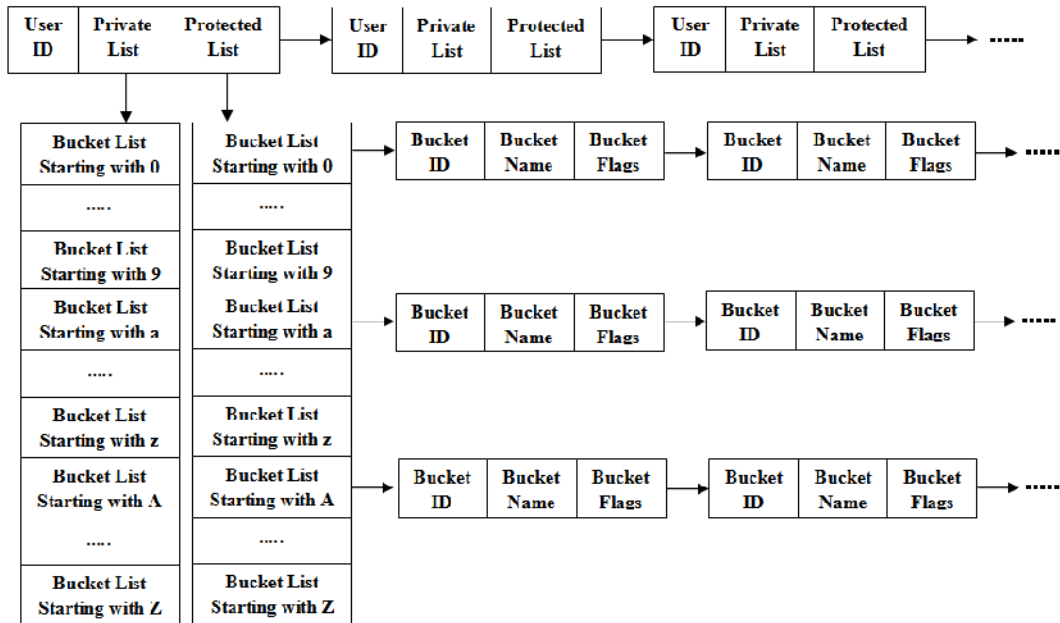


Figure 2: Data Structure for maintaining Private and Protected List of Buckets of Users Using Linked Organization

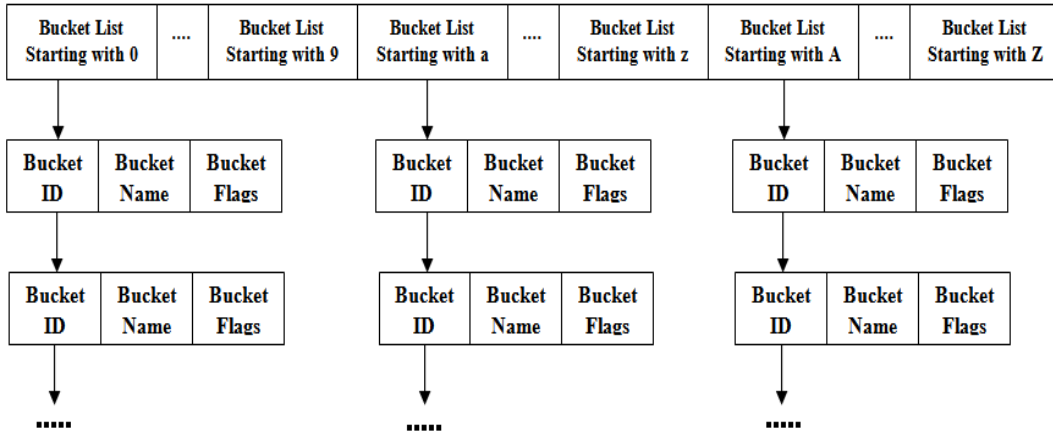


Figure 3: Data Structure for maintaining Public List of Buckets of Users Using Linked Organization

Given any query $q(u, b)$, where u is user id and b is bucket name, we have to retrieve an index $C\langle i, j, k \rangle$ if bucket b user u . If we could guess the presence of bucket in specific list and could modify the query q with additional parameter p such that $q(u, b)$ will become $q(u, b, p)$ where p is a list in which the bucket is more likely to be present, then the probability that the bucket is located faster increases just because this problem can be reduced to 3-coloring of graph problem and it can be represented as:

$$P(B | Li) = \frac{P(Li \cap B)}{P(Li)} \quad (3)$$

The parameter p can be calculated by asking the user to specify the additional information along with the query. For example, if user wants some bucket with name ‘abc’ then he can specify the query like “search my ‘abc’ bucket. Here we can guess that the bucket belongs to the user so it may be his private bucket and the probability that the bucket will be present in private list will be higher. In this case we can search it first in the private list.

Further each j^{th} list will hold the list of size n of all buckets starting with alphabet j . Hence in order to find the index k , we have to search all n values of the list. For reducing the time required to search further, we can use some threshold such that if $0 < n$ then we will perform sequential search and if $< n$ then we will perform binary search which will return the index k .

4.2 Representing Internal Data Structure Using Hashed Organization:

Let $H\langle k,v \rangle$ be the hash organization for all the users $u \in U$ where k is the key representing user id and v is the associated value representing the hash tables of private and protected buckets for key k as shown in Figure 4; such that $h(\text{UserID}) = k = v$. Here h is the hashing function which maps user id to corresponding key. Also let $H_{\text{Pub}}\langle b,v \rangle$ be the hash organization for all buckets $b \in B$ such that b is public. Since these buckets are accessible to all, they are stored in separate hash table.

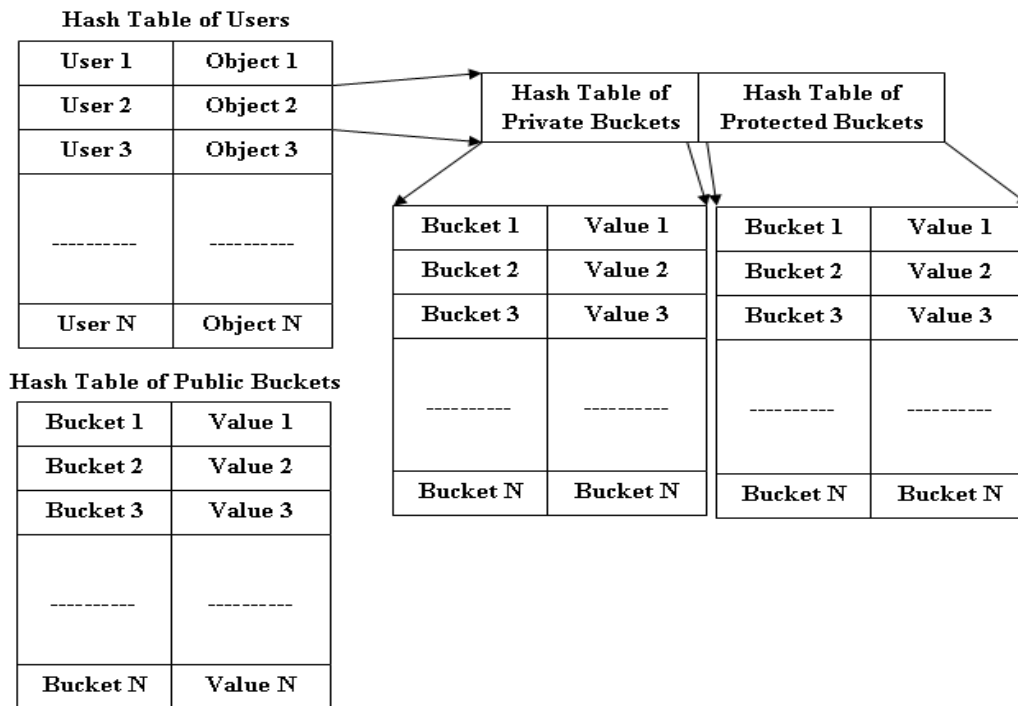


Figure 4: Data Structure for maintaining List of Buckets of Users Using Hashed Organization

Once the user u sends the request to access the data, the hash function h will map the user id to key k and will retrieve an object from hash table containing a $H_{\text{Priv}}\langle b,v_2 \rangle$ and $H_{\text{Prot}}\langle b,v_3 \rangle$ where H_{Priv} and H_{Prot} are the hash tables for private and protected buckets and b is the bucket name and v_2, v_3 represent data associated with the buckets. From these two hash tables the required data is retrieved along with the data present in hash table of public buckets. In this case

the time required for searching is reduced since directly hash value will be calculated and value associated with that hash value will be returned instead of traversing the whole list.

5. IMPLEMENTATION DETAILS AND RESULTS

We are using Eucalyptus Walrus to implement the prototype. The version of Eucalyptus used is 2.3.0. Since many tools are available to access the Storage Service, instead of modifying each tool, we have decided to modify the source code of the Walrus. From the source code, the main files in which changes done are WalrusManager.java from package called walrus and DatabaseAuthProvider.java from package called authentication.

The performance could be measured using the following parameters:

1. The additional amount of memory required for the new data structures.
2. The amount of time required for retrieving the list of all buckets accessible for a particular user.

6.1 Statistics for Additional Amount of Memory Required

The new data structures are implemented within Eucalyptus Walrus source code to locate the buckets in an efficient manner. The list of all users registered with Walrus is maintained in memory. Further for every user, two different lists are maintained; one for storing the information of private buckets and other for protected buckets of the user. In order to improve the searching performance further, each list is divided into 62 sub-lists (26 sub-lists are required for buckets starting the names with letters from a – z, in the same way 26 sub-lists are required for buckets starting with letters A – Z and 10 are required for buckets starting with letter 0 – 9). Here we have assumed that, the bucket name will start with either letter a – z or A – Z or 0 – 9. Since public buckets are accessible to all the users, instead of storing them in each user's list, a separate data structure is created for it. The additional amount of memory will be required for these data structures. For the calculation of memory requirement, the following assumptions are made:

- The average size of user name and bucket name field is 100 bytes.
- Every user has created 62 buckets which are private buckets for him and 62 protected buckets are accessible to him.
- Every user has given access of his one bucket to the group of all users. It means one bucket of every user is public.

With the above assumptions, the amount of memory required for every user could be given as:

Let U_{name} represents the size of the user id field and $S_{private}$, $S_{protected}$ and S_{public} represent the size of the private, protected and public lists of the user respectively. Also let L is the number of buckets present in one of the three lists and B_{name} is the size of the bucket name field. The memory required for each of the list could be given by:

$$S_{private} = L * B_{name} \quad (4)$$

$$S_{protected} = L * B_{name} \quad (5)$$

$$S_{public} = L * B_{name} \quad (6)$$

The total memory required for storing the bucket information of all the users could be calculated using equations (4), (5) and (6) and can be represented as:

$$Total\ Memory = \sum_{n=1}^N (S_{private} + S_{protected}) + S_{public} \quad (7)$$

Here N is the total number of users present in the Walrus storage system.

The amount of additional memory required for the different number of users is as shown in Table 1 and its respective graph showing how the memory requirement increases with the number of users is shown in Figure. 5. It could be found that even if the numbers of users accessing Walrus storage are 1 million, the additional amount of memory required to maintain the information about buckets is just nearly 1GB.

6.2 Statistics for Time Required to Locate Bucket List

The original implementation of Walrus Storage Service locates only the private bucket list of users and there is no mechanism to locate the information of non private buckets. The information about non private buckets will be provided only if bucket access URL is provided. On the other hand the modified implementation provides not only the information of private buckets, but also the information of protected and public buckets which are accessible to the user. Hence the amount of time required to locate the information of buckets with modified implementation would be more than that of original implementation. The following table shows that the time required with new implementation is comparable with the original implementation.

For measuring the time required to locate the buckets, both original and new Walrus implementations are configured on different machines with same configuration as Intel Core i5 processor, 8 GB Memory and 1TB disk storage. Initially 10 users are added in the system and the amount of time required for locating their buckets is noted. Then the number of users is increased gradually in multiple of 10 up to 100 and the time required to locate the buckets is

Sr. No.	Number of Users	Amount of Memory Required (MB)
1	100	1.20163
2	1000	12.0163
3	10000	120.163
4	100000	1201.63
5	1000000	12016.3

Table 1: Statistics for Memory Requirements

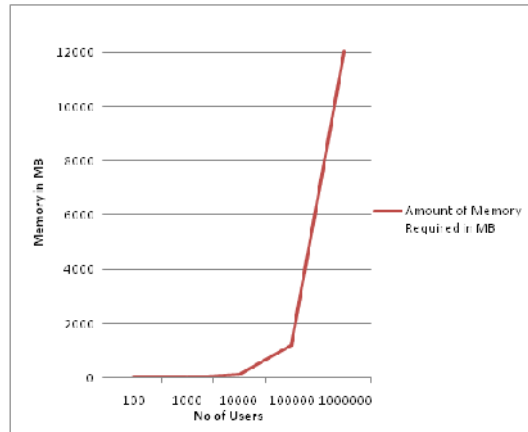


Figure 5: Memory Requirements for Internal Data Structures

calculated. Table 2 shows the statistics of the time required for locating buckets with existing Walrus implementation and Table 3 shows the statistics for the time required for locating buckets with new Walrus implementation with linked memory organization and Table 4 shows the same with hash memory organization. The average amount of time required for locating the buckets is shown in Table 5. The graph shown if Figure 10.2 shows the comparison between the time required for locating buckets with existing and new Walrus implementation.

No of Users	Time Taken By Users to Locate Buckets (mSec)									
	10	49.08	12.01	12.37	10.29	11.12	10.57	10.94	12.04	6.23
20	50.23	9.87	9.87	9.61	9.88	8.80	11.04	6.04	9.19	8.90
30	9.86	9.34	6.83	50.65	6.01	10.15	9.35	9.54	9.28	9.41
40	4.76	9.50	5.24	9.34	7.27	6.89	11.96	45.86	7.95	20.40
50	8.86	8.79	53.28	8.38	9.28	9.04	9.19	6.57	8.57	9.21
60	7.49	4.76	5.81	8.76	4.66	9.15	50.01	18.26	15.38	18.73
70	7.65	8.67	8.96	9.10	9.01	49.12	9.10	17.34	9.70	18.06
80	7.15	6.41	8.92	8.04	9.40	9.61	48.51	18.85	14.91	18.82
90	5.58	8.36	7.30	6.69	8.59	49.42	18.26	18.39	17.26	18.78
100	5.94	5.05	7.83	50.11	6.68	17.31	12.19	19.69	11.87	18.18

Table 2: The Time Required for Locating Buckets with Existing Walrus Implementation

No of Users	Time Taken By Users to Locate Buckets using Linked Memory Organization (mSec)									
	10	9.24	11.26	9.15	11.10	14.22	15.43	22.35	27.28	25.35
20	11.77	14.79	16.88	14.78	15.03	18.93	17.55	28.25	21.47	17.20
30	9.51	8.18	16.84	16.11	22.52	18.81	16.77	24.31	21.38	22.67
40	22.68	21.86	22.62	22.81	27.16	22.67	23.08	22.78	22.71	22.91
50	2.79	1.24	2.06	22.94	27.70	23.05	24.59	22.70	22.59	22.69
60	1.51	1.98	2.58	23.45	23.37	23.58	27.03	23.00	23.10	22.89
70	2.71	2.30	4.68	23.98	27.93	23.98	26.95	22.99	22.78	30.22
80	2.67	4.65	8.36	22.58	22.48	23.27	23.54	31.14	25.28	27.11
90	4.26	7.60	9.25	27.92	22.14	20.16	23.61	25.94	25.28	27.30
100	8.70	6.29	22.69	23.55	22.44	22.53	28.38	25.86	21.69	23.33

Table 3: The Time Required for Locating Buckets With New Storage Organization Using Linked Memory Organization

No of Users	Time Taken By Users to Locate Buckets using Hashed Memory Organization (mSec)									
	10	1.35	2.17	2.05	1.53	1.91	1.53	1.65	1.89	1.61
20	6.63	1.82	1.91	1.68	1.54	1.95	2.44	1.58	2.25	2.38
30	6.74	2.04	2.09	2.13	2.37	1.93	1.83	1.59	3.51	1.83
40	22.74	22.98	22.68	23.17	22.80	23.43	22.70	22.54	22.60	22.78
50	22.98	22.93	22.80	22.77	22.91	22.90	22.54	22.34	22.33	22.42
60	22.85	22.62	22.72	23.37	22.48	23.01	22.43	22.96	22.45	22.34
70	23.59	24.11	22.82	26.97	22.46	22.93	22.48	26.84	22.79	22.49
80	23/02	22.74	27.45	23.30	22.42	22.66	26.31	22.42	22.79	22.30
90	23.22	23.62	23.21	22.78	27.01	22.87	22.57	22.61	26.99	22.38
100	22.98	22.74	22.69	22.79	22.71	22.78	22.58	22.56	22.26	27.72

Table 4: The Time Required for Locating Buckets With New Storage organization Using Hashed Memory Organization

Figure 6 indicates that the time required for locating the buckets in modified implementation is more compared to original implementation. This is because the original implementation provides only the list of private buckets whereas the modified implementation provides the list of non private buckets also. We can also conclude that the time required for locating the buckets with linked memory organization is more compared to hashed memory organization. In linked memory organization we have to traverse the whole linked list for locating the users as well as buckets whereas in case of hashed memory organization the time required to traverse is less. It also indicates that, in case of linked memory organization, as the number of users is increased, the time required also increases but in case of hashed memory organization it is almost constant.

No. of Users	Time Required for Locating Buckets with Original Implementation (mSec)	Time Required for Locating Buckets for New Implementation Using Linked Memory Organization (mSec)	Time Required for Locating Buckets for New Implementation Using Hashed Memory Organization (mSec)
10	14.95205	17.27194	17.61534
20	13.34729	17.83423	16.78149
30	13.04566	17.68960	17.04811
40	12.92219	17.56440	17.05349
50	13.12308	17.23879	16.86654
60	14.30290	17.25353	16.60352
70	14.67446	18.85695	16.75666
80	15.07000	19.11327	16.73796
90	15.86134	19.74968	16.96284
100	15.48699	20.54831	16.92950

Table 5: Average Time Required for Locating Buckets

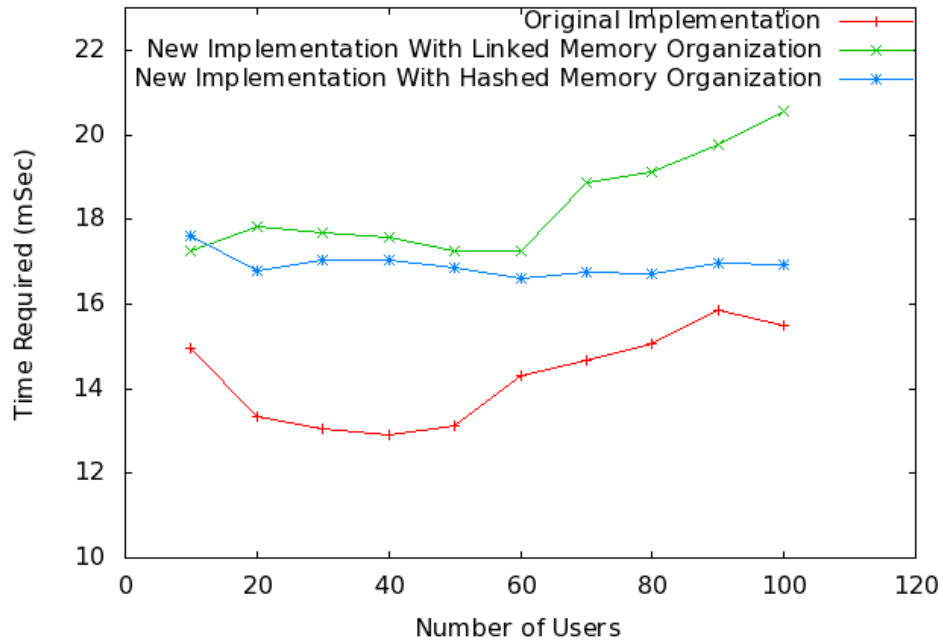


Figure 6: Time Required for Locating the Buckets

6. CONCLUSION

In this work an effective data storage organization is proposed which would reorganize the information of the data stored in cloud storage. This would be helpful to locate the data belonging for a particular user in an efficient manner. In contrast to the existing solutions which retrieve only the private data of the users easily, a new implementation provides non private data also to the user which the user is authorized to access. Even though access to non private data is provided, the privacy of the data will not be compromised. This will also help to eliminate the need to generate and maintain the access URLs which are required to provide the access to the private data of certain user to other users. When the access for certain data is provided to some user through access control policy, that data will be directly accessible to the user. The additional memory is required for the internal data structures, but this memory requirement could not be an overhead because even if one million users are using the storage service, just nearly 1GB of memory is required to be allocated. Since the new implementation provides the list of private as well as non private buckets to the user, the time required is more compared to original implementation. The algorithm being implemented inside the Cloud Storage, no other application tools providing the access to Walrus are required to be modified.

REFERENCES

- [1] Amazon Simple Storage Service [Online] Available at: <http://aws.amazon.com/s3/>
- [2] Windows Azure: Microsoft's Cloud Platform [Online] Available at: <http://www.windowsazure.com/enus/home/features/storage/>
- [3] Google Cloud Storage [Online] Available at: <http://cloud.google.com/products/cloud-storage.html>
- [4] Atmos – Cloud Storage, Big Data – EMC [Online] Available at: <http://www.emc.com/storage/atmos/atmos.htm>
- [5] Cloud Computing Software from Eucalyptus [Online] Available at: <http://www.eucalyptus.com/>

- [6] Enterprise Cloud Storage – Nirvanix Public Hybrid Private Clouds [Online] Available at: <http://www.nirvanix.com>
- [7] MezeoCloud – Mezeo Cloud Storage [Online] Available at: <http://www.mezeo.com/platform>
- [8] Cloud Computing, Cloud Hosting and Online Storage by Rackspace [Online] Available at: <http://www.rackspace.com/cloud>
- [9] “Amazon Simple Storage Service API reference, API version 2006-03-01,” 2006.
- [10] D. Nurmi, R. Wolski, C. Grzegorzczak, G. Obertelli, S. Soman, L. Youseff, and D. Zagorodnov, “The Eucalyptus open-source cloud-computing system,” in CCGRID '09: Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid. Washington, DC, USA: IEEE Computer Society, 2009.
- [11] “Amazon Elastic Compute Cloud (Amazon EC2),” <http://aws.amazon.com/ec2/>.
- [12] “Elastic Block Storage,” <http://aws.amazon.com/ebs/>.
- [13] “Amazon Web Services Developer Community: Amazon S3 Authentication Tool for Curl,” <http://developer.amazonwebservices.com/connect/entry.jspa>.
- [14] M. A. Kozuch, M. P. Ryan, R. Gass, S. W. Schlosser, D. O'Hallaron, J. Cipar, E. Krevat, J. L'opez, M. Stroucken, and G. R. Ganger, “Tashi: location-aware cluster management,” in ACDC '09: Proceedings of the 1st workshop on Automated control for datacenters and clouds. New York, NY, USA: ACM, 2009.
- [15] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia. Above the clouds: A Berkeley view of cloud computing. Technical report, University of California at Berkeley, 2009.
- [16] M. Brantner, D. Florescu, D. Graf, D. Kossmann, and T. Kraska. Building a database on S3. In SIGMOD '08: Proceedings of the 2008 ACM SIGMOD international conference on Management of data, pages 251_264. ACM, 2008.
- [17] R. Buyya, C. S. Yeo, and S. Venugopal. Market-oriented cloud computing: Vision, hype, and reality for delivering it services as computing utilities. In HPCC '08: Proceedings of the 2008 10th IEEE International Conference on High Performance computing and Communications, pages 5_13. IEEE Computer Society, 2008.