International Journal of Digital Information and Wireless Communications (IJDIWC) 9(3): 157-164
The Society of Digital Information and Wireless Communications, 2019 ISSN: 2225-658X (Online); ISSN 2412-6551 (Print)

Implementation of Proof of Retrievability on the Cloud

Kosmas Kapis and Baraka Mwasampeta
University of Dar es Salaam, Dar es Salaam, Tanzania
E-mail: kkapis@gmail.com
E-mail: bamwasa@gmail.com

ABSTRACT

Data integrity is among the important aspects for the clients to make full use of the benefits brought by the use of cloud computing. However, clients are worried of the protection of their dynamically changing data integrity because the administration of their data is controlled by cloud service providers. This paper proposes the implementation of the Proof of Retrievability (PoR) scheme for assuring data integrity on the cloud storage with dynamically changing data. The study uses experimental method to test cloud environment using a simulation tool. The method employed empirical data that were extracted from existing documents. It further provides the implementation design of dynamic PoR scheme as a solution to the data integrity problem. The developed cloud solution allows a client to challenge the server for the integrity of static and dynamic data, and improves cloud providers' security by enhancing services to the clients with assured data integrity.

KEYWORDS

Data integrity, hashing algorithm, proof of retrievability, cloud computing, cloud storage.

1 INTRODUCTION

Currently, cloud computing has become a popular and very useful technology. It is a result of the invention of the Internet as its services are offered through the Internet. Its benefits include provision of scalable computing resources to users and removal of maintenance cost of computing resources from the clients. Cloud Computing allows clients to rent resources from cloud providers depending on the capacity they need. Rented resources can easily be scaled up or down whenever there is a change in client's requirements. Another benefit of Cloud computing is the decrease in the cost of buying computing resources in premises which cannot be utilized to their full capacities.

Cloud computing gets its name as the representation for the Internet on diagrams [10]. Normally the cloud picture can represent the Internet in network diagrams. Cloud picture on network diagrams can represent other infrastructures that support your network which are beyond clients' control.

The services offered by cloud computing are normally referred to "as a service" suffix. The term services in cloud computing is the concept of being able to reuse various systems' components across a vendor's network. Common cloud's offerings with "as a service" as a suffix includes Software as a Service (SaaS), Hardware as a Service (HaaS) which sometimes is referred as an Infrastructure as a Service (IaaS) and Platform as a Service (PaaS) [5]. With these services clients can have access to application software, hardware like drives for storage, hardware infrastructure for network connections and programming environment through the Internet.

With SaaS, users can access and utilize any application with a web browser from servers on the Internet. Here servers centrally host software and users can access it on a subscription basis. With HaaS, users have an advantage of using scalable hardware services like hard disk storage from the servers on the Internet. Other HaaS services, referred as IaaS is the utilization of the backbone infrastructure offered by Internet Service Providers (ISPs). Depending on the bandwidth usage, the user will have to pay to ISP for the use of their infrastructure. Finally, with PaaS users especially programmers get the environment for creating and running application programs on servers on the Internet. PaaS gives users environment to type and runs their codes [5].

Clients' data in the cloud continuously change every moment when updated. This makes it difficult to guarantee data security, particularly

dynamic data integrity. Several scholars have attempted to address this problem. They only considered an issue of servers to declare data possession to clients [2] once stored, proving if clients are able to access their data after storing it on the cloud [3] and the issue of involving third party authority (TPA) on checking the former aspects for clients with limited storage and computing power [1].

Regardless of all the mentioned efforts made so far to benefit from the advantages of cloud computing service, still data integrity protection worries to clients exist. Clients do not trust cloud service providers because they have no control over their data. The clients' data in the cloud changes every moment when updated. Existing implementations have not considered a solution combining the static and dynamic data stored in the cloud. This calls for an implementation solution to guarantee data integrity when data stored in the cloud is dynamically changing. Furthermore various aspects concerning data integrity were solved separately and the issue of emphasizing the trust on the TPA was not addressed. The model presented on this paper address all these issues.

In order to minimise or remove the risk of compromising client data integrity in the cloud, this paper proposes a design for implementing dynamic proof of retrievability scheme on cloud storage The designed implementation is based on Task-Technology Fit theory [4] depicted in Figure 1. This theory demands the cloud storage to offer the best way of accomplishing the task of storing client's data at the convenience of the client. It underlines the clients requirement for their stored data to be well protected. Such cloud storage service would be the one which implements the dynamic PoR scheme for ensuring integrity of both static and dynamic data.

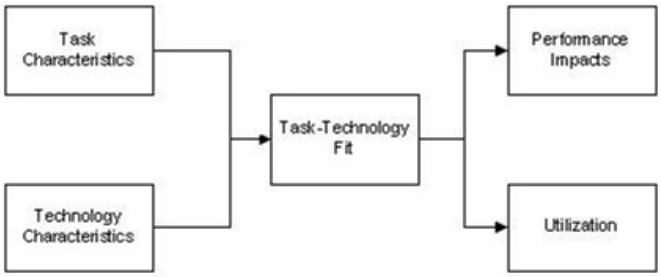

Figure 1: Task-Technology Fit Theory (Source: [4]

2 RELATED WORKS

Any cloud storage service is implemented based on one or more of the existing proven working deployment models. According [12], deployment models refer to various architectures that are used in implementing Cloud storage. The strength of the model to deal with various security threats makes it much useful on deploying cloud service to clients. Since there are different security issues and a client may be more interested with one or few issues than others then there are different models, which may be deployed. Various models have been presented and explained on how different users' concerns can be resolved by their use. Such models include Separation model, Availability model, Migration model, Tunnel model and Cryptography model [12]. Apart from the use of any of these models still the clients have to be able to challenge cloud servers about the integrity of their stored data.

Literature provides two security schemes, Provable Data Possession (PDP) [2] and Proof of Retrievability (POR) [3], on which various security models are based on. PDP assures clients of the data on the cloud storage that their data will remain there without being deleted. On the other end, POR provide the same service as PDP with addition provision of the proof that clients can be able to retrieve their data once stored on cloud storage. POR inform the client in any case of data corruption or modification. Both of these schemes work on static data.

In literature, there are schemes that have been proposed as modification of the primary schemes [2] and [3], to work with dynamic data. A scheme proposed by [11] is one of such schemes. This scheme supports dynamic data operation. It uses of Merkle Hash Tree on creation of meta-data. In so doing the scheme allows the clients to perform block-level operations, on the data files while maintaining the same level of data correctness assurance. The design is efficient enough to ensure seamless integration of public audit-ability and dynamic data operation support. The data files are split into blocks and stored on cloud storage as a tree, which is manipulated to ensure integrity due to dynamic operations on a file.

The structure of the merkle tree proposed by [11] is a binary tree since each node which is not a

leaf node is pointing to two child nodes. Thus the total number of leaf nodes under the binary tree is of the total number of even nodes. Any change on the binary tree is therefore required to maintain the even number of nodes. The number of nodes follows the sequence of the number on nodes of 1, 2, 4, 8, 16 … from the root to leaf nodes depending on the levels contained in the tree.

Figure 2 depicts an example of authentication using the Merkle hash tree, where the authentication of data blocks x2 and x7 are checked. First the computation of h(x2) and h(x7) is done then hc = h(h(x1)||h(x2)), hf = h(h(x7)||h(x8)), ha = (hc||hd), hb = (he||hf) and hr = (ha||hb). The root node at the highest level stores the hashes of its children nodes. The child node subsequently stores the hashes of its children nodes until leaves are met. Leaf nodes are hashes of actual data blocks. The leaf nodes store the hashes of actual data blocks and once a new block is added or a block is removed then the stored hashes are modified. Only the affected leaf nodes and their parent nodes will be hashed again in case of updates leaving the hashes of unaffected nodes un-updated.

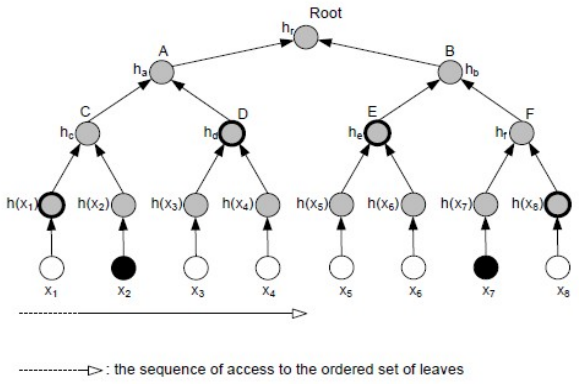

Figure 2: Merkle Hash Tree Authentication of Data Elements [12]

[6] provide a scheme that extends the static PoR scheme to the dynamic scenario. The scheme allows the client to perform update operations such as insertion, deletion, and modification. After each update, the client can still detect the lost data even if the server tries to hide them. The authors of the scheme developed a version of authenticated data structure based on a B+ tree and a merkle hash tree calling it Cloud Merkle B+ tree (CMBT). They proposed a dynamic version of PoR scheme by combining the CMBT with the Boneh–Lynn–Shacham (BLS) signature. Comparing with the existing dynamic POR scheme proposed by outhors in [11], the worst-case communication complexity of this scheme is O(logn) instead of O(n).

The following algorithms are suggested by [6]: First an algorithm, KeyGen(1k) (pk,sk). This is meant for generation of public and secret keys. It is run by the client. Security parameter, data which would be useful for the generation of keys, is an input to this algorithm and output are the public key, pk, and a private (secret) key, sk. The client stores the private key and the public key is sent to the server. This algorithm will be managed by Certification Authority (CA).

Prepare(sk, F', F_{tags}) (Φ, sigsk(v(R)), CMBT) is a second algorithm run by the client. It takes an encoded file, F', which is composed by a sequence of blocks m_i, where $0 \le i \le n$, the block tag set $F_{tags} = \{H(m_i), 0 \le i \le n\}$ and the private key sk. This algorithm outputs a set of signatures Φ which is an ordered collection of signatures $\{\sigma_i\}$ on $\{m_i\}$, where $0 \le i \le n$, CMBT and signature of the root sigsk(v(R)) are created by the client using the private key (Mo et al., 2012).

GenChallenge(n) Q is a third algorithm proposed in Mo et al., (2012). When executed, it allows the client to challenge the server for the integrity of the blocks of the data. The input to the algorithm is the total number of blocks, and the output is a query Q. Q is sent to the server as a request to verify the integrity of blocks [6].

A fourth algorithm in [6] is GenProof(Q, CMBT, F', Ftags, Φ) P. It is run by the server to generate a proof for the query Q. The input to this algorithm are Q, CMBT, F', F_8 and Φ. The output from this algorithm is the proof P that lets the client know the integrity of the blocks of stored file. P is sent back to the client so that client can test the file integrity [6].

Upon receiving of the proof P, a client runs the fifth algorithm to verify the received proof, Verify(sk, Q, P, v(R)) (TRUE, FALSE). It is run by the client to test the validity of the proof received from the server. Inputs to this algorithm are pk, Q, P and v(R). Having the private key, sk, as an input to the algorithm means that only appropriate client could be able to verify the proof. The algorithm returns TRUE if the proof received from the server is valid and returns FALSE otherwise [6].

To support data dynamic, [6] suggested more algorithms to deal with the aspect. UpdateRequest() Request is executed by the client to request modification of data file stored on the server. This algorithm receives nothing as input and gives out an update request which contains an Order $\in$ {Insert, Delete, Modify}. The other algorithm is Update(F', F_{tags}, Φ, R) (P_{old}, P_{new}) which is run by the server. This algorithm gives output containing two proofs P_{old} and P_{new}.

The last algorithm is UpdateVerify(P_{old}, P_{new}) (TRUE, FALSE) which is executed by the client. The UpdateVerify() algorithm verifies if the update algorithm ran on the server properly. It returns TRUE if the server's behaviour are honest and FALSE otherwise. The inputs to it are P_{old} and P_{new}.

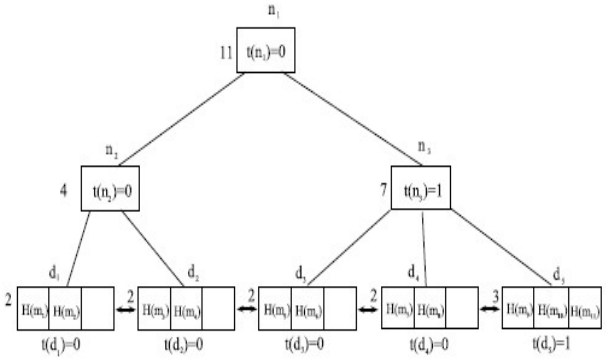

Figure 3: The Cloud Merkle B+ Tree [5]

Figure 3 show the CMBT on which leaf nodes are linked sequentially and stores three elements at most and two elements at least. The CMBT use the features of both a B+ tree and a Merkle Tree to take benefit of both. A B+ tree assists on ensuring the efficiency and a Merkle Tree assists on enforcing the integrity of data. This tree has two types of nodes which are index nodes and data nodes (which are the leaves on the tree) [6]. When updates on the CMBT tree are done they must leave the tree in a proper state with leaf nodes holding hashes of two or three blocks of data. On the same updates the index nodes should have ether two or three indices. The updates done on the file blocks are modification, deletion or insertion of blocks. From the UpdateRequest() Request method, the Request resulted will be modification, deletion or insertion depending on the type of update [6].

On modification of block say j^{th} block m_j to $m_j`$ (where $1 \leq j \leq n$) the update request Request = {Modify, j, $m_j`$, $\sigma_j`$} is generated. The request is sent to the server where the server will update the block, reconstruct the CMBT and generate the proof $P_{old}$ = Query(i). The server sends the new root node to the client, which is verified for the correctness. The correct new root node is signed, sigsk(v(R`)), and sent back to the server [6].

On insertion update, a block can directly be inserted into the node with two elements to make at most three elements. If the node has three elements, then the data node split into two nodes each with two elements, together with the new element. The splitting of nodes goes up until reaching an index node with two elements otherwise a new root is created adding a new level for CMBT. As with modification, the proof for insertion will be generated together with the signing of the new root [6].

On deletion, an element can easily be removed from the node with three elements and updating the CMBT. If the deletion needs to be performed on the node with two elements then the remaining node will be deficient for the requirement of CMBT. In this case, the borrowing from or merging with the next same level node should be done to keep the tree balanced. Like on modification and insertion, the tree will change hence proof for deletion is generated and the new root created by the server is signed [6].

Anuja and Kumar [1] propose a scheme called Outsourced Proof of Retrievability (OPoR). The scheme tackles the challenge of limited computation power of some devices. It is a cloud storage scheme that involves a cloud storage server and a cloud audit server. The latter is assumed to be semi-honest. This work considers the task of allowing the cloud audit server to pre-process the data before uploading to the cloud storage server and later verifying the data integrity, on behalf of the cloud users. OPoR outsources the heavy computation of the tag generation to the cloud audit server and eliminates the involvement of user in the auditing and in the preprocessing phases. Furthermore, OPoR strengthens the PoR model to support dynamic data operations, as well as ensure security against reset attacks launched by the cloud storage server in the upload phase.

From the works above, it has been noticed that [6] deals with the drawback on the complexity of

the work of [11]. But, still, the later work has not given a proper way forward on how this scheme could be implemented on the real environment of cloud computing. Various POR schemes just provide the way these schemes needs to operate without giving details on how various technologies can be combined to achieve the desired results. To solve this challenge, therefore, this paper provides a model and environment on the implementation of dynamic POR scheme. In this paper, the dynamic scheme with low complexity which make use of Merkle Hash Tree on creation of meta data has been implemented.

3 METHODOLOGY

The implementation of the dynamic proof of retrievability serves as a step toward ensuring the integrity of data stored on the cloud server. Experimental approach was used on testing the implementation of dynamic POR that will have low complexity to both the client and the cloud server. Simulation tool was created and used to analyze integrity aspects of files stored on the cloud. Figure 3 shows the three parties which communicate when the dynamic POR scheme is implemented on the cloud storage.

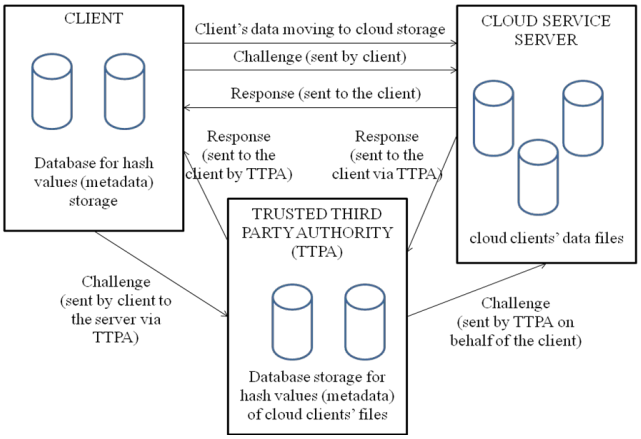

Figure 4. Interaction between Client, Cloud Server and TTPA

The designed figure 4 depicts direct interaction between the client and cloud server together with their interaction through the Trusted Third Party Authority (TTPA). The arrows show that a client may move data to the cloud storage and can be able to challenge the server for the integrity of the stored files. A client may only store database with meta-data and not actual files hence utilize effectively the storage capacity. The process of sending challenge and receiving response can be done by a client direct to the server or through TTPA. TTPA combines the functions of normal TPA as suggested by related works with the functions of Certification authority (CA). CA is a trusted part used to create users' certificates for public key infrastructure (PKI) [9]. This enhances the trust between the three parties, client, cloud storage and TTPA.

The model of cloud storage proposed shows that there is a creation of meta data as the hashed values of client's data files. In cryptography, the hash function is one of the techniques used for ensuring data security through integrity on computer systems. It allows creation of a fixed length hash value from a variable length message [9]. Hash functions are commonly used for the storage of passwords in databases and message authentication when communicating through networks. They are also used to determine whether data has changed or not. Many algorithms and processes that provide a security service use a hash function as a component of the algorithm or process, including Keyed-Hash Message Authentication Code (HMAC), Digital Signatures, Key Derivation Functions (KDFs) and Random Number Generators (RNGs). We tested our model by the use of Secure Hash (SHA1) and message digest (MD5) functions which have been compared for their effectiveness.

Figure 5 presents the solution design that had been simulated. The aspects shown on the design are cloud storage service settings, data storage operations, cloud storage access, data dynamic implementations and integrity checking. The design came out with a good solution to the integrity problem on the cloud storage.

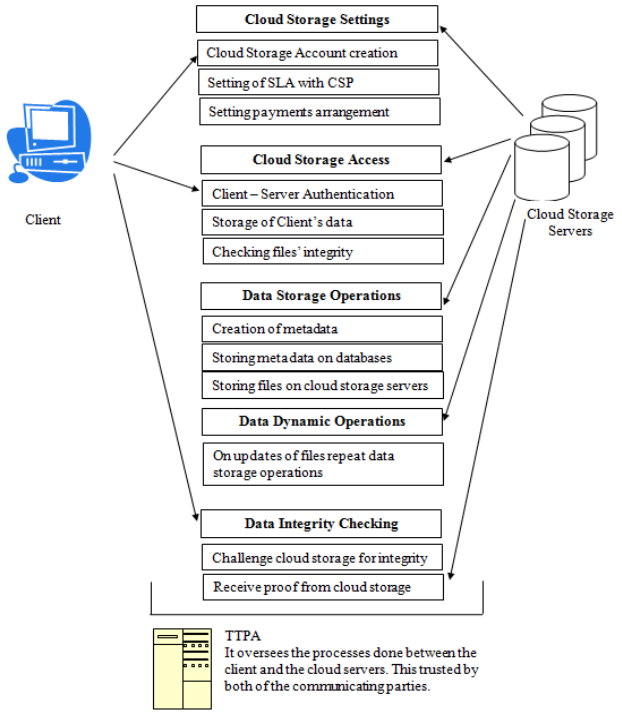

Figure 5: Solution Design

4 FINDINGS

Table 1 shows the details for file sizes and upload time with the use of MD5 and SHA1 encryptions respectively on the selected files. MD5 and SHA1 hashing algorithms were used due to their less use of computer's computation power resulted by lower bit size [9]. Other algorithms might be used based on the corresponding requirements of the storage system. It was observed that the larger the file the longer it takes to upload it to the server. Also the time spent while using SHA1 was much higher than time spent while using MD5.

Table 1. Simulation's Upload Durations when Using SHA1 and MD5.

Sn	File name	Size (Bytes)	Duration (msec)	
			SHA1	MD5
1.	email.txt	258	1,064	1,053
2.	vitabu.txt	276	1,046	1,069
3.	index.html	729	2,437	2,328
4.	graph.jsp	997	3,387	3,338
5.	contact.txt	1,155	3,871	3,800
6.	myfiles.jsp	2,395	7,971	7,650
7.	Plot.java	3,415	11,089	10,720
8.	check.jsp	3,521	11,766	11,319
9.	upload.jsp	7,104	22,896	22,783
10	index.jsp	8,597	28,306	27,420

Response time to computer user affect user behaviour on accessing desired service. Web users may face a long waiting time during accessing web pages, and there are various technologies and techniques are normally implemented to control the situation and to calm the annoyed users [8]. Many factors affect waiting times especially the processing power, and memory of a server and researchers show that there are some thresholds for minimum waiting times for web users. Waiting times on data collected and shown on Table 1 may not be tolerable for most users but on real cloud environment, the issue could be solved by providing enough resources. The work by [1] suggests that website owners like CSPs should investigate ways in which they can attract and retain more users to their services and one of the issues is to improve response time. This will provide a way to reduce frustration and stress for the users of cloud services. The response time values for MD5 and SHA1 have been plotted and shown in the line graph on Figure 6.

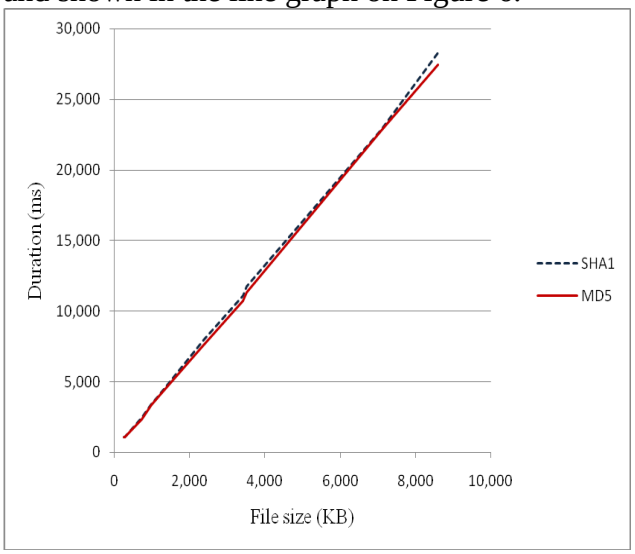

Figure 6. Line Graph Relating SHA1 and MD5

Experimentally, SHA1is more secure than MD5 though a computer experiences more delays due to its long hash code bit size. Both SHA1 and

MD5 have been used for experimental purposes due to their less utilization of computing resources. There are more secure algorithms than these two algorithms which are more complex due to their large bit size and mode of their functioning. SHA1 and MD5 have small bit size which can be easily studied on a computer with limited processing capacity. The complexity of hashing algorithms increases with the increase of the bit size and are effective as long as there are enough resources for them to rum.

5 DISCUSSION

This paper suggests the PoR implementation design which allows the client to be informed in case of any unintentional or illegal modification of stored files on cloud storage. Cloud servers can not compromise client's files since the client will be aware of any case of modification upon request or based on automated integrity messages. This is achieved on practical cloud storage once PoR is implemented and gives more trust to clients on the utilization of cloud storage service.

Many security concerns raised on using cloud storage were reviewed among which integrity is included. Specifically, the work provided the way forward to deal with the integrity of user's data placed on the cloud storage. Integrity is among the concerns of which users need to be assured of their provision for their data once placed on the cloud storage. This work shows how clients on cloud storage can now be able to keep track of their files by having an interface which allows them to request the status of data integrity.

This work combined the conventional TPA with CA to provide TTPA which is more effective compared to the TPA previously used on proposed schemes. TTPA is easily trusted by both the client and CSP as it provides authentic identification of communicating parties by having abilities of CA. TTPA will establish trusted communications between client and CSP at the same time will be used for POR on behalf of the clients. This study does not cover authentication and authorization aspects of security. These aspects are left open for future research.

6 CONCLUSION AND FUTURE WORK

The locations of all servers are normally transparent form user's point of service access. Message digest should be stored on a database that is placed on the location different from the cloud storage for the file. These will improve the security of the files stored and an assurance that the proof given by POR is true. Also, the performance of the entire system is improved when tasks are distributed across multiple servers.

The design presented in this research is general and can be implemented using a different number of technologies depending on preferences. As being observed in the simulation of the cloud storage, the performance was affected by the scheme used for hashing files before storing them. It was observed that new technologies invented in very short intervals of time. Since the technology is changing one could consider using the hash function that is more effective to be used at the time.

In addition, to increase the performance, it is advised to use the server having enough specifications capable of handling the required processes. This will reduce the server side delays for the processes. Improvement of re-computation of hashes every time a user has updates on the stored data can be a case for future works. This should be considered and one must find a model that could avoid the bottleneck on the system and improve the entire performance of the cloud storage system. As seen on [7], delays affects much the performance of the systems hence add stress to online users.

Furthermore, implementers of the dynamic PoR scheme may improve this scheme by allowing it to check retrievability automatically and finding a way to inform a user in case of the corruption of data integrity. Information to users can be provided by various ways such as email or phone numbers. Enabling this will make the cloud storage provider put appropriate measures on ensuring their customers' data integrity otherwise they may lose customers. This is a simple task of automation and in some cases could be troublesome to clients since it could result on forwarding messages to them without request. Thus clients may be given options to decide whether to receive automated messages about integrity of their files or not.

REFERENCES

1. Anuja, K., & Kumar, A. N. (2015). A Survey On Outsourced Proof Of Retrievability In Cloud Computing. Journal of Applied Sciences Research, 11(19), 10-13.
2. Ateniese, G., Burns, R., Curtmola, R., Herring, J., Kissner, L., Peterson, Z., & Song, D. (2007). Provable data possession at untrusted stores. Paper presented at the Proceedings of the 14th ACM conference on Computer and communications security.
3. Bowers, K. D., Juels, A., & Oprea, A. (2009). Proofs of retrievability: Theory and implementation. Paper presented at the Proceedings of the 2009 ACM workshop on Cloud computing security.
4. Goodhue, D. L., & Thompson, R. L. (1995). Task-technology fit and individual performance. MIS quarterly, 213-236.
5. Miller, M. (2008). Cloud computing: Web-based applications that change the way you work and collaborate online. Indianapolis, Indiana: Que publishing.
6. Mo, Z., Zhou, Y., Chen, S. (2012). A dynamic Proof of Retrievability (PoR) scheme with O (logn) complexity. 2012 IEEE International Conference on Communications (ICC) (pp. 912-916). IEEE.
7. Moody, G. D., & Galletta, D. F. (2015). Lost in Cyberspace: The Impact of Information Scent and Time Constraints on Stress, Performance, and Attitudes Online. Journal of Management Information Systems, 32(1), 192-224.
8. Nah, F. (2004). A study on tolerable waiting time: how long are Web users willing toto wait?. Behaviour & Information Technology, 23(3), 153-163.
9. Stallings, W. (2007). Cryptography and Network Security Principles and Practices. Upper Saddle River, NJ: Prentice Hall.
10. Velte, T., Velte, A., & Elsenpeter, R. (2009). Cloud computing, a practical approach. New York: McGraw-Hill, Inc.
11. Wang, Q., Wang, C., Ren, K., Lou, W., & Li, J. (2011). Enabling public audit-ability and data dynamics for storage security in cloud computing. IEEE transactions on parallel and distributed systems, 22(5), 847-859.
12. Zhao, G., Rong, C., Jaatun, M. G., & Sandnes, F. E. (2012). Reference deployment models for eliminating user concerns on cloud security.