A Novel Heuristic Auditor for Revealing Strong Consistency Violations in Cloud

Y. Narasimha Rao, PhD
Professor ,Department of Computer Science
School of Computing & Informatics,
Mizan-Tepi University, Ethiopia

D. Sudha
Assistant Professor, Department of CSE
KCG College of Technology
Tamilnadu, India

ABSTRACT
To ensure that the services are always-on and globally distributed, cloud service providers sacrifice consistency for availability. Most Cloud Service Provider's provide only eventual consistency which is a form of weak consistency. Strong Consistency is required for certain applications which are interactive. In such cases an SLA is to be engaged between the Cloud Service Provider and the users which stipulate the level of consistency the cloud service provider should provide to the users of the data cloud. Existing Commercial clouds provide strong consistency guarantees but it is hard for the users to verify it. This paper proposes a Novel Heuristic Auditor based on loosely synchronized clocks which help the users to verify whether the data cloud provides the assured level of consistency as stated in the SLA. It uses a two level auditing structure to check for strong consistency violations namely Read-After-Write(RAW) consistency and Monotonic-Write(MW) consistency. Experiments were done to verify the strong consistency guarantees provided by Google Cloud Storage (GCS). The different types of storage buckets are tested for consistency violations and are quantified with different metrics.

Keywords
Cloud Computing, Strong Consistency, Data Staleness, Heuristic Auditor, Google Cloud Storage.

1. INTRODUCTION
Cloud computing while at its peak has greater impact on our everyday life as the number of users leveraging cloud is constantly increasing. It has been predicted that the annual global data centre IP traffic will nearly triple over the next five years and more than 75 percent of the workload will be processed by cloud data centres. This is the result of cheaper processors and cheaper storage options available today. While there is a significant growth in the use of cloud services such as Saas, Paas and Iaas there has been humongous growth in the use of consumer cloud storage as well. Users can store their music, photos, videos and documents in any of the cloud storage options available at a relatively low or no cost. The consumer internet population who will use personal cloud storage will definitely increase in the near future.

When considering the design of distributed data storage systems it is necessary to consider the CAP theorem. The CAP principle states that only two among the three factors namely Consistency, Availability and Partition tolerance can be achieved. Many distributed data stores offer availability and partition tolerance over strong consistency. The reason being stated is that short intervals of data staleness are less problematic than short intervals of unavailability. Hence most of the Cloud Service Providers (CSP) promise only eventual consistency which informally guarantees that if no new updates are made to a given data item, eventually all accesses to that item will return the last updated value.

Actually different applications have different consistency requirements. Not all applications could cope up with eventual consistency. Several applications require strong consistency which means that data viewed immediately after an update will be consistent for all observers of the entity. Some use cases that require strong or immediate consistency are "Online Document Storage systems where a group of users work collaboratively on a set of documents", "banking transactions that check balance, withdraw money or deposit money"," Online shopping systems" and so on.

It is imperative that these actions do not leave the database in an incorrect state even for split second. In such cases strong consistency is mandatory. If such applications which have strong consistency requirements needs to be deployed in cloud then an SLA should be engaged between the cloud user and the cloud service provider to ensure that these requirements are met. A third party auditor can be assigned to perform such auditing works but it may lead to disclosure of data or other important information related to the data to the third party auditor.

Let us consider an online shopping system where the users are geographically dispersed and several transactions are done in a single minute. The lists of available stocks are updated by a stock manager once a user has purchased a set of items. It is important that the list of available stocks is updated with strong consistency. Failing to do so will have adverse effects. A user may order an item which is out of stock or the sales report may vary by a large margin if a high valued transaction is not updated.

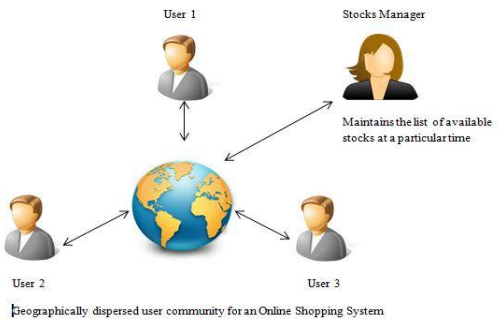

Fig 1. An application that require Strong Consistency

Hence it is important that such an application should be updated with strong consistency and no stale data exists.

Even when a cloud service provider promises strong consistency guarantees for its users, it is not evident for the

users whether the assured level of consistency is actually provided or not.

This paper proposes a novel heuristic auditor which verifies whether strong consistency requirements of an application are met. A group of users of a data cloud can perform auditing themselves and can measure the severity of consistency violations. Also we should notice that maintaining a global clock in such a scenario can be complex. Hence loosely synchronized clocks are used to maintain a global ordering among operations. The loosely synchronized clock is based on vector clocks algorithm which uses timestamping of events to achieve a global ordering among them.

The auditor proposed is used to verify whether strong consistency violations namely Read-After-Write (RAW) consistency and Monotonic-Write (MW) consistency are violated. A two-level auditing strategy to verify the above strong consistency models is proposed which can be done by the users and an auditor who is elected by the users. Finally the severity of such violations is measured.

2. RELATED WORKS

The three major factors of a distributed storage system are consistency, Availability and Partition Tolerance. Ref [3] made a comparison about ACID (Atomicity-Consistency-Isolation-Durability) Vs. BASE (Basically Available Soft state Eventual consistency) and mentioned traits about several systems that forfeits the above mentioned factors. It also suggested to keep consistency and availability within a single cluster but it is hard to reach at present.

Ref [6] analysed the historical perspective of consistency and how this changed after the introduction of distributed databases. The two ways of looking at consistency namely developer/client point of view i.e. how clients observe data updates and the later the server view which represents how updates flow through the system, are analysed. Strong and weak consistency and the variations of such consistency models are also presented.

Ref[7] which forms the base of trace based verifications aimed to check whether a series of events is safe, regular or atomicity analysing the trace of interactions between the client machine and the data store. By obtaining a global trace which is a list of read/write requests from all the clients, as well as the value retrieved or stored, it proposed algorithms to quantify the traces. It also verified the consistency provided by pahoehoe, a cloud storage system designed to offer extreme availability.

Ref [9] proposed online verification algorithms i.e. how to detect a violation as soon as one happens and proposed ways to quantify the severity of atomicity and commonality violations. It attempted to quantify the maximum staleness of all reads and the commonality of such violations.

Ref [10] proposed a Consistency-as-a-service model where a group of users of a data cloud can form a group to verify the consistency provided by the cloud service provider. A two level auditing structure is proposed to verify Monotonic Read Consistency and Read-Your-Write Consistency by means of local auditing and preserving Causal consistency through global auditing. An auditor was chosen randomly from the group of users of the audit cloud and was assigned to perform the global auditing task. A graph was constructed based on the events and if the resultant graph was acyclic then a violations was not encountered. The severities of such violations are quantified using different metrics. Data staleness and Commonality were the most commonly used metrics. A Heuristic Auditing Strategy (HAS) is proposed to reveal as many violations as possible.

Ref [11] used an approach of geographically distributed servers combined with a writer to fit the benchmarking system. They aimed to evaluate Amazon S3 in terms of consistency guarantees and their results proved that S3 frequently violates monotonic read consistency. Their findings justify the two-level auditing strategy used in this paper. Ref [13] assessed Amazon, Google and Microsoft's service and verified their consistency properties. The results proved that Amazon S3 doesn't provide its promised level of consistency and only eventual consistency was achieved which cannot be tolerated by all applications.

3. PRELIMINARIES

In this section we describe the structure of the user operation table (UOT) through which each user timestamps his own events. They form the basis for partial ordering of events in the system. The strategy used by the auditor as well as the users are briefed as well.

3.1 User Operation Table (UOT)

Each user maintains a user operation table (UOT) which comprises the entire list of operation done by him. Each user performs auditing through his user operation table. Each UOT comprise of operation, logical vector, physical vector and timestamp. Unlike the work of [10] an additional factor of timestamp is added to the UOT for the logging. The timestamp may represent the local time of the user's machine. Each operation in the system can be either a write W(K,a) which represents writing the value a to data identified by the key K or a read R(K,a) which represents reading the data a from the entity identified by the key K. . Each entity in the data store is identified by a key. Each W(K,a) has several dictated reads which are reads from same or other processes where a read R(K,a) will have a single dictated write as a read will represent value from a single write and not too many versions of it.

Each user will maintain two vectors apart from the operation id and timestamp namely the logical vector and physical vector. Each vector is an array of N elements where N is the number of users using the system, also $1 \leq i \leq N$ where I is the user id of the user. For example if Bob's user id is 1 and the number of users of the system is 3 then the logical vector would be of the form <LC1, LC2, LC3> where LC1 represents his own logical clock. In a similar way physical vector is also an array of n elements <PC1, PC2, …, PCN> and each user maintains his own physical clock at PCi. Initially all the clocks are initialized to zero. These two vectors are always incremented and they are never decremented. The logical vector and physical vector are updated in a similar way except that a physical vector is updated nevertheless an event occurs or not whereas a logical vector is updated only when an event occurs. An event can be one among the following: Read, Write, Send a message, Receive a message. The users of the system communicate asynchronously through messages. When a user sends a message to another user he appends his last entry in UOT along with the message so that the receiver can update his UOT in case he is not aware about others clock value.

The logical vector is updated via vector clocks algorithm. The vector clocks algorithm has the following four rules:

R1: Initially all values are zero.

R2: The local clock value is incremented at least once before

each atomic event.

R3: The current value of the entire timestamp array is piggybacked on every outgoing signal.

R4: Upon receiving a message, a process sets the value of each entry in the timestamp array to be the maximum of the two corresponding values in the local array, and in the piggybacked array received.

if localarray[q] := other_array[q]

then localarray[q] := 1 + other_array[q];

for i := 1 ton do locaLarray[i] := max(locaLarray[i], other_array[i]);

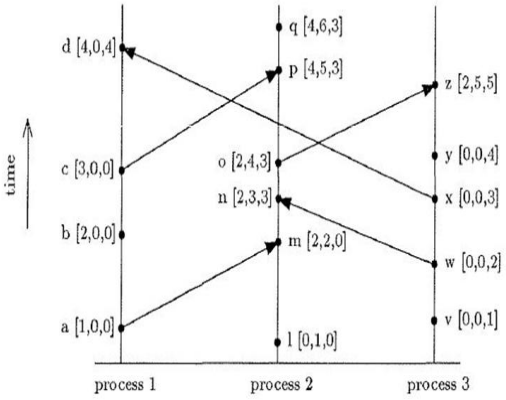

Fig 2. Use of Timestamped Arrays for Asynchronous Communication

3.2 Overview of Strong Consistency models

Read-After-Write Consistency:

Read-after-write consistency, guarantees immediate visibility of new data to all the clients. With read-after-write consistency, a newly created object or file or table row will immediately be visible, without any delays. There's also read-after-update and read-after-delete. Read-after-update consistency would allow edits to an existing file or changes to an already-existing object or updates of an existing table row to be immediately visible to all clients. That's not the same thing as read-after-write, which is only for new data.

Read-after-write consistency allows you to build distributed systems with less latency. Without read-after-write consistency we need to incorporate some kind of delay to ensure that the data you just wrote will be visible to the other parts of your system.

Monotonic Write Consistency:

A write operation by a process on a data item x is completed before any successive write operation on x by the same process (i.e. a write operation on a copy of data item is performed only if that copy has been brought up to date by mean of any preceding write operation, even if taken place on another copy of x.)

4. AUDITING STRATEGY

In this section we describe the auditing strategy used by the auditor as well as the users. Each user individually performs local auditing and the auditor performs global auditing.

4.1 Local Auditing

Local auditing (Alg. 1) can be done by each user as and when he is performing a read/write operation. This paper aims to verify strong consistency violations in a data store and hence require auditing reads as well as auditing writes to perform the auditing operation.

Algorithm 1 Local Auditing

Initial UOT with ∅
while issue an operation op do
if op = W(a)
if W(a) → W(b)
where W(b) is the last write in the UOT
then
Delay write
Monotonic write consistency is violated
record W(a) in UOT
R(c) ∈ UOT is the last read
if op = r(a) then
if W(a) → W(c) then
Read-after-write consistency is violated
record r(a) in UOT

Let W(a) denote user's current write operation, if there is another W(b) from the same process that has not yet been updated then the current write must be delayed until all the replicas are updated. Hence monotonic write consistency is violated. The number of violations is increased by 1 and the write entry is updated in the UOT. Let R(a) denote user's current read operation and its dictating write is W(a). If the last read entry in the UOT is R(c) whose dictating write is W(c) and if W(a) happens before W(c) then Read-after-write consistency violated. The read entry is updated in the UOT.

4.2 Global Auditing

An auditor is chosen among the users of the data cloud and is assigned to perform the global auditing work. Each user send their UOT entries to the auditor to obtain a global trace of operations. The consistency property is verified by constructing a directed graph. If the resultant graph is a Directed Acyclic Graph (DAG) then causal consistency is preserved else it is violated. This solution is inspired by [7] and their results justify the solution. An edge is added to the graph under the following conditions:

1) Time edge. For operation op1 and op2, if op1 → op2, then a directed edge is added from op1 to op2.

2) Data edge. For operations R(a) and W(a) that come from different users, a directed edge is added from W(a) to R(a).

3) Causal edge. For operations W(a) and W(b) that come from different users, if W(a) is on the route from W(b) to R(b), then a directed edge is added from W(a) to W(b).

The algorithm used for constructing the graph is shown below:

Algorithm 2 Global Auditing

Each operation in the global trace is denoted by a vertex

for any two operations op1 and op2 do

if op1 → op2 then

A time edge is added from op1 to op2

if op1 = W(a), op2 = R(a), and two operations come from different users then

A data edge is added from op1 to op2

if op1 = W(a), op2 = W(b), two operations come from different users, and W(a) is on the route from W(b) to R(b) then A causal edge is added from op1 to op2

Check whether the graph is a DAG by topological sorting

Global Auditing is performed to verify if causal consistency is preserved. In order to verify if the constructed graph is DAG, topological sort is performed on the nodes. The users can delete their UOT entries except the last read and write to save some space and through this way the last read and write are always available to the auditor.

5. EVALUATION OF GOOGLE CLOUD STORAGE

Google Cloud Storage is an Internet service to store data in Google's cloud. Google Cloud Storage allows world-wide storage and retrieval of any amount of data and at any time. This paper aims to test the consistency provided by Google Cloud Storage (GCS). It is mentioned by Google that Google Cloud Storage provides strong read-after-write consistency for all upload and delete operations. This means that after an object is uploaded successfully we can immediately download it, delete it, or get its metadata. Likewise, any attempt to access an object immediately after it is successfully deleted will result in a 404 Not Found status code. List operations are eventually consistent from anywhere on the Internet. In order to test GCS an instance of GCS is created. The application is designed to store files to the GCS bucket. The GCS bucket is by default placed in a high replication region hence a number of replicas of it are available. Any user of Google can access the application and can store files in the GCS bucket. As mentioned above each user has a user operation table and performs auditing as per the strategy mentioned. The auditor is not chosen at random as in [10]. A modified auditor election scheme is used.

5.1 Modified Auditor Election Scheme

Instead of choosing the auditor randomly among the users it would be wise to choose based on the user's abilities. There are several factors like availability, CPU, memory, bandwidth which can be considered. Here we choose bandwidth and availability as the two factors to choose the auditor. A user who has more bandwidth and who is more available at the system is the one who might perform well. Hence this scheme is more effective than choosing an auditor randomly. Each users bandwidth is tested when he login to the application. This data is cached and if there is a drastic change in the bandwidth at a later stage before he logs out then it is again recalculated. The amount of time the user was available at the system is logged. This process continues for a predetermined amount of time and once the time elapses a new auditor is elected based on the logged data.

5.2 Evaluation

Each user of the application logs their operations in the UOT. The UOT has a timestamp entry to mark the time of action performed. The UOT is updated as mentioned in section 3.

id	operation	user1_logical_vector	user1_physical_vector	user2_logical_vector	user2_physical_vector	user3_logical_vector	user3_physical_vector	timeEntry
1	initial	0	0	0	0	0	0	0000-00-00 00:00:00
2	Received Message	1	1	2	2	5	5	2015-03-31 19:34:35
3	write	2	2	2	2	5	5	2015-03-31 19:38:06
4	send_message	3	3	2	2	5	5	2015-03-31 19:39:18
5	write	4	4	2	2	5	5	2015-03-04 04:38:10
6	send_message	5	5	2	2	5	5	2015-03-04 04:38:47

Fig 3. Sample UOT from the application

As soon as a file is uploaded it is checked for monotonic write consistency if the file has been updated in all the replicas by making continuous read request to the uploaded files. The delay in making the write operation is also noted.

In order to verify monotonic consistency a predefined number of auditing reads are performed on the updated file. The time is divided into time slices and if a violation occurs in a timeslice then it is considered abnormal. The number of auditing reads in a timeslice i depends on the number of violations occurred in the i-1 timeslice. If a violation occur then it is more possible to continue in the upcoming timeslices too. The test runs are conducted several times. Each time the average delay for committing the write is noted and this is used as the adjusting factor so that a delay in write cannot be considered a violation.

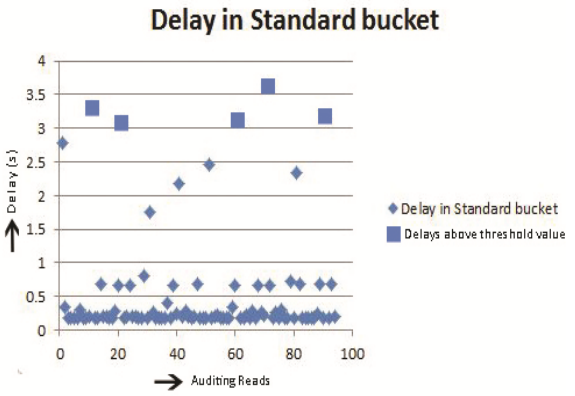

Fig 4a Average Delay noted in Standard GCS bucket

Fig 4 shows the average delay noted during the sample runs in Standard bucket. Similar to other works trying to verify the consistency properties of Amazon S3, here we try to deploy several auditing reads at different geographical regions. These auditing reads are performed from an application deployed in a Google App Engine project. This application uses a cloud SQL instance belonging to D0 — 128 MB RAM and a storage of up to 256 GB. In GCS the standard storage bucket can reside in 3 different regions. A standard storage bucket is created in the Asia region. The Reader processes as well as the SQL instance were deployed in the US region. The storage bucket and the reader processes are at different geographical locations. Objects in the standard bucket were continuously updated using cron jobs at very short intervals and the reader processes where made to read the updated data. Such reads are called as auditing reads. The number of such reads are not chosen at random but based on the number of violations that occurred at a specific interval. The physical time is divided into timeslices or intervals and several auditing reads are performed during these intervals. If a violation occurs at specific interval then it is more possible to occur at the upcoming intervals too and hence the numbers of such auditing reads are increased with the increase in violations. These runs were plotted using the average delay experienced

during each interval. At each interval the number of auditing reads is increased. As seen in Figure 4 the three runs experienced different delays, with runs having an average delay reaching 0.3 seconds. This can be used to set the threshold value after which such an operation can be considered a violation. This threshold value can be set after an extended set of runs.

The other type of storage bucket available in GCS bucket is Durable Reduced Availability (DRA) Bucket. All type of operations available in a standard GCS bucket is also available in a DRA bucket at a relatively low cost with a trade-off in availability. While a standard can be placed only in continental regions a DRA bucket can be placed in specific regions within the three main continental regions. Since they are less available the number of requests that could be processed in a minute is less compared to a standard GCS bucket. A DRA bucket is created in the Asia-East region. The reader process and the DRA bucket are present at different geographical region.

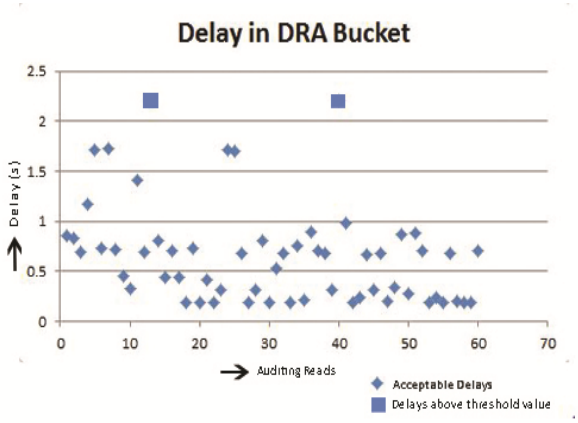

Fig 4b Average Delay noted in DRA bucket

They are less available than a standard bucket. Figure 4 shows the average delay noted in a DRA bucket. Cron jobs were deployed each minute with a timeslice of 10 seconds. The number of reader process deployed are less than the standard GCS bucket since the availability factor should also be considered. From the above two figures (Fig 3 and Fig 4) it can be seen that objects in DRA bucket experience larger delays than an object in the standard bucket. Since a DRA bucket is available at a relatively low cost this can be tolerable but there reads in timeslices 10-20 and 40-50 experiencing a delay of more than 2 seconds which is unacceptable. Such operations are considered as violations.

Nearline buckets are currently available as a Beta Release and have the same durability and nearly same availability characteristics as a standard bucket. They can be used to store data which are not frequently accessed for a long duration. A Nearline storage bucket is created in the Asia Region.

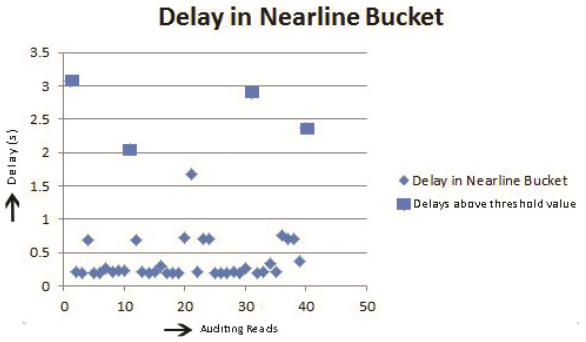

Fig 5. Average Delay noted in Nearline Bucket

The delays noted in Nearline buckets as seen in Fig 5 are randomly distributed with delays mostly averaging 0.25 seconds. The number of auditing reads performed on a Nearline Bucket within a single timeslice is very much lesser than the other two buckets. If the number of reads in a timeslice is increased the read operations quit with a fatal error. These types of buckets are mostly used as cold backup and cannot be accessed more frequently. As seen in Fig 5 objects in the nearline bucket experience very large delays. Such buckets cannot be considered for normal operations within an application. An SLA cannot be engaged for a Nearline bucket in GCS. Leaving out the delays it has been found that there have been few inconsistencies during the list and update operations in Google Cloud Storage and only eventual consistency was achieved. This is not the case of an Upload/ Delete operation in the GCS. They are strongly consistent.

The modified election scheme proves worthy as the auditor chosen based on the abilities performs well than a randomly chosen one.

6. CONCLUSION

In this paper we presented a Heuristic Auditor which verifies whether the data cloud provides the promised level of consistency. Here we considered only the strong consistency models like monotonic write consistency and read after write consistency. Such strong consistency is mandatory for interactive and several other critical applications. The users who deploy such applications can themselves verify whether the cloud service provider is actually providing the promised level of strong consistency. Google Cloud Storage is evaluated for strong consistency violations. Several reader processes called auditing reads where deployed to read data which could reveal violations. This work could be extended with other major cloud service providers who offer strong consistency guarantees to their users.

7. REFERENCES

[1] Kopetz, Hermann, 'Clock Synchronization in Distributed Real-Time Systems', IEEE Transactions On Computers, Volume:C-36 , Issue: 8, Page(S): 933 - 940 Aug. 1987.

[2] Cong Wang, Kui Ren, Wenjing Lou and Jin Li, 'Toward Publicly Auditable Secure Cloud Data Storage Services', IEEE Transactions On Network, Volume:24 ,Issue: 4, Page(s): 19 - 24 July-August 2010.

[3] E. Brewer, "Towards robust distributed systems," in Proc. 2000 ACM PODC.

[4] Mehdi Sookhak, Hamid Talebian, Ejaz Ahmed, Abdullah Gani, Muhammad Khurram Khan 'A review on remote

data auditing in single cloud server: Taxonomy and open issues', Elsevier Journal On Network And Computer Applications Volume 43 Page(s): 121–141 25 April2014.

[5] Qian Wang, Cong Wang, Kui Ren, Wenjing Lou and Jin Li, 'Enabling Public Auditability and Data Dynamics for Storage Security in Cloud Computing', IEEE Transactions On Parallel And Distributed Systems, Vol. 22, No. 5, Page(s): 847 - 859 May 2011.

[6] Werner Vogels, "Eventually consistent," Commun. ACM, vol. 52, no. 1, 2009.

[7] E. Anderson, X. Li, M. Shah, J. Tucek, and J. Wylie, "What consistency does your key-value store actually provide," in Proc. 2010 USENIX

[8] R. Zhang and L. Liu, "Security models and requirements for healthcare application clouds," in IEEE 3rd Int. Conf. on Cloud Computing, 2010, pp. 268–275.

[9] Marian K. Iskander, Tucker Trainor, Dave W. Wilkinson, Adam J. Lee and Panos K. Chrysanthis, 'Balancing Performance, Accuracy, and Precision for Secure Cloud Transactions', IEEE Transactions On Parallel And Distributed Systems, Vol. 25, No. 2, Page(s): 417 - 426 February 2014.

[10] Qin Liu, Guojun Wang, and Jie Wu, 'Consistency as a Service: Auditing Cloud Consistency', IEEE Transactions On Network And Service Management, Vol. 11, No. 1, Page(s): 25 - 35 March 2014.

[11] W. Vogels, "Data access patterns in the Amazon.com technology platform," in Proc. 2007 VLDB.

[12] Qian Wang, Cong Wang, Kui Ren, Wenjing Lou and Jin Li, 'Enabling Public Auditability and Data Dynamics for Storage Security in Cloud Computing', IEEE Transactions On Parallel And Distributed Systems, Vol. 22, No. 5, Page(s): 847 - 859 May 2011.

[13] D. Kossmann, T. Kraska, and S. Loesing, "An evaluation of alternative architectures for transaction processing in the cloud," in Proc. 2010 ACM SIGMOD.