Consistency as a Service: Auditing Cloud Consistency

Mr. Rupesh Chhatrapal Bichwe
M.Tech (CSE), Nuva college of Engineering & Technology,
University of Nagpur (RTMNU), Maharashtra, India
bichwerupesh@yahoo.com

Abstract—Cloud storage services have become commercially popular due to their overwhelming advantages. To provide ubiquitous always-on access, a cloud service provider (CSP) maintains multiple replicas for each piece of data on geographically distributed servers. A key problem of using the replication technique in clouds is that it is very expensive to achieve strong consistency on a worldwide scale. In this paper, we first present a novel consistency as a service (CaaS) model, which consists of a large data cloud and multiple small audit clouds. In the CaaS model, a data cloud is maintained by a CSP, and a group of users that constitute an audit cloud can verify whether the data cloud provides the promised level of consistency or not. We propose a two-level auditing architecture, which only requires a loosely synchronized clock in the audit cloud. Then, we design algorithms to quantify the severity of violations with two metrics: the commonality of violations, and the staleness of the value of a read. Finally, we devise a heuristic auditing strategy (HAS) to reveal as many violations as possible. Extensive experiments were performed using a combination of simulations and a real cloud deployment to validate HAS.

Index Terms—Cloud storage, consistency as a service (CaaS), two-level auditing, heuristic auditing strategy (HAS).

I. Introduction

Cloud computing has become commercially popular, as it promises to guarantee scalability, elasticity, and high availability at a low cost. Guided by the trend of the everything-as-a-service (XaaS) model, data storages, virtualized infrastructure, virtualized platforms, as well as software and applications are being provided and consumed as services in the cloud. Cloud storage services can be regarded as a typical service in cloud computing, which involves the delivery of data storage as a service, including database-like services and network attached storage, often billed on a utility computing basis, e.g. per gigabyte per month. Examples include Amazon SimpleDB[1], Microsoft Azure storage[2], and so on. By using the cloud storage services, the customers can access data stored in a cloud anytime and anywhere, using any device, without caring about a large amount of capital investment when deploying the underlying hardware infrastructures. To meet the promise of ubiquitous 24/7 access, the cloud service provider (CSP) stores data replicas on multiple geo graphically distributed servers. A key problem of using the replication technique in clouds is that it is very expensive to achieve strong consistency on a worldwide scale, where a user is ensured to see the latest updates.

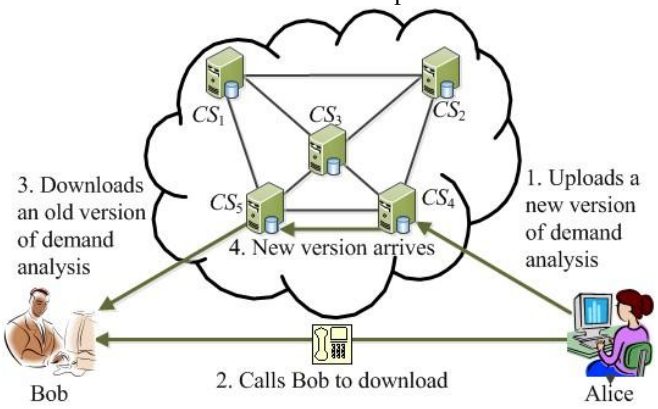

Fig.1 An application that requires causal Consistency

Actually, mandated by the CAP principle[3], many CSPs (e.g., Amazon S3) only ensure weak consistency, such as eventual consistency, for performance and high availability, where a user can read stale data for a period of time. The domain name system (DNS) is one of the most popular applications that implement eventual consistency. Updates to a name will not be visible immediately, but all clients are ensured to see them eventually.

However, eventual consistency is not a catholicon for all applications. Especially for the interactive applications, stronger consistency assurance is of increasing importance. Consider the following scenario as shown in Fig. 1. Suppose that Alice and Bob are cooperating on a project using a cloud storage service, where all of the related data is replicated to five cloud servers, CS_1, CS_5.

After uploading a new version of the requirement analysis to a CS_4, Alice calls Bob to download the latest version for integrated design. Here, after Alice calls Bob, the causal relationship is established between Alice's update and Bob's read. Therefore, the cloud should provide causal consistency,

which ensures that Alice's update is committed to all of the replicas before Bob's read. If the cloud provides only eventual consistency, then Bob is allowed to access an old version of the requirement analysis from CS_5. In this case, the integrated design that is based on an old version may not satisfy the real requirements of customers.

Actually, different applications have different consistency requirements. For example, mail services need monotonic-read consistency and read-your-write consistency, but social network services need causal consistency. In cloud storage, consistency not only determines correctness but also the actual cost per transaction. In this paper, we present a novel *consistency as a service* (CaaS) model for this situation. The CaaS model consists of a large *data cloud* and multiple small *audit clouds*. The data cloud is maintained by a CSP, and an audit cloud consists of a group of users that cooperate on a job, e.g. a document or a project. A service level agreement (SLA) will be engaged between the data cloud and the audit cloud, which will stipulate what level of consistency the data cloud should provide, and how much (monetary or otherwise) will be charged if the data cloud violates the SLA.

The implementation of the data cloud is opaque to all users due to the virtualization technique. Thus, it is hard for the users to verify whether each replica in the data cloud is the latest one or not. Inspired by the solution in, we allow the users in the audit cloud to verify cloud consistency by analysing a trace of interactive operations. Unlike their work, we do not require a global clock among all users for total ordering of operations. A loosely synchronized clock is suitable for our solution. Specifically, we require each user to maintain a logical vector for partial ordering of operations, and we adopt a two-level auditing structure: each user can perform *local auditing* independently with a local trace of operations; periodically, an auditor is elected from the audit cloud to perform *global auditing* with a global trace of operations. Local auditing focuses on monotonic-read and read-your-write consistencies, which can be performed by a light-weight online algorithm. Global auditing focuses on causal consistency, which is performed by constructing a directed graph. If the constructed graph is a directed acyclic graph (DAG), we claim that causal consistency is preserved. We quantify the severity of violations by two metrics for the CaaS model: commonality of violations and staleness of the value of a read, as in [9]. Finally, we propose a *heuristic auditing strategy* (HAS) which adds appropriate reads to reveal as many violations as possible. Our key contributions are as follows:

1) We present a novel consistency as a service (CaaS) model, where a group of users that constitute an audit cloud can verify whether the data cloud provides the promised level of consistency or not.
2) We propose a two-level auditing structure, which only requires a loosely synchronized clock for ordering operations in an audit cloud.
3) We design algorithms to quantify the severity of violations with different metrics.

4) We devise a heuristic auditing strategy (HAS) to reveal as many violations as possible. Extensive experiments were performed using a combination of simulations and a real cloud deployment to validate HAS.

The remainder of this paper is organized as follows: We introduce related work in Section II and present preliminaries in Section III. We describe verification algorithms for the two-level auditing structure in Section IV, before we provide algorithms to quantify the severity of violations in Section V. After we propose a heuristic auditing strategy to reveal as many violations as possible in Section VI, we conduct experiments to validate the heuristic auditing strategy in Section VII. Finally, we provide additional discussion in Section VIII and conclude this paper in Section IX.

II. RELATED WORK

A cloud is essentially a large-scale distributed system where each piece of data is replicated on multiple geographically- distributed servers to achieve high availability and high performance. Thus, we first review the consistency models in distributed systems. As a standard textbook, proposed two classes of consistency models: data-centric consistency and client-centric consistency. Data-centric consistency model considers the internal state of a storage system, i.e. how updates flow through the system and what guarantees the system can provide with respect to updates. However, to a customer, it really does not matter whether or not a storage system internally contains any stale copies. As long as no stale data is observed from the client's point of view, the customer is satisfied. Therefore, client-centric consistency model concentrates on what specific customers want, i.e., how the customers observe data updates. Their work also describes different levels of consistency in distributed systems, from strict consistency to weak consistency. High consistency implies high cost and reduced availability. States that strict consistency is never needed in practice, and is even considered harmful. In reality, mandated by the CAP protocol many distributed systems sacrifice strict consistency for high availability.

Then, we review the work on achieving different levels of Consistency in a cloud investigated the consistency properties provided by commercial clouds and made several useful observations. Existing commercial clouds usually re-strict strong consistency guarantees to small datasets (Google's Megastore and Microsoft's SQL Data Services), or provide only eventual consistency (Amazon's simpleDB and Google's BigTable) described several solutions to achieve different levels of consistency while deploying database applications on Amanzon S3. In the consistency requirements vary over time depending on actual availability of the data, and the authors provide techniques that make the system dynamically adapt to the consistency level by monitoring the state of the data. Proposed a novel consistency model that allows it to automatically adjust the consistency levels for different semantic data.

Finally, we review the work on verifying the levels of consistency provided by the CSPs from the users' point of view. Existing solutions can be classified into trace-based verifications and benchmark-based verifications. Trace-based verifications focus on three consistency semantics: safety, regularity, and atomicity, which are proposed by Lamport and extended by Aiyer et al. A register is safe if a read that is not concurrent with any write returns the value of the most recent write, and a read that is concurrent with a write can return any value. A register is regular if a read that is not concurrent with any write returns the value of the most recent write, and a read that is concurrent with a write

returns either the value of the most recent write, or the value of the concurrent write.

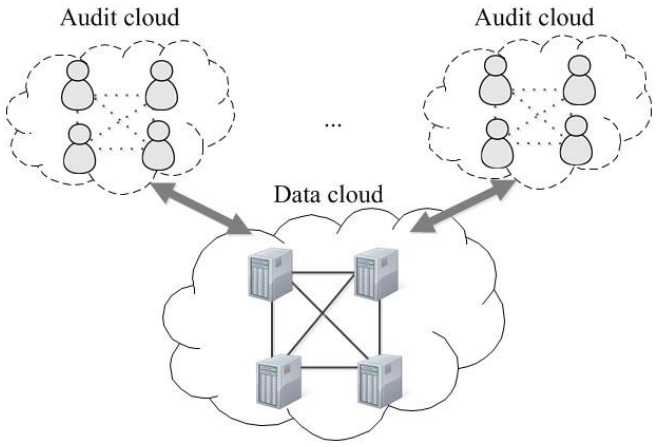

Fig 2. Consistency as a service model.

A register is atomic if every read returns the value of the most recent write. Misra is the first to present an algorithm for verifying whether the trace on a read/write register is atomic. Following his work, proposed offline algorithms for verifying whether a key-value storage system has safety, regularity, and atomicity properties by constructing a directed graph. Proposed an online verification algorithm by using the GK algorithm and used different metrics to quantify the severity of violations. The main weakness of the existing trace-based verifications is that a global clock is required among all users. Our solution belongs to trace-based verifications. However, we focus on different consistency semantics in commercial cloud systems, where a loosely synchronized clock is suitable for our solution.

Benchmark-based verifications focus on benchmarking staleness in a storage system. Both and evaluated consistency in Amazon's S3, but showed different results. used only one user to read data in the experiments, and showed that few inconsistencies exist in S3. Used multiple geographically-distributed users to read data, and found that S3 frequently violates monotonic-read consistency. The results of justify our two-level auditing structure. Presents a client-centric benchmarking methodology for understanding eventual consistency in distributed key- value storage systems. Assessed Amazon, Google, and Microsoft's offerings, and showed that, in Amazon S3, consistency was sacrificed and only a weak consistency level known as, eventual consistency was achieved.

III. PRELIMINARIES

In this section, we first illustrate the consistency as a service (CaaS) model. Then, we describe the structure of *the user operation table (*UOT), with which each user records his operations. Finally, we provide an overview of the two-level auditing structure and related definitions.

A. Consistency as a Service (CaaS) Model

As shown in Fig. 2, the CaaS model consists of a *data cloud* and multiple *audit clouds*. The data cloud, maintained by the cloud service provider (CSP), is a key-value data storage system where each piece of data is identified by a unique key. To provide always-on services, the CSP replicates all of the data on multiple geographically distributed cloud servers. An audit cloud consists of a group of users that cooperate on a job, e.g., a document or a program. We assume that each user in the audit cloud is identified by a unique ID. Before outsourcing the job to the data cloud, the audit cloud and the data cloud will engage in a service level agreement (SLA), which stipulates the promised level of consistency that should be provided by the data cloud. The audit cloud exists to verify whether the data cloud violates the SLA or not, and to quantify the severity of violations.

In our system, a two-level auditing model is adopted: each user records his operations in a user operation table (UOT), which is referred to as a local trace of operations in this paper. Local auditing can be performed independently by each user with his own UOT; periodically, an auditor is elected from the audit cloud. In this case, all other users will send their UOTs to the auditor, which will perform global auditing with a global trace of operations. We simply let each user become an auditor in turn, and we will provide a more comprehensive solution in Section VIII. The dotted line in the audit cloud means that users are loosely connected. That is, users will communicate to exchange messages after executing a set of reads or writes, rather than communicating immediately after executing every operation. Once two users finish communicating, a causal relationship on their operations is established.

B. User Operation Table (UOT)

Each user maintains a UOT for recording local operations. Each record in the UOT is described by three elements: *operation*, *logical vector*, and *physical vector*. While issuing an operation, a user will record this operation, as well as his current logical vector and physical vector, in his UOT.

Each operation *op* is either a write $W(K, a)$ or a read $R(K, a)$, where $W(K, a)$ means writing the value *as* to data that is identified by key K, and $R(K, a)$ means reading data that is identified by key K and whose value is a. As in [7], we call $W(K, a)$ as $R(K, a)$'s *dictating write*, and $R(K, a)$ as $W(K, a)$'s *dictated read*. We assume that the value of each write is unique. This is achieved by letting a user attach his ID, and current vectors to the value of write. Therefore, we have the following properties: (1) a read must have a unique dictating write. A write may have zero or more dictated reads. (2) From the value of a read, we can know the logical and physical vectors of its dictating write.

C. Overview of Two-Level Auditing Structure

Vogel's investigated several consistency models provided by commercial cloud systems. Following their work, we provide a two-level auditing structure for the CaaS model. At the first level, each user independently performs local auditing with his own UOT. The following consistencies (also referred to as local consistencies) should be verified at this level:

Monotonic-read consistency. If a process reads the value of data K, any successive reads on data K by that process will always return that same value or a more recent value.

Read-your-write consistency. The effect of a write by a process on data K will always be seen by a successive read on data K by the Read-your-write consistency. The effect of a write by a process on data K will always be seen by a successive read on data K by the same process.

Causal consistency. Writes that are causally related must be seen by all processes in the same order.

Concurrent writes may be seen in a different order on different machines.

IV. VERIFICATION OF CONSISTENCY PROPERTIES

In this section, we first provide the algorithms for the two-level auditing structure for the CaaS model, and then analyze their effectiveness. Finally, we illustrate how to perform a garbage collection on UOTs to save space. Since the accesses of data with different keys are independent of each other, a user can group operations by key and then verify whether each group satisfies the promised level of consistency. In the remainder of this paper, we abbreviate read operations with R (a) and write operations with W (a).

A. Local Consistency Auditing

Local consistency auditing is an online algorithm (Alg. 1). In Alg. 1, each user will record all of his operations in his UOT. While issuing a read operation, the user will perform local consistency auditing independently.

Let R(a) denote a user's current read whose dictating write is W (a), W (b) denote the last write in the UOT, and R(c) denote the last read in the UOT whose dictating write is W (c). Read-your-write consistency is violated if W (a) happens before W (b), and monotonic-read consistency is violated if W (a) happens before W (c). Note that, from the value of a read, we can know the logical vector and physical vector of its dictating write. Therefore, we can order the dictating writes by their logical vectors.

B. Global Consistency Auditing

Global consistency auditing is an offline algorithm periodically; an auditor will be elected from the audit cloud to perform global consistency auditing. In this case, all other users will send their UOTs to the auditor for obtaining a global trace of operations.

C. Effectiveness

The effectiveness of the local consistency auditing algorithm is easy to prove. For monotonic-read consistency, a user is required to read either the same value or a newer value. There- fore, if the dictating write of a new read happens before the dictating write of the last read, we conclude that monotonic- read consistency is violated. For read-your-write consistency, the user is required to read his latest write. Therefore, if the dictating write of a new read happens before his last write, we conclude that read-your-write consistency is violated.

D. Garbage Collection

In the auditing process, each user should keep all operations in his UOT. Without intervention, the size of the UOT would grow without bound. Furthermore, the communication cost for transferring the UOT to the auditor will be excessive. Therefore, we should provide a garbage collection mechanism which can delete unneeded records, while preserving the effectiveness of auditing.

V. QUANTIFYING SEVERITY OF VIOLATIONS

As we provide two metrics to quantify the severity of violations for the CaaS model: commonality and staleness. Commonality quantifies how often the violations happen. Staleness quantifies how much older the value of a read is compared to that of the latest write. Staleness can be further classified into time-based staleness and operation-based staleness, where the former counts the passage of time, and the latter counts the number of intervening operations, between the read's dictating write and the latest write.

VI. HEURISTIC AUDITING STRATEGY

From the auditing process in the CaaS model, we observe that only reads can reveal violations by their values. Therefore, the basic idea of our heuristic auditing strategy (HAS) is to add appropriate reads for revealing as many violations as possible. We call these additional reads auditing reads.

HAS divides physical time into L timeslice, where l timeslice constitute an interval. Each timeslice is associated with a state, which can be marked with either normal or abnormal. A normal state means that there is no consistency violation, and an abnormal state means that there is one violation in this timeslice.

HAS determines the number of auditing reads in the (i +1)- th interval, based on the number of abnormal states in the i-th interval. Let ni denote the number of auditing reads in interval i. HAS determines ni+1, which is the number of auditing reads in the next interval with Eq. 1:

$$n_{i+1} = \min(l, k \times n_i), \quad n_i \geq \alpha \qquad (1)$$

$$n_{i+1} = \max(1, \tfrac{1}{k} \times n_i), \quad n_i < \alpha$$

where k is a parameter that is used to adjust the value of ni+1, l is the number of time slices in an interval, and α is a threshold value that is used to determine whether the number of auditing reads in the next round should be increased by k times or be reduced to 1/k, compared to the number of auditing reads in the current round.

Specifically, given a threshold value α, if a user issues ni Auditing reads and reveals more than α violations in interval i, in interval i + 1, the user will issue $n_{i+1} = \min(l, k * n_i)$ auditing reads; that is, each timeslice will be issued, at most, one auditing read, and the maximal number of auditing reads will not exceed l. Otherwise, the user will issue $n_{i+1} = \max(1, \tfrac{1}{k} \times n_i)$ auditing reads, that is, each interval will be issued at least one auditing read. Since the number of auditing reads should be an integer, $\tfrac{1}{k} \times n_i$ is actually the abbreviation of $\lfloor \tfrac{1}{k} \times n_i \rfloor$.

VII. DISCUSSION

In this section, we will discuss some additional issues about CaaS in terms of the election of an auditor and other consistency models.

A. Election of an Auditor

In section III, an auditor is simply elected from the auditor cloud in turn, where each user becomes the auditor with the same probability. However, different users have different

levels of ability in terms of available bandwidth, CPU, and Memory of clients. The users with a higher ability should have a higher probability of being selected as auditor. In this section, we provide a more comprehensive solution to elect an auditor as follows: We construct an ID ring for a group of users, where each node is associated with a node ID, and each user is denoted by a set of nodes in the ring. Suppose the number of nodes in the ring is n. To elect an auditor, we can randomly generate a number r, and let the user who is denoted by the node with an ID of (r mod n) in the ring to be the auditor. Note that the selection of each user does not have to be uniform. The number of nodes associated with a user can be determined by his abilities, e.g., the capability of his client, his trusted rank, and so on. In this way, the probability of a user with a higher ability of being chosen as the auditor becomes higher. For example, given 3 users and 6 nodes, user Alice is denoted by 3 nodes, user Bob is denoted by 2 nodes, and user Clark is denoted by 1 node. Therefore, the probability of Alice being the auditor is 50%, for Bob it is 33%, and for Clark it is 17%.

B. Other Consistency Models

In local auditing, we only consider two kinds of consistencies, i.e., monotonic-read consistency and read-your-write consistency. Now, we discuss other consistency models such as read-after-write consistency and monotonic-write consistency models. Read-after-write consistency requires that all clients immediately see new data. With read-after-write consistency, a newly created object, file, or table row will immediately be visible, without any delays. Therefore, we can build distributed systems with less latency. Today, Amazon S3 provides read- after-write consistency in the EU and US-west regions. So far, it is hard to achieve read-after-write consistency in a world- wide scale.

Monotonic-write consistency requires that a write on a copy of data item x is performed only if that copy has been updated by any preceding write operations that occurred in other copies. However, monotonic-write is not always necessary for all applications. For example, the value of x is first set to 4 and, later on, is changed to 7. The value 4 that has been overwritten isn't really important.

IX.CONCLUSION

In this paper, we presented a consistency as a service (CaaS) model and a two-level auditing structure to help users verify whether the cloud service provider (CSP) is providing the promised consistency, and to quantify the severity of the violations, if any. With the CaaS model, the users can assess the quality of cloud services and choose a right CSP among various candidates, e.g., the least expensive one that still provides adequate consistency for the users' applications. For our future work, we will conduct a thorough theoretical study of consistency models in cloud computing.

X.REFERENCES

[1] M. Armbrust, A. Fox, R. Griffith, A. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, *et al.*, "A view of cloud computing," *Commun. ACM*, vol. 53, no. 4, 2010.

[2]P. Mell and T. Grance, "The NIST definition of cloud computing (draft)," NIST Special Publication 800-145 (Draft), 2011.

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

[4] "Pushing the CAP: strategies for consistency and availability,"*Computer*, vol. 45, no. 2, 2012.

[5] M. Ahamad, G. Neiger,. Burns, P. Kohli and P. Hutto, "Causal memory: definitions, implementation, and programming," *Distributed Computing*, vol. 9, no. 1, 1995.

[6] W. Lloyd, M. Freedman, M. Kaminsky, and D. Andersen, "Don't settle

[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 HotDep*.

[8]C. Fidge, "Timestamps in message-passing systems that preserve the partial ordering," in *Proc. 1988 ACSC*.

[9]W. Golab, X. Li, and M. Shah, "Analyzing consistency properties

[10]A. Tanenbaum and M. Van Steen, Distributed Systems: Principles and Paradigms. Prentice Hall PTR, 2002.