International Journal of Computer Science & Mechatronics
A peer reviewed International Journal | Article Available online www.ijcsm.in
©smsamspublications.com | Vol.1.Issue 2.2015

Consistency as a Service: A Novel Service in Cloud for Auditing

Research Article
ISSN: 2455-1910

Sk.Noor Fathima[1], Md.Imran[2]
[1]M.Tech (CSE), Nimra Institute of Science & Technology, A.P., India.
[2]Assistant Professor, Dept. of Computer Science & Engineering, Nimra Institute of Science & Technology, A.P., India.

Abstract: A cloud provider is a company that offers some component of cloud computing – typically Infrastructure as a Service (IaaS), Software as a Service (SaaS) or Platform as a Service (PaaS) – to other businesses or individuals. Ever 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 real cloud deployments to validate HAVE.

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

I. INTRODUCTION

Ever since Cloud computing is a model for enabling ubiquitous network access to a shared pool of configurable computing resources. Cloud computing has become commercially popular, as it promises to guarantee scalability, elasticity, and high availability at a low cost [1], [2]. 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 SimpleDB1, Microsoft

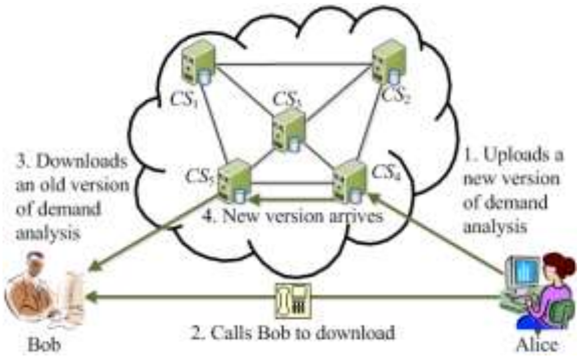

Fig. 1. *An application that requires causal consistency*.

Azure storage2 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 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, where a user is ensured to see the latest updates. Actually, mandated by the CAP principle3, 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 [5] 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 [6]. 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 [7], we allow the users in the audit cloud to verify cloud consistency by analyzing 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 [8] 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 real cloud deployments to validate HAVE.

II. Problem Statement

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. The cloud service provider (CSP) stores data replicas on multiple geographically distributed servers. 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. The replication technique in clouds is that it is very expensive to achieve strong consistency. Hard to verify replica in the data cloud is the latest one or not.

III. Related Work

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. Do not require a global clock among all users for total ordering of operations. The users can assess the quality of cloud services. Choose a right CSP. Among various candidates, e.g, the least expensive one that still provides adequate consistency for the users' applications.

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. Ref. [10], 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. Ref. [11] states that strict consistency is never needed in practice, and is even considered harmful. In reality, mandated by the CAP protocol [3], [4], many distributed systems sacrifice strict consistency for high availability. Then, we review the work on achieving different levels of consistency in a cloud. Ref. [12] investigated the consistency properties provided by commercial clouds and made several useful observations. Existing commercial clouds usually restrict strong consistency guarantees to small datasets (Google's Mega Store and Microsoft's SQL Data Services), or provide only eventual consistency (Amazon's simple DB and Google's Big Table). Ref. [13] described several solutions to achieve different levels of consistency while deploying database applications on Amazon S3. In Ref. [14], 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. Ref. [15] 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 [7], [9] and benchmark-based verifications [13]–[16]. Trace-based verifications focus on three consistency semantics: safety, regularity, and atomicity, which are proposed by Lamport [10], and extended by Aiyer et al. [11]. 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

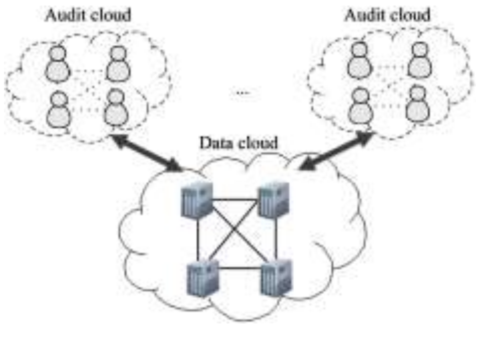

Fig. 2. Consistency as a service model.

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. A register is atomic if every read returns the value of the most recent write. Misra [2] is the first to present an algorithm for verifying whether the trace on a read/write register is atomic. Following his work, Ref. [7] proposed offline algorithms for verifying whether a key-value storage system has safety, regularity, and atomicity properties by constructing a directed graph. Ref. [9] proposed an online verification algorithm by using the GK algorithm [13], 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 [16] and [7] evaluated consistency in Amazon's S3, but showed different results. Ref. [16] used only one user to read data in the experiments, and showed that few inconsistencies exist in S3. Ref. [7] used multiple geographically-distributed users to read data, and found that S3 frequently violates monotonic-read consistency. The results of [7] justify our two-level auditing structure. Ref. [8] presents a client-centric benchmarking methodology for understanding eventual consistency in distributed key value storage systems. Ref. [1] 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.

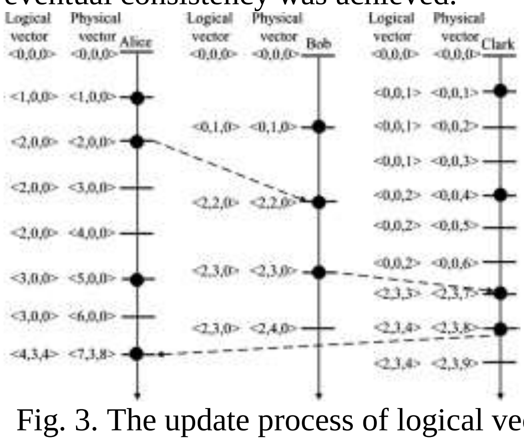

Fig. 3. The update process of logical vector and physical vector.

A black solid circle denotes an event (read/write/send message/receive message), and the arrows from top to bottom denote the increase of physical time. The logical vector is updated via the *vector clocks* algorithm

[8]. The physical vector is updated in the same way as the logical vector, except that the user's physical clock keeps increasing as time passes, no matter whether an *event* (read/write/send message/receive message) happens or not. The update process is as follows: All clocks are initialized with zero (for two vectors); The user increases his own physical clock in the physical vector continuously, and increases his own logical clock in the logical vector by one only when an event happens; Two vectors will be sent along with the message being sent. When a user receives a message, he updates *each element* in his vector with the maximum of the value in his own vector and the value in the received vector (for two vectors). **Monotonic-read consistency.** *If a process reads the value of data K, any successive reads on data K by that process will*

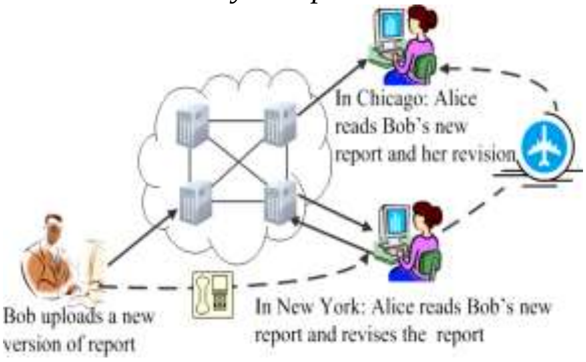

Fig. 4. An application that has different consistency requirements.

```
Algorithm 1 Local consistency auditing
  Initial UOT with ∅
  while issue an operation op do
    if op = W(a) then
      record W(a) in UOT
    if op = r(a) then
      W(b) ∈ UOT is the last write
      if W(a) → W(b) then
        Read-your-write consistency is violated
      R(c) ∈ UOT is the last read
      if W(a) → W(c) then
        Monotonic-read consistency is violated
      record r(a) in UOT
```

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 same process. Intuitively, monotonic-read consistency requires that a user must read either a newer value or the same value, and read your-write consistency requires that a user always reads his latest updates. To illustrate, let us consider the example in Fig.4. Suppose that Alice often commutes between New York and Chicago to work, and the CSP maintains two replicas on cloud servers in New York and Chicago, respectively, to provide high availability. In Fig. 4, after reading Bob's new report and revising this report in New York, Alice moves to Chicago. Monotonic-read consistency requires that, in Chicago, Alice must read Bob's new version, i.e., the last update she ever saw in New York must have been propagated to the server in Chicago. Read-your-write consistency requires that, in Chicago, Alice must read her revision for the new report, i.e., her own last update issued in New York must have been propagated to the server in Chicago. The above models can be combined. The users can choose a subset of consistency models for their applications.

IV. Conclusion

In this paper, 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 .We have 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. For our future work, we will conduct a thorough theoretical study of consistency models in cloud computing.

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, J. 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 for eventual: scalable causal consistency for wide-area storage with COPS," in *Proc. 2011 ACM SOSP*.

[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 for fun and profit," in *Proc. 2011 ACM PODC*.

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

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

[12] ——, "Eventually consistent," *Commun. ACM*, vol. 52, no. 1, 2009.

[13] M. Brantner, D. Florescu, D. Graf, D. Kossmann, and T. Kraska, "Building a database on S3," in *Proc. 2008 ACM SIGMOD*.

[14] T. Kraska, M. Hentschel, G. Alonso, and D. Kossmann, "Consistency rationing in the cloud: pay only when it matters," in *Proc. 2009 VLDB*.

[15] S. Esteves, J. Silva, and L. Veiga, "Quality-of-service for consistency of data geo-replication in cloud computing," *Euro-Par 2012 Parallel Processing*, vol. 7484, 2012.

[16] H. Wada, A. Fekete, L. Zhao, K. Lee, and A. Liu, "Data consistency properties and the trade-offs in commercial cloud storages: the consumers' perspective," in *Proc. 2011 CIDR*.