IJREAT International Journal of Research in Engineering & Advanced Technology, Volume 2, Issue 2, Apr-May, 2014
ISSN: 2320 – 8791 (Impact Factor: 1.479)
www.ijreat.org

Secured Consistent Transactions

Rakesh A[1], Saravanan K[2], Hari Prasath L[3]

[1, 2]Student, Department of Information Technology, Anand Institute of Higher Technology, Chennai

[3]Assistant Professor, Department of Information Technology, Anand Institute of Higher Technology, Chennai

Abstract

In distributed transactional database systems deployed over cloud servers, entities cooperate to form proofs of authorizations that are justified by collections of certified credentials. These proofs and credentials may be evaluated and collected over extended time periods under the risk of having the underlying authorization policies or the user credentials being in inconsistent states. It therefore becomes possible for policy-based authorization systems to make unsafe decisions that might threaten sensitive resources. In this paper, we highlight the criticality of the problem. We then define the notion of trusted transactions when dealing with proofs of authorizations. Accordingly, we propose several increasingly-stringent levels of policy consistency constraints, and present different enforcement approaches to guarantee the trustworthiness of transactions executing on cloud servers. We propose a Two-Phase Validation Commit protocol as a solution, which is a modified version of the basic Two-Phase Commit protocols. We finally analyze the different presented approaches using both analytical evaluation of the overheads and simulations to guide the decision makers to which approach to use.

Index Terms—*Cloud databases, authorization policies, consistency, distributed transactions, atomic commit protocol*

1. Introduction

Research in cloud computing is receiving distributed process of transacting database systems deployed over cloud servers, entities work to form proofs of authorizations that are justified by collections of certified evidence of authority. These proofs and status may be evaluated and collected over extended time periods under the risk of having the underlying authorization policies or the user confidence being in lacking agreement states. In this paper, we highlight the criticality of the problem. We then define the general understanding of trusted transactions when dealing with proofs of authorizations. In cloud computing, users can outsource their computation and storage to servers (also called clouds) using Internet. Clouds can provide several types of services like applications (e.g., Google Apps, Microsoft online,). Much of the data stored in clouds is highly sensitive, Security and privacy are thus very important issues in cloud computing.

In one hand, the user should authenticate itself before initiating any transaction, and on the other hand, it must be ensured that the cloud does not tamper with the data that is outsourced. User privacy is also required so that the cloud or other users do not know the identity of the user. The cloud can hold the user accountable for the data it outsources, and likewise, the cloud is itself accountable for the services it provides. Existing work on access control in cloud are authorization in nature. we must also handle two types of security remark conditions. First, the system may suffer from policy inconsistencies during policy updates due to the informal agreement model fundamental most cloud services. For example, it is possible for several versions of the policy to be observed at multiple position within a single transaction, leading to inconsistent (and likely unsafe)access decisions during the transaction. Second, it is possible for external factors to cause user credential inconsistencies (evidence of authority) over the lifetime of a transaction. For instance, a user's login evidence of authority could be invalidated or to bring after collection by the authorization server, but before the act of completing of the transaction.

We propose several increasingly strict levels of policy agreement restriction, and present different the act approaches to guarantee the confidence of transactions executing on cloud servers. In the proposed scheme, the cloud verifies the authenticity of the server without knowing the user's identity. We propose a Two-Phase Validation Commit protocol as a solution, which is a modified version of the basic Two-Phase Commit protocols. We finally analyze the different presented approaches using both analytical Evaluation of the overheads and enactment to guide the decision makers to which approach to use. We then present a more general term, unauthorization person not able to access policy and credential consistency, our scheme is a safe transaction, that identifies transactions that are both trusted and conform to the ACID properties of distributed database systems.

I.a User Enrollment

Users have an initial level Registration Process at the web end. The users provide their own personal information for this process. The server in turn stores the information in its database.

I.b Transaction Manager

TM first sends a Prepare to-Validate message to each participant server. In response to this message, each participant evaluates the proofs for each query of the transaction using the latest policies it has available and sends a reply back to the TM containing the truth value (True/False) of those proofs along with the version number and policy identifier for each policy used. Once the TM receives the replies from all the participants, it moves on to the validation phase. TM sends out a Prepare-to-Commit message for a transaction, The yes or no reply for the satisfaction of integrity constraints as in 2PC, the true or false reply for the satisfaction of the proofs of authorizations as in 2PV, and the version number of the policies used to build the proofs as in 2PV. It is similar to that of 2PV with the exception of handling the yes or no reply for integrity constraint validation and having a decision of commit rather than continue. The TM enforces the same behavior as 2PV in identifying policies inconsistencies and sending the Update messages.

I.c Unauthorized Person

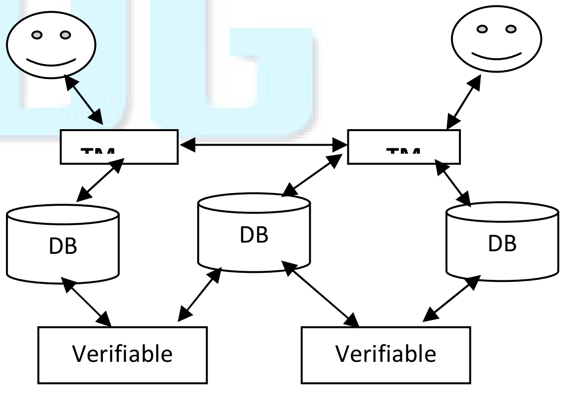
If come unauthorized person change the version and policy, the TM needs to check the version number it receives from each server with that of the very first participating server. If they are different, the transaction aborts due to a consistency violation. At commit time, all the proofs will have been generated with consistent policies and only 2PC is invoked. TM needs to validate the policy versions used against the latest policy version known by the master policies server to decide whether to abort or not. At commit time, 2PVC is invoked by the TM to check the data integrity constraints and verify that master policies server has not received any newer policy versions. Continuous proofs invoke 2PV at the execution of each query which will update the older policies with the new policy and re-evaluate.

2. System Assumptions and Problem Definition

A. System Model

We assume a cloud infrastructure consisting of a set of S servers, where each server is responsible for hosting a subset D of all data items D belonging to a specific application domain (D ⊂ D). Users interact with the system by submitting queries or update requests encapsulated in ACID transactions.

A transaction is submitted to a Transaction Manager (TM) that coordinates its execution. Multiple TMs could be invoked as the system workload increases for load balancing, but each transaction is handled by only one TM. We denote each transaction as T = q1, q2, . . . , qn, where qi ϵ Q is a single query/update belonging to the set of all queries Q. The start time of each transaction is denoted by α(T), and the time at which the transaction finishes execution and is ready to commit is denoted by ω(T). We assume that queries belonging to a transaction execute sequentially, and that a transaction does not fork sub-transactions. These assumptions simplify our presentation, but do not affect the correctness or the validity of our consistency definitions. Let P denote the set of all authorization policies, and let Psi (D) denote the policy that server si uses to protect data item D. We represent a policy P as a mapping P : S × 2D → 2R ×A×N that associates a server and a set of data items with



a set of inference rules from the set R, a policy administrator from the set A, and a version number. We denote by C the set of all credentials, which are issued by the Certificate Authorities (CAs) within the system. We assume that each CA offers an online method that allows any server to check the current status of credentials that it has issued. Given a credential ck ϵ C, α(ck) and ω(ck) denote issue and expiration times of ck, respectively. Given a function m : Q → 2D that identifies the data items accessed by a particular query, a proof of authorization for query qi evaluated at server s j at time tk is a tuple (qi, s j, Ps j (m(qi)), tk,C), where C is the set of credentials presented by the querier to satisfy Ps j (m(qi)). In this paper, we use the function eval : F ×TS →B to denote whether a proof f ϵ F is valid at time t ϵTS.

B. Problem Definition

Since transactions are executed over time, the state information of the credentials and the policies enforced by different servers are subject to changes at any time instance, therefore it becomes important to introduce precise definitions for the different consistency levels that could be achieved within a transactions lifetime. These consistency models strengthen the trusted transaction definition by defining the environment in which policy versions are consistent relative to the rest of the system. Before we do that, we define a transaction's view in terms of the different proofs of authorizations evaluated during the lifetime of a particular transaction.

Definition 1: (View) A transaction's view VT is the set of proofs of authorizations observed during the lifetime of a transaction [α(T), ω(T)] and defined as VT = { fsi | fsi = (qi, si, Psi (m(qi)), ti,C) ∧ qi ∈ T}. Following from Def. 1, a transaction's view is built incrementally as more proofs of authorizations are being evaluated by servers during the transaction execution. We now present two increasingly more powerful definitions of consistencies within transactions.

Definition 2: (View Consistency) A view VT = {(qi, si, Psi (m(qi)), ti,C), . . . , (qn, sn, Psn (m(qn)), tn,C)} is view consistent, or φ-consistent, if VT satisfies a predicate φ-consistent that places constraints on the versioning of the policies such that φ-consistent(VT) ↔ ∀i, j : ver(Psi) = ver(Ps j) for all policies belonging to the same administrator A, where function ver is defined as ver : P → N.

With a view consistency model, the policy versions should be internally consistent across all servers executing the transaction. The view consistency model is weak in that the policy version agreed upon by the subset of servers within the transaction may not be the latest policy version v. It may be the case that a server outside of the S servers has a policy that belongs to the same administrative domain and with a version v > v. A more strict consistency model is the global consistency and is defined as follows.

Definition 3: (Global Consistency) A view VT = {(qi, si, Psi (m(qi)), ti,C), . . . , (qn, sn, Psn (m(qn)), tn,C)} is global consistent, or ψ-consistent, if VT satisfies a predicate ψ-consistent that places constraints on the versioning of the policies such that ψ-consistent(VT) ↔ ∀i : ver(Psi) = ver(P) for all policies belonging to the same administrator A, and function ver follows the same aforementioned definition, while ver(P) refers to the latest policy version.
With a global consistency model, policies used to evaluate the proofs of authorizations during a transaction execution among S servers should match the latest policy version among the entire policy set P,for all policies enforced by the same administrator A. Given the above definitions, we now have a precise vocabulary for defining the conditions necessary for a transaction to be asserted as "trusted".

Definition 4: (Trusted Transaction) Given a transaction T = {q1, q2, . . . , qn} and its corresponding view VT , T is trusted iff ∀fsi ∈VT : eval(fsi , t), at some time instance t : α(T) ≤ t ≤ ω(T) ∧ (φ-consistent(VT) ∨ ψ-consistent(VT)).
Finally, we say that a transaction is safe if it is a trusted transaction that also satisfies all data integrity constraints imposed by the database management system. A safe transaction is allowed to commit, while an unsafe transaction is forced to rollback.

3. Trusted Transaction Enforcement

A. Deferred Proofs of Authorization

Definition 5: Deferred proofs present an optimistic approach with relatively weak authorization guarantees. The proofs of authorizations are evaluated simultaneously only at commit time (using either view or global consistency from Defs. 2 and 3) to decide whether the transaction is trusted.

B. Punctual Proofs of Authorization

Definition 6: Punctual proofs present a more proactive approach in which the proofs of authorizations are evaluated instantaneously whenever a query is being handled by a server. This facilitates early detections of unsafe transactions which can save the system from going into expensive undo operations. All the proofs of authorizations are then re-evaluated at commit time to ensure that policies were not updated during the transaction in a way that would invalidate a previous proof, and that credentials were not invalidated.

C. Incremental Punctual Proofs of Authorization

Before we define the Incremental Punctual proofs of authorization approach, we define a view instance, which is a view snapshot at a specific time instance.

4. Implementing Safe Transactions

A. Two-Phase Validation Algorithm

A common characteristic of most of our proposed approaches to achieve trusted transactions is the need for policy consistency validation at the end of a transaction. That is, in order for a trusted transaction to commit, its TM has to enforce either view or global consistency among the servers participating in the transaction. Toward this, we propose a new algorithm called Two-Phase Validation (2PV).

Algorithm 1: Two-Phase Validation - 2PV(TM)

```
1.  Send "Prepare-to-Validate" to all participants
2.  Wait for all replies (a True/False, and a set of
    policy versions for each unique policy)
3.  Identify the largest version for all unique
    policies
4.  If all participants utilize the largest version for
    each unique policy
5.  If any responded False
6.  ABORT
7.  Otherwise
8.  CONTINUE
9.  Otherwise, for all participants with old
    versions of policies
10. Send "Update" with the largest version
    number of each Policy
11. Goto 2.
```

2PV operates in two phases: collection and validation. During collection, the TM first sends a Prepare to-Validate message to each participant server. In response to this message, each participant (1) evaluates the proofs for each query of the transaction using the latest policies it has available and (2) sends a reply back to the TM containing the truth value (TRUE/FALSE) of those proofs along with the version number and policy identifier for each policy used. Further, each participant keeps track of its reply (i.e., the state of each query) which includes the id of the TM (TMid), the id of the transaction (Tid) to which the query belongs, and a set of policy versions used in the query's authorization (vi, pi).
Once the TM receives the replies from all the participants, it moves on to the validation phase. If all polices are consistent, then the protocol honors the truth value where any FALSE causes an ABORT decision and all TRUE causes a CONTINUE decision. In the case of inconsistent policies, the TM identifies the latest policy and sends an Update message to each out-of-date participant with a policy identifier and returns to the collection phase. In this case, the participants (1) update their policies, (2) re evaluate the proofs and (3) send a new reply to the TM. Algorithm 1 shows the process for the TM.
In the case of view consistency (Def. 2), there will be at most two rounds of the collection phase. A participant may only be asked to re-evaluate a query using a newer policy by an Update message from the TM after one collection phase.
For the global consistency case (Def. 3), the TM retrieves the latest policy version from a master policies server (Step 2) and use it to compare against the version numbers of each participant (Step 3). This master version may be retrieved only once or each time Step 3 is invoked. For the former case, collection may only be executed twice as in the case of view consistency. In the latter case, if the TM retrieves the latest version every round, global consistency may execute the collection many times. This is the case if the policy is updated during the round. While the number of rounds are theoretically infinite, in a practical setting, this should occur infrequently.

B. Two-Phase Validate Commit Algorithm

The 2PV protocol enforces trusted transactions, but does not enforce not safe transactions because it does not validate any integrity constraints. Since the Two-Phase Commit atomic protocol (2PC) commonly used to enforce integrity constraints has similar structure as

2PV, we propose integrating these protocols into a Two-Phase Validation Commit (2PVC) protocol.

Algorithm 2: Two-Phase Validation Commit - 2PVC (TM)

```
1.  Send"Prepare-to-Commit" to all participants
2.  Wait for all replies (Yes/No, True/False, and a
    set of policy versions for each unique policy)
3.  If any participant replied No for integrity
    check
4.  ABORT
5.  Identify the largest version for all unique
    policies
6.  If all participants utilize the largest version for
    each unique policy
7.  If any responded False
8.  ABORT
9.  Otherwise
10. COMMIT
11. Otherwise, for participants with old policies
12. Send "Update" with the largest version
    number of each policy
13. Wait for all replies
14. Goto 5.
```

2PVC can be used to ensure the data and policy consistency requirements of safe transactions. Specifically, 2PVC will evaluate the policies and authorizations within the first, voting phase. That is, when the TM sends out a Prepare-to-Commit message for a transaction, the participant server has three values to report: (1) the YES or NO reply for the satisfaction of integrity constraints as in 2PC, (2) the TRUE or FALSE reply for the satisfaction of the proofs of authorizations as in 2PV, and (3) the version number of the policies used to build the proofs (vi, pi) as in 2PV. The process given in Algorithm 2 is for the TM under view consistency. It is similar to that of 2PV with the exception of handling the YES or NO reply for integrity constraint validation and having a decision of COMMIT rather than CONTINUE. The TM enforces the same behavior as 2PV in identifying policies inconsistencies and sending the Update messages. The same changes to 2PV can be made here to provide global consistency by consulting the master policies server for the latest policy version (Step 5).

C. Using 2PV & 2PVC in Safe Transactions

2PV and 2PVC can be used to enforce each of the consistency levels defined in Sec. 3. Deferred and Punctual (Defs. 5 and 6) proofs are roughly the same. The only difference is that Punctual will return proof evaluations upon executing each query. Yet, this is done on a single server, and therefore, does not need 2PVC or 2PV to distribute the decision. To provide for trusted transactions, both require at commit time evaluation at all participants using 2PVC.

The TM needs to check the version number it receives from each server with that of the very first participating server. If they are different, the transaction aborts due to a consistency violation. At commit time, all the proofs will have been generated with consistent policies and only 2PC is invoked. In the global consistency case, the TM needs to validate the policy versions used against the latest policy version known by the master policies server to decide whether to abort or not. At commit time, 2PVC is invoked by the TM to check the data integrity constraints and verify that master policies server has not received any newer policy versions.

5. Evaluations

Environment and Setup

We used Java to implement each proof approach described in Sec. 3 with support for both view and global consistency. Although the approaches were implemented in their entirety, the underlying database and policy enforcement systems were simulated with parameters. To understand the performance implications of the different approaches, we varied the (i) protocol used, (ii) level of consistency desired, (iii) frequency of master policy updates, (iv) transaction length, and (v) number of servers available.

Our experimentation framework consists of three main components: a randomized transaction generator, a master policy server that controls the propagation of policy updates, and an array of transaction processing servers. Our experiments were run within a research lab consisting of 38 Apple Mac Mini computers. These machines were running OS X 10.6.8 and had 1.83 GHz Intel Core Duo processors coupled with 2GB of RAM. All machines were connected to a gigabit ethernet LAN with average round trip times of 0.35 ms. All WAN experiments were also conducted within this testbed by artificially delaying packet transmission by an additional 75 ms.

6. External Interface Requirements

A. Interfaces User

1. All the contents in the project are implemented JSP and Servlets.
2. Every conceptual part of the projects is reflected using Servlets Classes.
3. System gets the input and delivers through the GUI based.

B. Hardware Interfaces

Ethernet

Ethernet on the AS/400 supports TCP/IP, Advanced Peer-to-Peer Networking (APPN) and advanced program-to-program communications (APPC).

ISDN

You can connect your AS/400 to an Integrated Services Digital Network (ISDN) for faster, more accurate data transmission. An ISDN is a public or private digital communications network that can support data, fax, image, and other services over the same physical interface. Also, you can use other protocols on ISDN, such as IDLC and X.25.

C. Software Interfaces

This software is interacted with the TCP/IP protocol, Socket and listening on unused ports. Server Socket and listening on unused ports and JDK 1.6
D. Communications Interfaces
1. TCP/IP protocol.
2. LAN Settings.

7. Other Nonfunctional Requirements

A. Performance Requirements

The performance of the wireless sensor network, to execute this project on LAN or wifi communication channel . So we need to one or more than machine to execute the demo. Machine needs the enough hard disk space to install the software and run our project.

B. Safety Requirement

1. The software may be safety-critical. If so, there are issues associated with its integrity level.
2. The software may not be safety-critical although it forms part of a safety-critical system. For example, software may simply log transactions.
3. If a system must be of a high integrity level and if the software is shown to be of that integrity level, then the hardware must be at least of the same integrity level.
4. There is little point in producing 'perfect' code in some language if hardware and system software (in widest sense) are not reliable.
5. If a computer system is to run software of a high integrity level then that system should not at the same time accommodate software of a lower integrity level.
6. Systems with different requirements for safety levels must be separated.
7. Otherwise, the highest level of integrity required must be applied to all systems in the same environment.

C. Security Requirements

Do not block the some available ports through the windows firewall.

D. Software Quality Attributes

1. Functionality are the required functions available, including interoperability and security.
2.Reliability maturity, fault tolerance and recoverability.
3.Usability how easy it is to understand, learn, and operate the software system.
4.Efficiency performance and resource behavior.
5.Maintainability Maintaining the software.
6.Portability can the software easily be transferred to another environment, including install ability.

8. Conclusions

Despite the popularity of cloud services and their wide adoption by enterprises and governments, cloud providers still lack services that guarantee both data and access control policy consistency across multiple data centers. Here we identified several consistency problems that can arise during cloud-hosted transaction processing using weak consistency models, particularly if policy-based authorization systems are used to enforce access controls. To this end, we developed a variety of light-weight proof enforcement and consistency models i.e., Deferred, Punctual, Incremental, and Continuous proofs, with view or global consistency that can enforce increasingly strong protections with minimal runtime overheads.

We used simulated workloads to experimentally evaluate implementations of our proposed consistency models relative to three core metrics:transaction processing performance, accuracy (i.e., global vs. view consistency and recency of policies used), and precision (level of agreement among transaction participants). We found that high performance comes at a cost: Deferred and Punctual proofs had minimal overheads, but failed to detect certain types of consistency problems. On the other hand, high accuracy models (i.e., Incremental and Continuous) required higher code complexity to implement correctly, and had only moderate performance when compared to the lower accuracy schemes. To better explore the differences between these approaches, we also carried out a trade-off analysis of our schemes to illustrate how application-centric requirements influence the applicability of the eight protocol variants explored in this article.

References

[1] M. Armbrust *et al.*, "Above the clouds: A berkeley view of cloud computing," University of California, Berkeley, Tech. Rep., Feb. 2009.

[2] S. Das, D. Agrawal, and A. El Abbadi, "Elastras: an elastic transactional data store in the cloud," in *USENIX HotCloud*, 2009.

[3] D. J. Abadi, "Data management in the cloud: Limitations and opportunities," IEEE Data Engineering Bulletin, Mar. 2009.

[4] A. J. Lee and M. Winslett, "Safety and consistency in policy-based authorization systems," in *ACM CCS*, 2006.

[5] M. Myers, R. Ankney, A. Malpani, S. Galperin, and C. Adams, "X.509 internet public key infrastructure online certificate status protocol - ocsp," RFC 2560, Jun. 1999, http://tools.ietf.org/html/rfc5280.

[6] E. Rissanen, "extensible access control markup language (xacml) version 3.0," Jan. 2013, http://docs.oasis-open.org/xacml/3.0/xacml-3.0- core-spec-os-en.html.

[7] D. Cooper *et al.*, "Internet x.509 public key infrastructure certificate and certificate revocation list (crl) profile," RFC 5280, May 2008, http://tools.ietf.org/html/rfc5280.

[8] J. Li, N. Li, and W. H. Winsborough, "Automated trust negotiation using cryptographic credentials," in *ACM CCS*, Nov. 2005.

[9] L. Bauer *et al.*, "Distributed proving in access-control systems," in *Proc. of the IEEE Symposium on Security and Privacy*, May 2005.

[10] J. Li and N. Li, "OACerts: Oblivious attribute based certificates," *IEEE TDSC*, Oct. 2006.

[11] J. Camenisch and A. Lysyanskaya, "An efficient system for nontransferable anonymous credentials with optional anonymity revocation," in *EUROCRYPT*, 2001.

[12] P. K. Chrysanthis, G. Samaras, and Y. J. Al-Houmaily, "Recovery and performance of atomic commit processing in distributed database systems," in *Recovery Mechanisms in Database Systems*. PHPTR, 1998.

[13] M. K. Iskander, D. W. Wilkinson, A. J. Lee, and P. K.Chrysanthis, "Enforcing policy and data consistency of cloud transactions," in *IEEE ICDCS-SPCC*, 2011.