www.ijrream.in Impact Factor: 2.9463

INTERNATIONAL JOURNAL OF RESEARCH REVIEW IN ENGINEERING AND MANAGEMENT (IJRREM)

Examining and Adaptable Secure Cloud Exchanges for Priority Assignment

[1]R. Monisha, PG Student, M.E Computer Science and Engineering

[2]Dr. S. Chandrasekaran, Professor / CSE

P.S.V. College of Engineering & Technology

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 authorization. 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 Validation Commit protocols. We finally analyze the different approaches presented 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

Cloud computing has recently emerged as a computing paradigm in which storage and computation can be outsourced from organizations to next generation data centers hosted by companies such as Amazon, Google, Yahoo, and Microsoft. Such companies help free organizations from requiring expensive infrastructure and expertise in-house, and instead make use of the cloud providers to maintain, support, and broker access to high-end

resources. From an economic perspective, cloud consumers can save huge IT capital investments and be charged on the basis of a pay only for what you use pricing model.

One of the most appealing aspects of cloud computing is its elasticity, which provides an illusion of infinite, on- demand resources making it an attractive environment for highly scalable, multitiered applications. However, this can create additional challenges for back-end, transactional database systems, which were designed without elasticity in mind. Despite the efforts of key-value stores like Amazon's SimpleDB, Dynamo, and Google's Bigtable to provide scalable access to huge amounts of data, transactional guarantees remain a bottleneck.

To provide scalability and elasticity, cloud services often make heavy use of replication to ensure consistent performance and availability. As a result, many cloud services rely on the notion of eventual consistency when propagating data throughout the system. This consistency model is a variant of weak consistency that allows data to be inconsistent among some replicas during the update process, but ensures that updates will eventually be propagated to all replicas. This makes it difficult to strictly maintain the ACID guarantees, as the "C" (consistency) part of ACID is sacrificed to provide reasonable availability. In systems that host sensitive resources, accesses are protected via authorization policies that describe the conditions under which users should be permitted access to resources. These policies describe relationships between the system principles, as well as the certified credentials that users must provide to attest to their attributes. In a transactional database system that is deployed in a highly distributed and elastic system such as the cloud, policies would typically be replicated very much like data among multiple sites, often following the same weak or eventual consistency model. It therefore becomes possible for a policy-based authorization system to make unsafe decisions using stale policies.

Interesting consistency problems can arise as transactional database systems are deployed in cloud environments and use policy based authorization systems to protect sensitive resources. In addition to handling consistency issues among database replicas, we must also handle two types of security inconsistency conditions. First, the system may suffer from policy inconsistencies during policy updates due to the relaxed consistency model underlying most cloud services. For example, it is possible for several versions of the policy to be observed at

multiple sites 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 over the lifetime of a transaction. For instance, a user's login credentials could be invalidated or revoked after collection by the authorization server, but before the completion of the transaction. We make the following contributions:

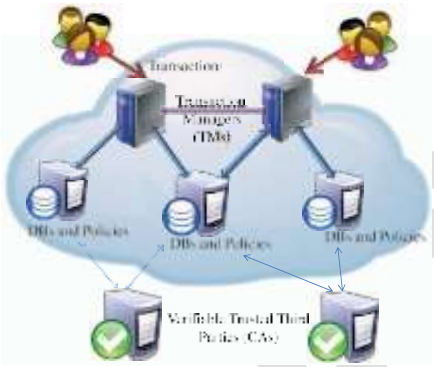

Fig. 1. Interaction among the system components.

- We formalize the concept of trusted transactions. Trusted transactions are those transactions that do not violate credential or policy inconsistencies over the lifetime of the transaction. We then present a more general term, safe transactions, that identifies transactions that are both trusted and conform to the ACID properties of distributed database systems.
- We define several different levels of policy consistency constraints and corresponding enforcement approaches that guarantee the trustworthiness of transactions executing on cloud servers.
- We propose a Two-Phase Validation Commit (2PVC) protocol that ensures that a transaction is safe by checking policy, credential, and data consistency during transaction execution.
- We carry out an experimental evaluation of our proposed approaches, and present a tradeoff discussion to guide decision makers as to which approach is most suitable in various situations.

2. SYSTEM MODEL

Fig. 1 illustrates the interaction among the components in our system. 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 c 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.

To enhance the general applicability of the consistency models developed in this paper, the above formalism is intentionally opaque with respect to the policy and credential formats used to implement the system. For instance, this formalism could easily be used to model the use of XACML policies as the set of inference rules R, and traditional credentials for the set C. On the other hand, it can also model the use of more advanced trust management policies for the inference rules R, and the use of privacy-friendly credentials for the set C.

3. 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 transaction's 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 authorization evaluated during the lifetime of a particular transaction.

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.

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.

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.

4. IMPLEMENTING SAFE TRANSACTIONS

A safe transaction is a transaction that is both trusted (i.e., satisfies the correctness properties of proofs of authorization) and database correct (i.e., satisfies the data integrity constraints). We first describe an algorithm that enforces trusted transactions, and then expand this algorithm to enforce safe transactions. Finally, we show how these algorithms can be used to implement the approaches.

Two-Phase Validation (2PV) Algorithm

A common characteristic of most of our proposed approaches to achieve trusted transactions is the need for policy consistency validation. The transaction 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.

As the name implies, 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.

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 cause 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) reevaluate the proofs and, 3) send a new reply to the TM. Algorithm 1 shows the process for the TM.

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

Two - Phase Validate Commit Algorithm

The 2PV protocol enforces trusted transactions, but does not enforce safe transactions because it does not validate any integrity constraints. Since the Two-Phase Commit atomic protocol commonly used to enforce integrity constraints has similar structure as 2PV, we propose integrating these protocols into a Two-Phase Validation Commit protocol.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).

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

5. ENVIRONMENT AND SETUP

We used Java to implement each proof approach described in Section 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 chosen according to Table 1. To understand the performance implications of the different approaches, we varied the

1. Protocol used,
2. Level of consistency desired,
3. Frequency of master policy updates,
4. Transaction length, and
5. 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.

6. 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 centres. In this paper, 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 lightweight 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.

References:

[1] E. Rissanen, "Extensible Access Control Markup Language (Xacml) Version 3.0," http://docs.oasis-open.org/xacml/3.0/ xacml-3.0-core-spec-os-en.html, Jan. 2013.

[2] D. Cooper et al., "Internet x.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile," RFC 5280, http://tools.ietf.org/html/rfc5280, May 2008.

[3] J. Li, N. Li, and W.H. Winsborough, "Automated Trust Negotia- tion Using Cryptographic Credentials," Proc. 12th ACM Conf. Computer and Comm. Security (CCS '05), Nov. 2005.

[4] L. Bauer et al., "Distributed Proving in Access-Control Systems," Proc. IEEE Symp. Security and Privacy, May 2005.

[5] J. Li and N. Li, "OACerts: Oblivious Attribute Based Certificates," IEEE Trans. Dependable and Secure Computing, vol. 3, no. 4, pp. 340- 352, Oct.-Dec. 2006.

[6] J. Camenisch and A. Lysyanskaya, "An Efficient System for Non- Transferable Anonymous Credentials with Optional Anonymity Revocation," Proc. Int'l Conf. Theory and Application of Crypto- graphic Techniques: Advances in Cryptology (EUROCRYPT '01), 2001.