Enforcing Policy and Data Consistency of Cloud Transactions: A Simulation

Tucker Trainor

Department of Computer Science, University of Pittsburgh

`tmt33@pitt.edu`

April 30, 2012

Abstract

With an increase in services offering data storage in the cloud comes an increase in scrutiny over the security of that data. Narrowing our focus on distributed transactional database systems deployed over cloud servers, we can look at how distribution of user authorization policies affects the trustworthiness of transactions. If policy versions are in an inconsistent state across the cloud servers utilized during that transaction, how can we be sure that the transaction is trusted? A proposed solution lies in the use of Two-Phase Validation protocols [1], a modified version of basic Two-Phase Commit protocols. We can simulate the protocol's deferred proofs of authorization to quantify the cost of implementing the protocol in terms of time cost and commit success rate.

1. Model

In this section we introduce 2-phase validation (2PV) [1] and how validation factors into the different proofs of authorization. We describe in general terms the roles of clients, cloud servers, and policy servers. We then briefly illustrate the transactions between all parties and how the proofs alter their interactions and possible outcomes.

A. Protocols

To model the simulation, we begin by examining Two-Phase Validation (2PV) and deferred proofs of authorization and determine what we will require to accurately produce a scenario in which their performance can be measured. Deferred proofs of authorization define a situation where policy validation occurs at commit time, similar in spirit to deferred integrity checks on a database. Over the lifetime of a transaction, multiple servers are likely to be utilized to complete the transaction. Once a server is called upon to assist in

completion of the transaction, it stores its most recent local (to the server) policy version for that transaction. The transaction's stored policy version is untouched until a commit is requested for the transaction. At this point there are two levels of policy consistency that can be enforced to determine if the transaction is to be considered trustworthy: view consistency and global consistency.

For view consistency to be true, each server which took part in the transaction must have the same stored policy, i.e., the policy versions should be internally consistent across all servers involved in the transaction [1]. In 2PV, we use a collection phase to gather the policy versions from the servers, and a validation phase to determine whether or not the servers are in agreement. In our model, if the policy versions are not consistent amongst all servers, then the transaction is aborted. If the view is consistent, then the transaction is allowed to commit. View consistency is quick in terms of checking validity but suffers from an inherent weakness in terms of policy freshness. Once the policy used for validation is set at the beginning of each server's role in the transaction, any updates to policy versions that occur after that point are not taken into account. As is common in the realm of computer science, speed is gained at the cost of security.

Global consistency involves a more stringent approach to validation to eliminate the weakness noted above for view consistency. At the validation phase, the policy versions collected from each server are compared to the latest policy version globally available from the originator of policies. Using the latest policy for validation purposes is an improvement of the trust level that view consistency provides. Additionally, instead of aborting the transaction in the case of any server's policy not matching the latest policy version, we use the latest version to execute a new local authorization check on each operation performed during the transaction that used an earlier policy for the local authorization check during the initial run of the transaction. As each server records all operations, it can execute local authorizations from its records without repeating the entire transaction. If all operations in the transaction are locally authorized by the latest policy version, then the transaction is allowed to commit. Otherwise, the transaction is aborted. Due to the possibility of running a series of local authorizations, global consistency is likely to be more costly in terms of time when compared to view consistency. However, by offering an alternative to an abort after server policy inconsistency, successful commits may improve under certain conditions.

B. Transaction Flow

clients, servers, transaction flow

2. Experimental Testbed

In this section we describe the implementation of the model and algorithms that apply to it. The test environment is detailed, with details such as hardware and software (i.e., Java)

touched upon. The different client/server modules and their child threads are described and their variables are listed. The effects of the variables on their respective modules are summarized in terms of relevance to the simulation.

Parameters

3. Experiments

In this section we produce representations of the resulting data from the simulation. The parameters used for each set of simulations is listed and followed by the results. Each set of results is prefaced by the real-world environment that we attempted to simulate, along with explanations of specific decisions in parameter setting.

Performance evaluation

describe graphs
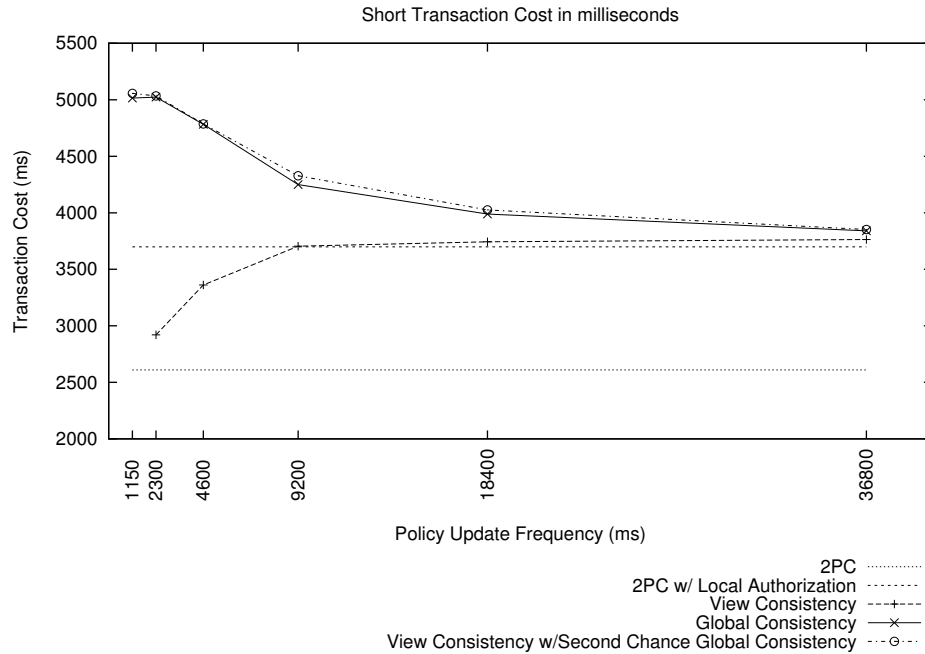
Figure 1: Short Transaction Cost
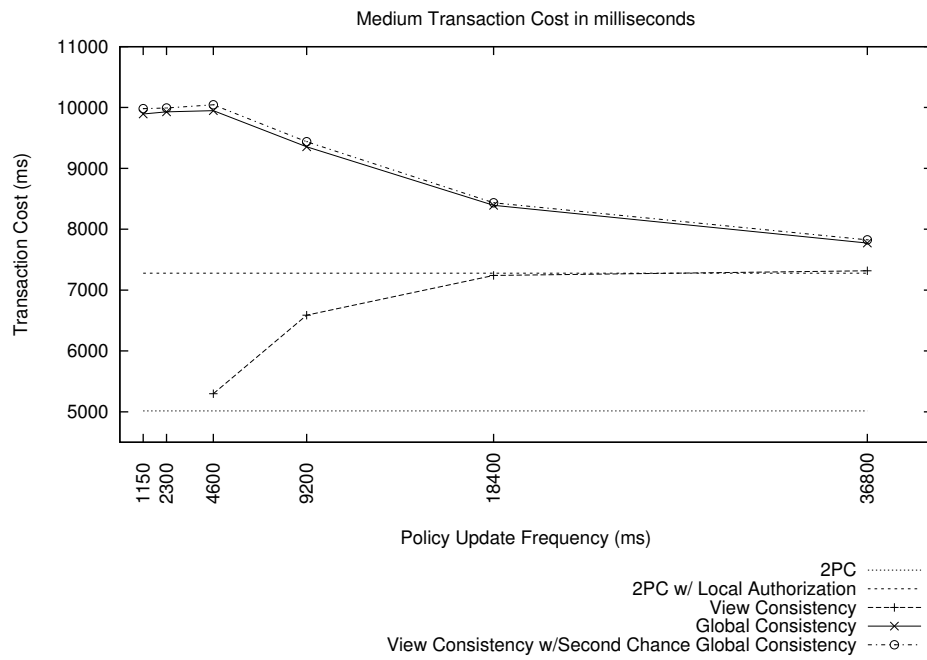
Figure 2: Medium Transaction Cost
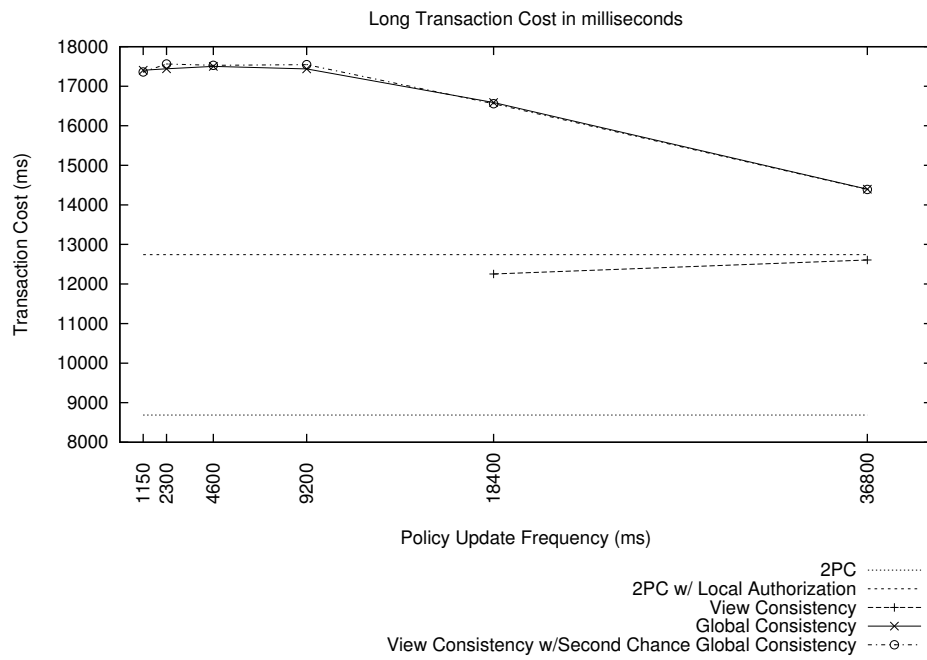
Long Transaction Cost in milliseconds
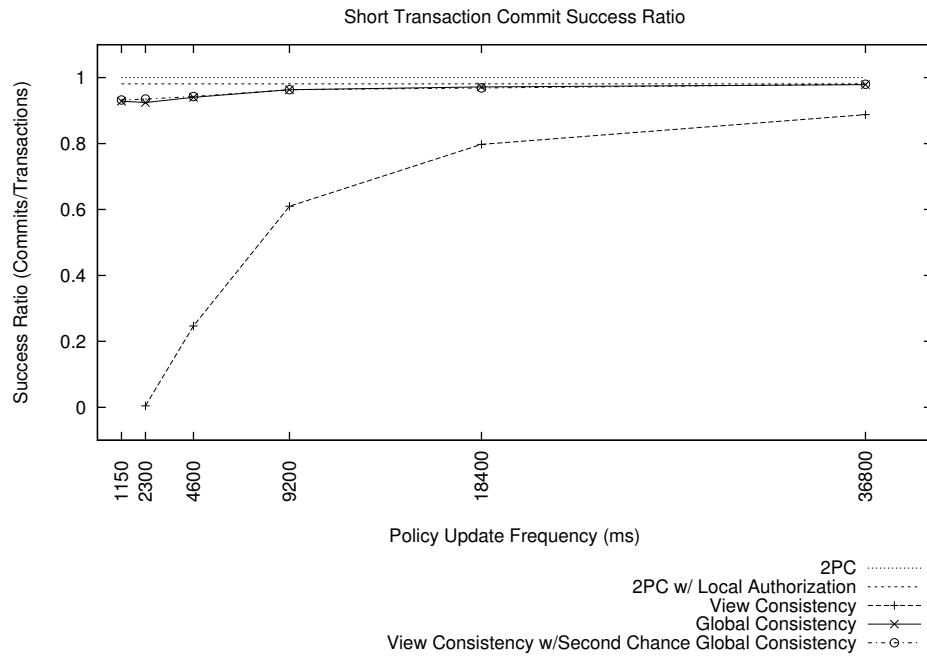

Figure 3: Long Transaction Cost

5

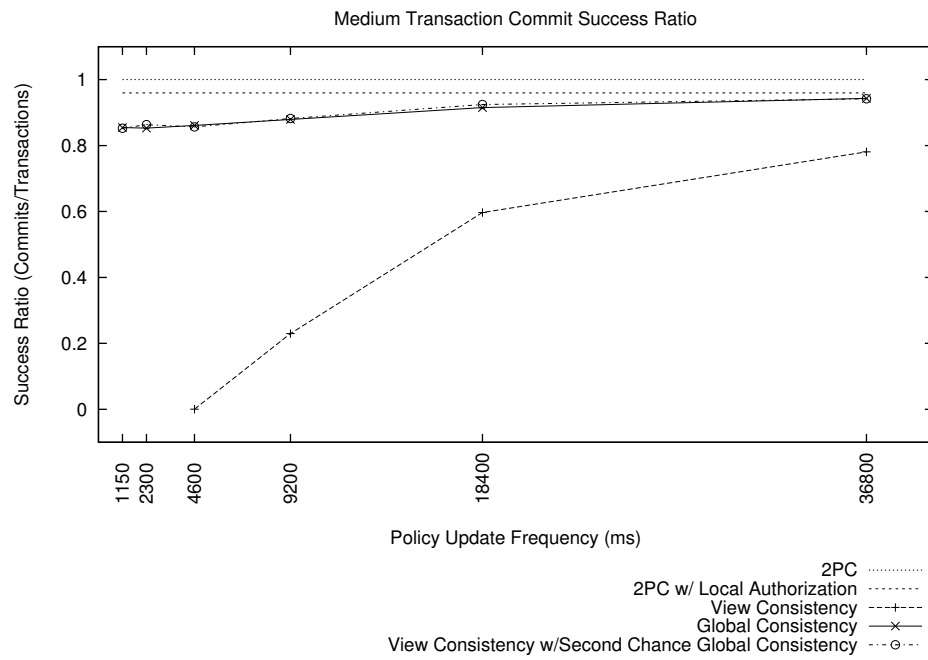Figure 4: Short Transaction Commit Success Ratio
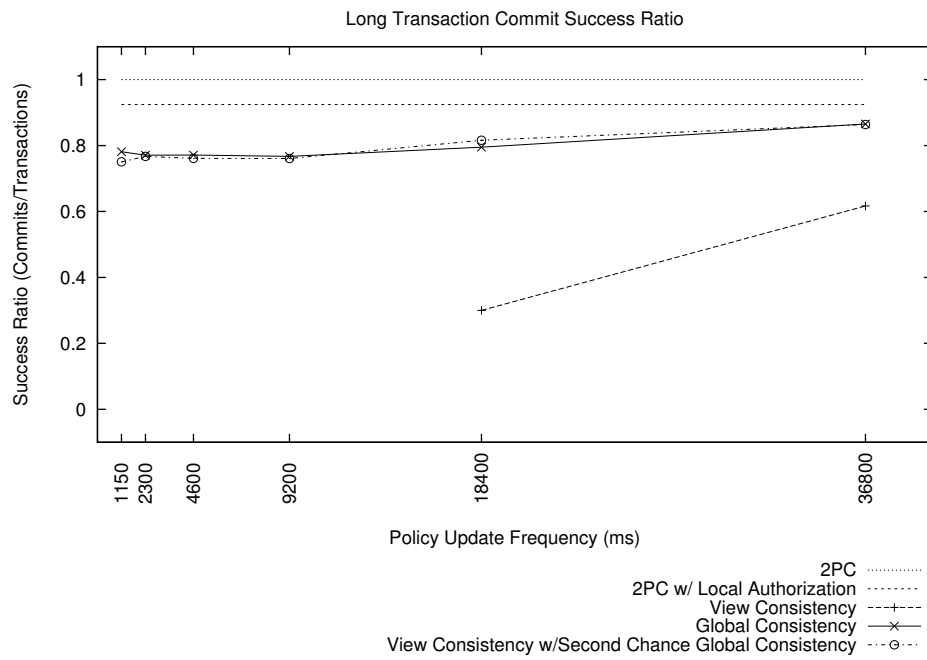
Figure 5: Medium Transaction Commit Success Ratio
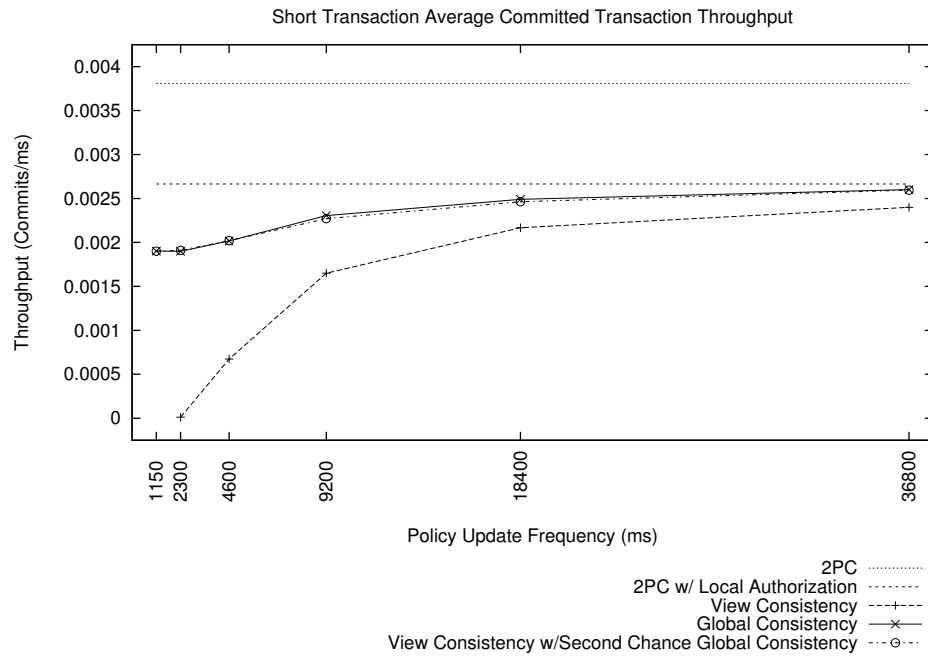
Figure 6: Long Transaction Commit Success Ratio

Figure 7: Short Transaction Throughput

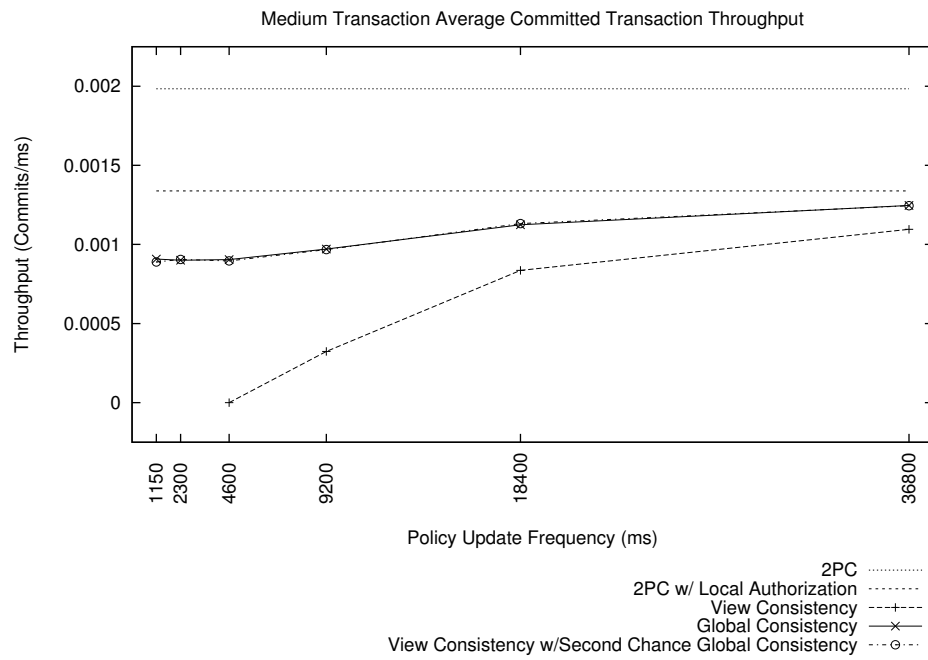Medium Transaction Average Committed Transaction Throughput
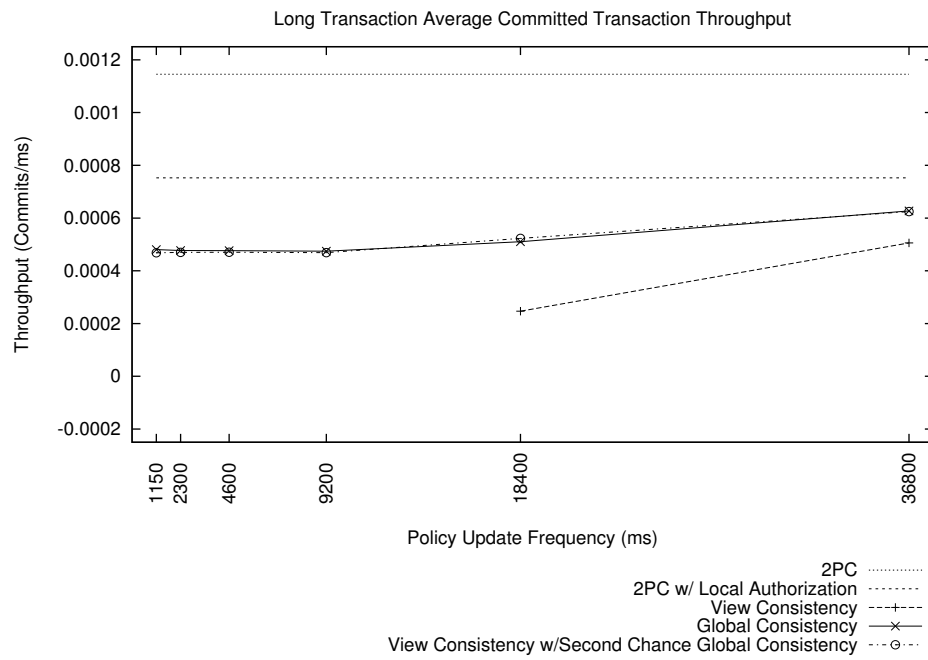

Figure 8: Medium Transaction Throughput

Figure 9: Long Transaction Throughput

Sensitivity analysis

give analysis

4. Conclusions & Observations

In this section we summarize the findings from the simulation and explore the potential of the algorithms simulated. We will note any findings that suggest further simulation or experimentation. We will also recount any notable observations that occurred during implementation or simulation.

notes

Mention reduction of group size, object was to implement all proofs, instead doing half, leaving remainder for future research

References

[1] Iskander *et al.*, "Enforcing Policy and Data Consistency of Cloud Transactions," Department of Computer Science, University of Pittsburgh.