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 two-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 again at commit time, 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.

We can also create a protocol that combines the strengths of both view consistency and global consistency. Called view consistency with second chance global consistency, the protocol allows a transaction to check for view consistency at first, and if that fails then it can invoke a global consistency check to possibly save the transaction from aborting. Thus, the transaction has the ability to quickly be validated for commit via view consistency, but is possibly saved from an initial abort by a global consistency check. We suspect that time cost may increase when transactions are subject to both consistency checks, but commit success rates will improve over view consistency checks only.

B. Transaction Flow

clients, servers, transaction flow

To benchmark these protocols, we will record the interaction between experimental clients and servers as they process sample transactions. A robot-controlled client will send transactions to one of several cloud data servers. This primary server will serve as a transaction manager for the entire transaction, routing operations requiring other cloud data servers to those servers as necessary. The transaction manager is also responsible for collection and validation phases of any 2PC and 2PV

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.

A. Performance evaluation

Transaction Cost

We represent transaction cost as the time in milliseconds that a transaction requires for completion. We performed simulations of all protocols for short transactions (see fig. 1), medium transactions (see fig. 2), and long transactions (see fig. 3). We began with a policy update frequency of 1,150 ms and doubled the frequency for each subsequent set of runs up to 36,800 ms. For each simulation we averaged the duration of each successfully committed transaction to provide a value of cost for view consistency, global consistency, and view consistency with second chance global consistency.

For short length transactions we observed a 2PC baseline cost of 2,610 ms, rounded to the nearest integer. Using 2PC with a 99.5% local authorization success rate as a baseline, we observed a cost of 3,698 ms, rounded to the nearest integer. Figure 1 shows the results of the 2PV protocols against the baseline values.

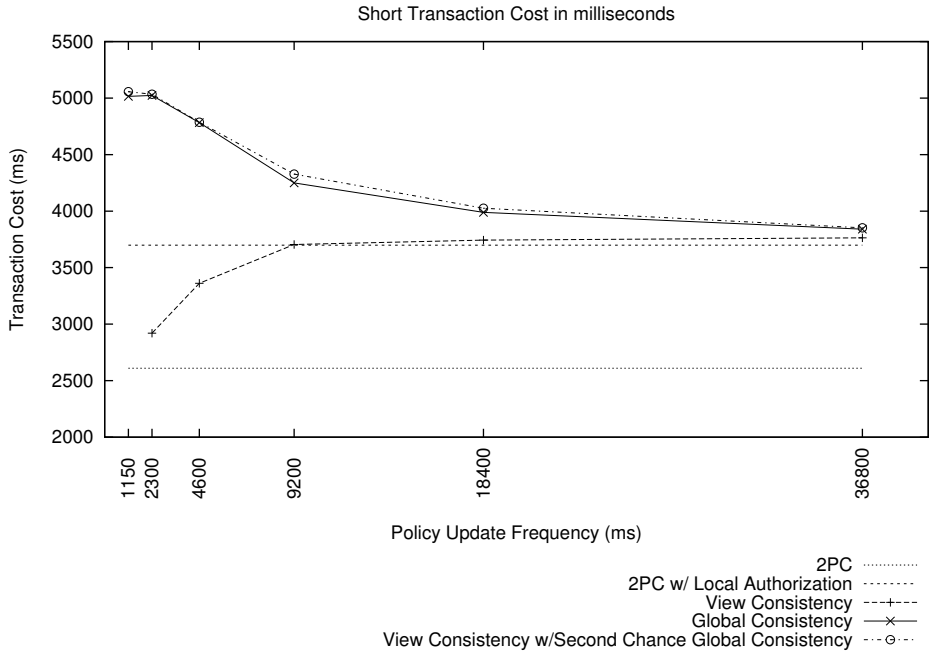

Figure 1: Short Transaction Cost

For medium length transactions we observed a 2PC baseline cost of 5,015 ms, rounded to the nearest integer. Using 2PC with a 99.5% local authorization success rate as a baseline, we observed a cost of 7,279 ms, rounded to the nearest integer. Figure 2 shows the results of the 2PV protocols against the baseline values.

For long length transactions we observed a 2PC baseline cost of 8,688 ms, rounded to the nearest integer. Using 2PC with a 99.5% local authorization success rate as a baseline, we observed a cost of 12,742 ms, rounded to the nearest integer. Figure 3 shows the results of the 2PV protocols against the baseline values.

Successful Commit Ratio

We represent the commit success ratio as the number of commits divided by the total number of transactions attempted. We performed simulations of all protocols for short transactions (see fig. 4), medium transactions (see fig. 5), and long transactions operations each (see fig. 6). We began with a policy update frequency of 1,150 ms and doubled the amount for each subsequent set of runs up to 36,800 ms.

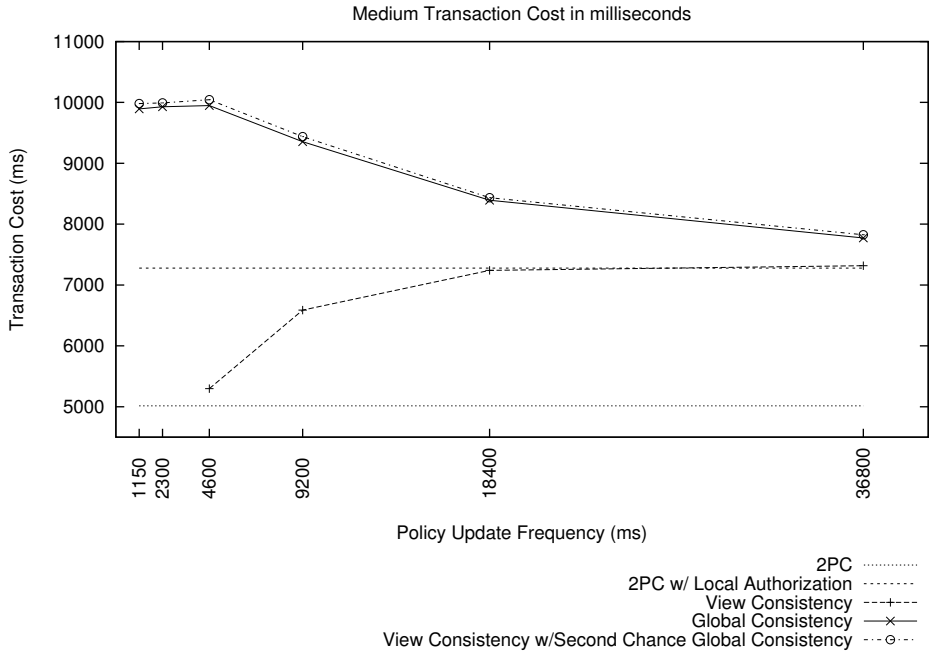

Figure 2: Medium Transaction Cost

Transaction Throughput

We represent transaction throughput as the number of commits in a simulation run divided by the time in milliseconds of the duration of that simulation run. We performed simulations of all protocols for short transactions (see fig. 7), medium transactions (see fig. 8), and long transactions (see fig. 9). We began with a policy update frequency of 1,150 ms and doubled the amount for each subsequent set of runs up to 36,800 ms.

B. 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.

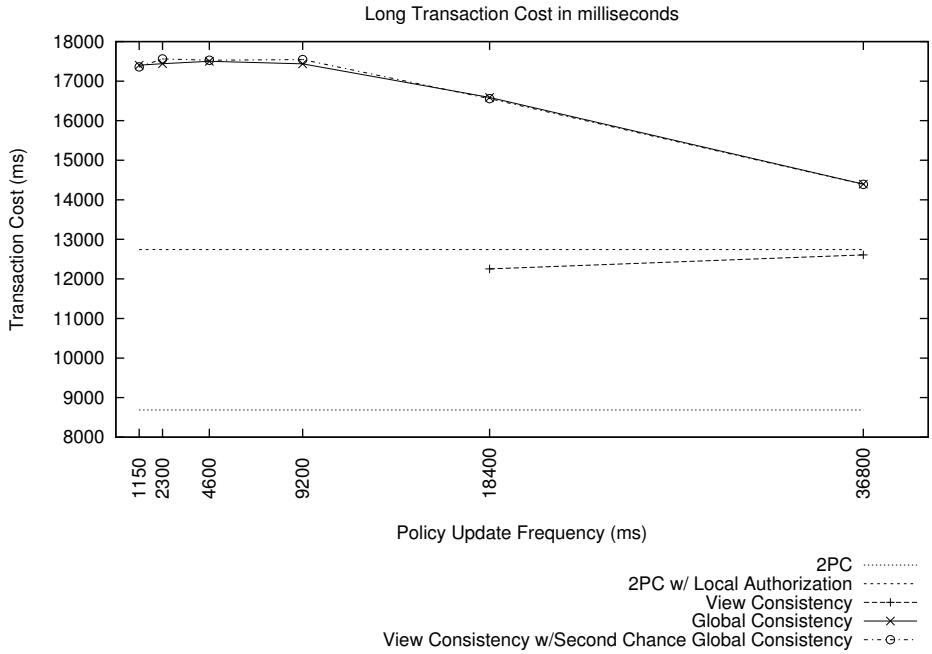

Figure 3: Long Transaction Cost

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.

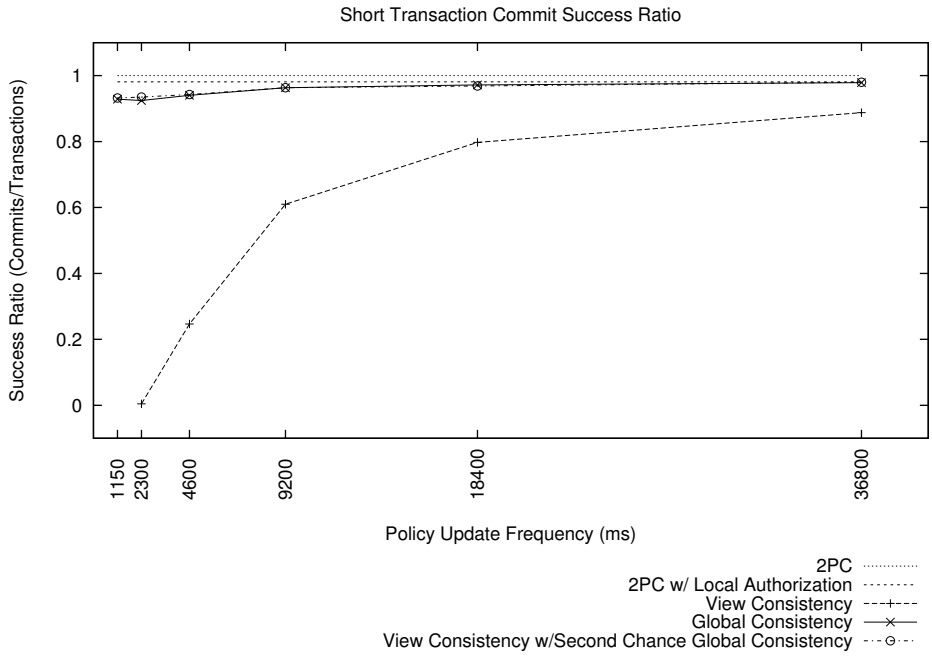

Figure 4: Short Transaction Commit Success Ratio

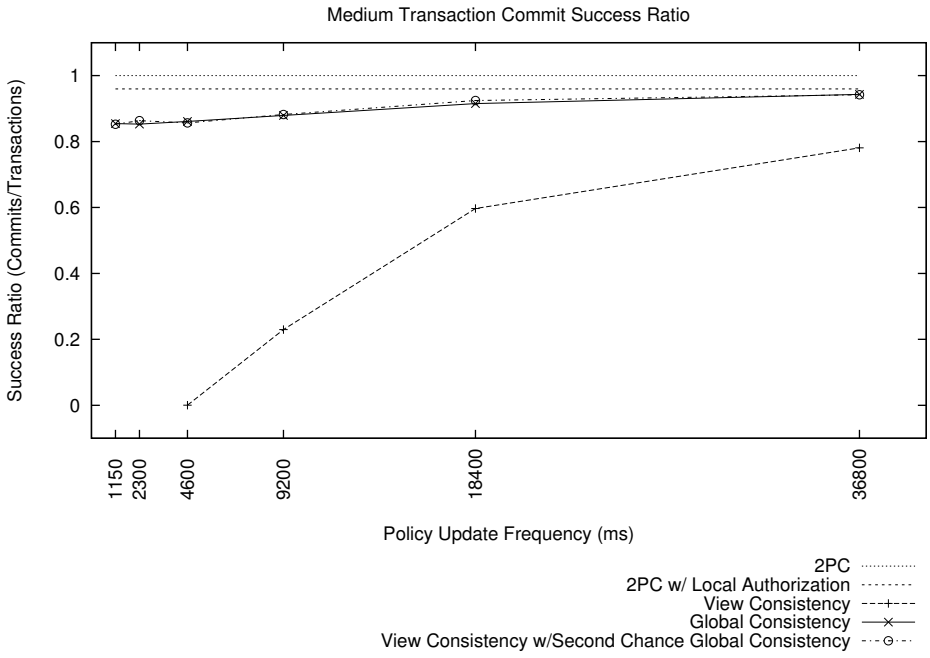

Figure 5: Medium Transaction Commit Success Ratio

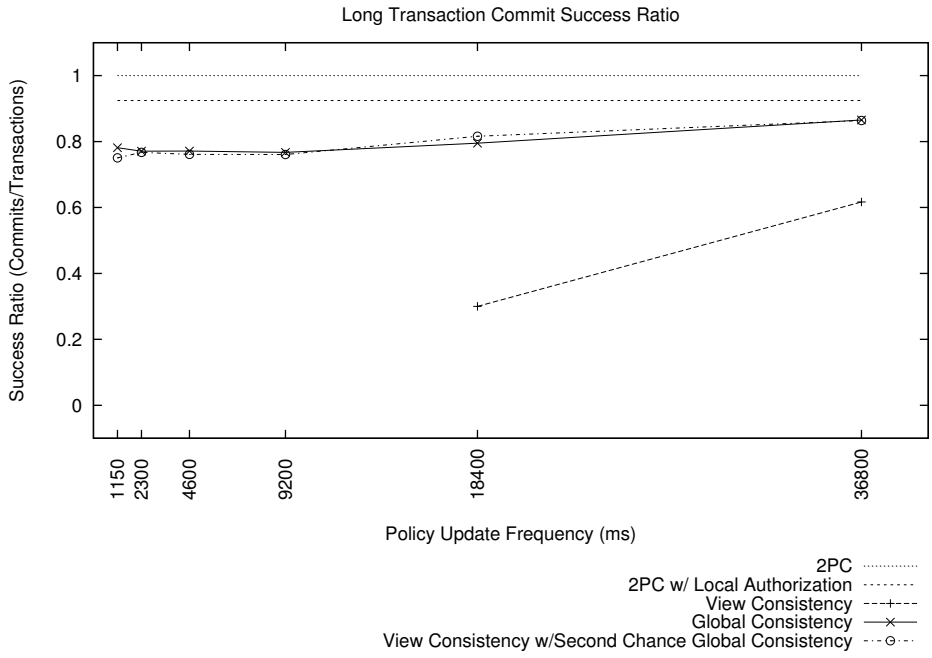

Figure 6: Long Transaction Commit Success Ratio

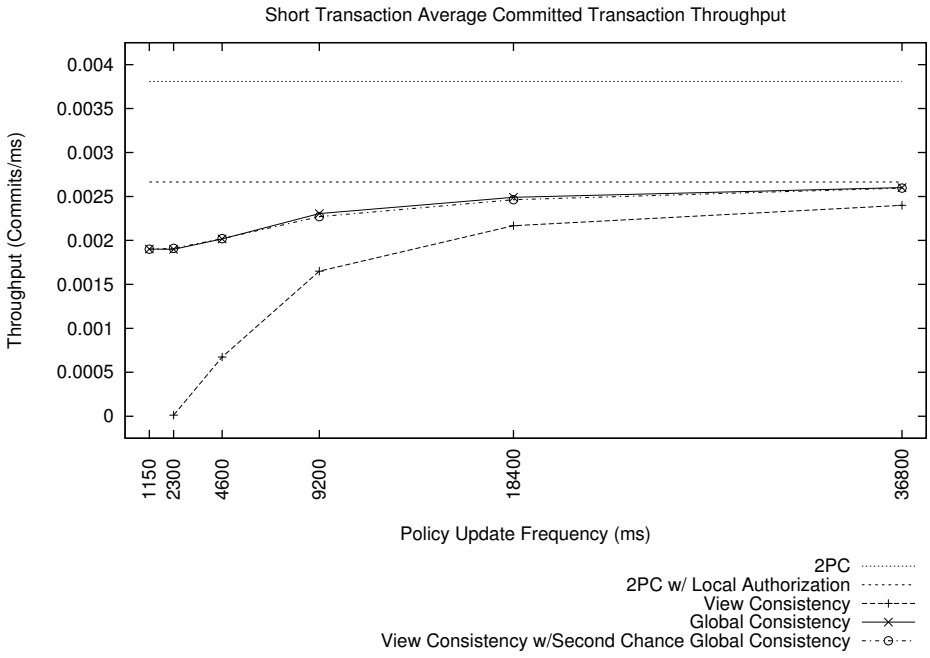

Figure 7: Short Transaction Throughput

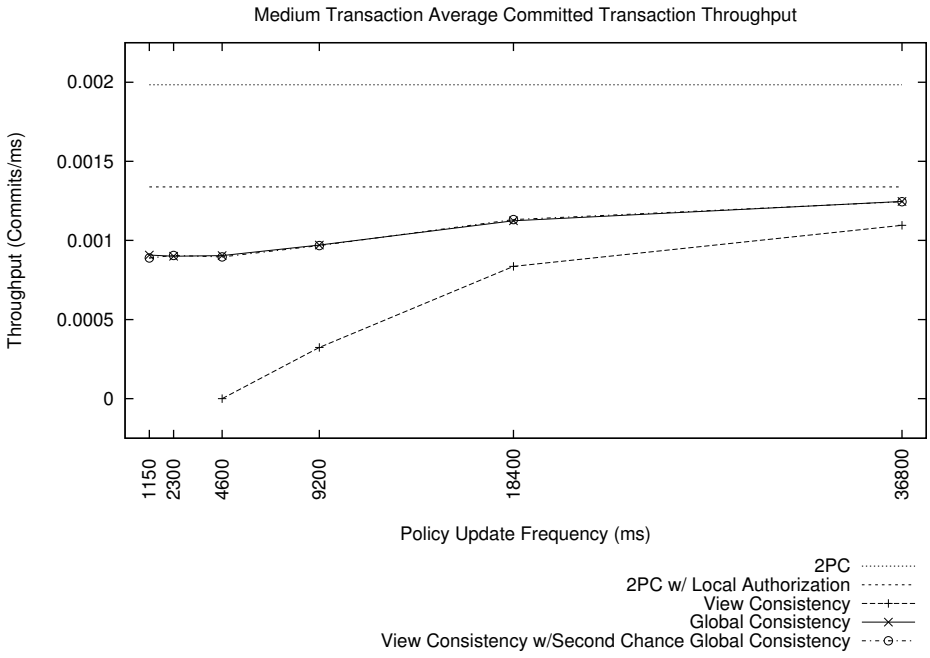

Figure 8: Medium Transaction Throughput

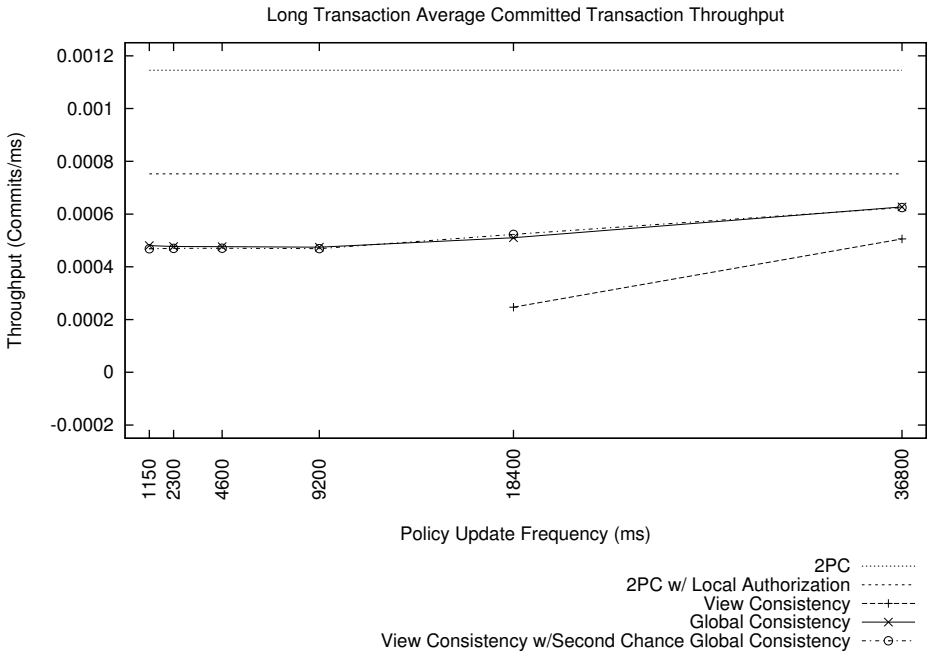

Figure 9: Long Transaction Throughput