An Investigation of the Visibility and Correctness of Read-Only Transactions in Fault-Tolerant Systems

Al-kesna Foster

December 2025

1 Introduction

Read-only transactions are highly prevalent in real systems of today. They dominate the workload of systems used by companies such as Meta and Google. The dominance of read-only transactions means that they are primary determinants of user latency and system throughput, making them critical to the performance of a system.

2 Invisible Reads

Invisible reads are read-only transactions that leave no visible components - they do not update any metadata that affects other transactions (e.g., read timestamp, read locks, transaction IDs). If a protocol's reads update any concurrency-control state, that state becomes replication-critical and must survive failures; otherwise,

3 Research Question

Which existing concurrency control protocols claim to support invisible read-only transactions, and how do their correctness and performance hold up in replicated, fault-tolerant settings?

4 Analysis

System	Consistency Level	Visible Components	Invisible Read Support	Correctness in Fault-Tolerant Environment
2-Phase Locking	Serializability	Yes - Locks	No - A ROX may acquire a lock, blocking a conflicting TX	In Distributed 2PL, the acquisition and release of locks is coordinated between shards/replicas, ensuring consistent decisions
Timestamp-Ordering	Serializability	Yes - Read and Write Timestamps	No - A ROX can update the read timestamp of an object, blocking a conflicting TX	In Distributed Timestamp Ordering, read and write timestamps of objects are coordinated between shards/replicas, ensuring consistent decisions
Cops-SNOW	Causal Consistency	Yes - Versions, Causal Dependencies	Yes - ROX do not update any metadata that directly block concurrent TXs	In a replicated environment, all versions and causal dependencies are replicated before committing, ensuring consistent decisions
Rococo-SNOW	Strict Serializability	No	Yes - ROXs do not update any metadata; only writes do	Rococo's coordinator ensures consensus on TXNs amongst shards before committing, ensuring consistent decisions
Scylla-PORT	Process-Ordered Serializability	Yes - Version Clocks, Versions	Yes - ROX update metadata but other TXNs are not affected by them	In a replicated environment, all versions and version clocks are replicated before committing, ensuring consistent state
Eiger-PORT	Causal Consistency	Yes - Versions, Lamport Timestamps for writes	Yes - Eiger-PORT reorders the ROXs and conflicting writes without blocking	In a replicated environment, all versions and timestamps are replicated before committing, ensuring consistent state
Spanner	Strict Serializability	Yes - Locks for RW TXNs, truetime interval timestamp	Yes - Spanners ROX don't affect the outcome of concurrent write TXNs	Spanner uses Paxos for consensus, ensuring consistent decisions