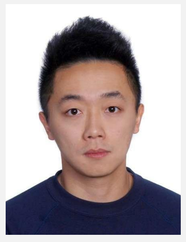
ETH zürich

Department of Computer Science
Information Security Group

VerIso: Verifiable Isolation Guarantees for Database Transactions


Shabnam Ghasemirad


Si Liu


Christoph Sprenger


Luca Multazzu

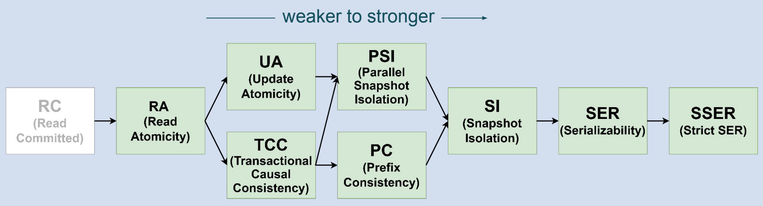

David Basin

shabnam.ghasemirad@inf.ethz.ch

Project repository:

1 Motivation

- Distributed databases (key-value stores)
- **Isolation** of concurrent transactions, realized by concurrency control protocols.
- Spectrum of isolation levels:



- Concurrency control protocols are highly complex and prone to design errors and isolation bugs.
- **What is missing?** A mathematically **rigorous** framework with systematic **tool support** for verifying isolation guarantees.

Contributions:

VerIso

➢ The first **mechanized** framework for **formal** (deductive) **verification** of **isolation guarantees** in **concurrency control protocols**

➢ Supports a wide range of isolation levels

➢ Protocol design **verification** & **falsification**
- Found an atomic visibility bug in **TAPIR**!

2 Modelling Isolation Levels

We have modeled isolation levels as one abstract *parameterized* model in VerIso.

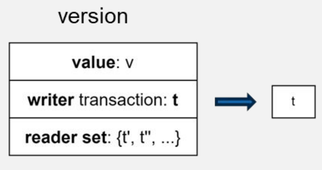
The abstract model's event system:

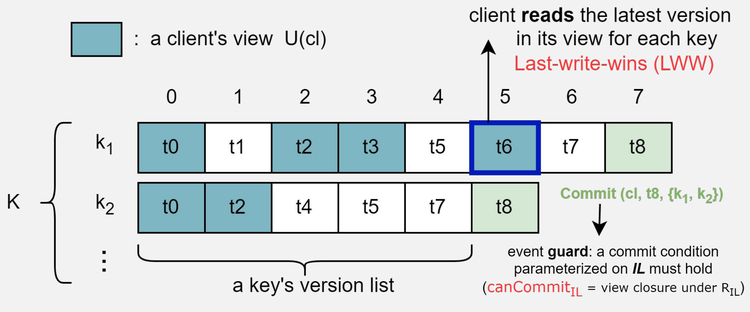
➔ States: (K, U)
- **K**: centralized multi-versioned key-value store
- **U**: client views, capturing the distributed aspect

➔ Events:

1) Atomic Commit
(K, U) $\xrightarrow{\text{commit}}_{IL}$ (K', U')

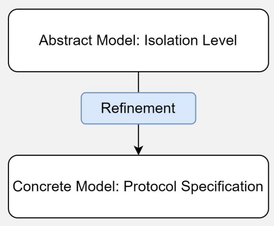
2) View Extension
(K, U) $\xrightarrow{\text{view ext}}_{IL}$ (K, U')

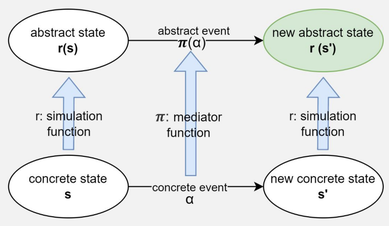




3 Protocol Verification Method

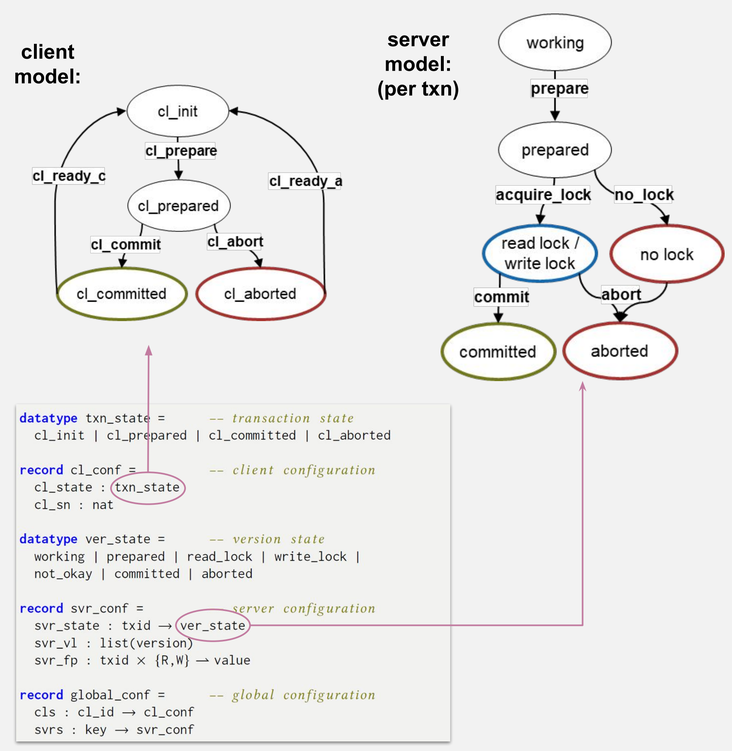
In VerIso, we verify isolation guarantees of a concurrency control protocol by **Refinement**.





The abstract and protocol model specification are based on labeled transition systems (LTS).

Example: S2PL + 2PC protocol




```
datatype txn_state =      -- transaction state
  cl_init | cl_prepared | cl_committed | cl_aborted

record cl_conf =          -- client configuration
  cl_state : txn_state
  cl_sn : nat

datatype ver_state =      -- version state
  working | prepared | read_lock | write_lock |
  not_okay | committed | aborted

record svr_conf =         -- server configuration
  svr_state : txid → ver_state
  svr_vl : list(version)
  svr_fp : txid × {R,W} → value

record global_conf =      -- global configuration
  cls : cl_id → cl_conf
  svrs : key → svr_conf
```

Proof obligations

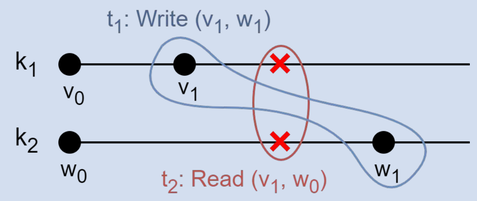
- View atomicity baseline (RA)
- Last-write-wins (LWW)
- canCommit (IL specific): for **SSER**, global view for all clients → achieved by *Locking*

5 Atomic Visibility Bug in TAPIR

TAPIR (Transactional Application Protocol for Inconsistent Replication [Zhang et al., '15])

- Claims **SSER**
- Timestamp-based protocol
- 2PC + optimistic concurrency control (OCC)
- OCC decision based on committed and pending reads and writes' timestamps

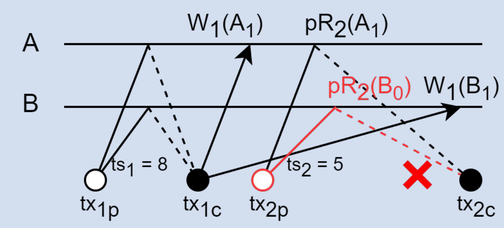
Fractured Read: Read Atomicity (RA) violation



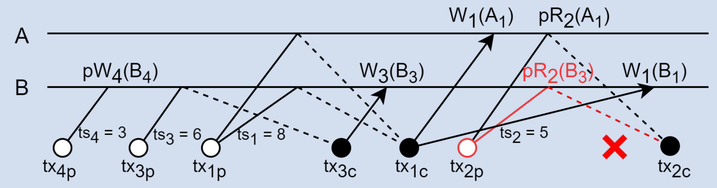
❖ **VerIso** reveals an RA violation in TAPIR:

➢ Client can read a transaction's update on one server but an older version on another.

counterexample for the **journal** version:



counterexample for the **conference** version:

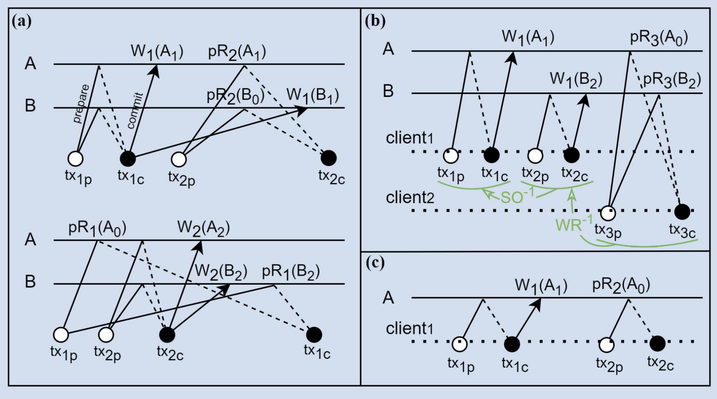


❖ Counterexample constructed from **failed Last-Write-Wins (LWW)** proof by assigning timestamps that would pass the OCC check

❖ The bug had escaped manual proofs and model checking (TLA+)

4 Protocol Falsification Method

- Set up a refinement proof in VerIso for the **claimed IL**
- Encounter an **unprovable** proof obligation
- Write down the scenario (**execution**) making the proof impossible
- Use the protocol model to validate the execution: **counterexample**!



Picture: Interleaving of transactions' events, violating (a) LWW, (b) canCommit_{TCC}, and (c) vShift_{RYW}.