Modular verification of MongoDB Transactions using TLA+

Will Schultz and Murat Demirbas

MongoDB Research
http://mongodb.com

May 4, 2025

Transactions

A transaction groups multiple operations into an all-or-nothing logical box

- Consider each box in isolation to simplify concurrency & fault handling

ACID: Atomicity, Consistency, Isolation, Durability

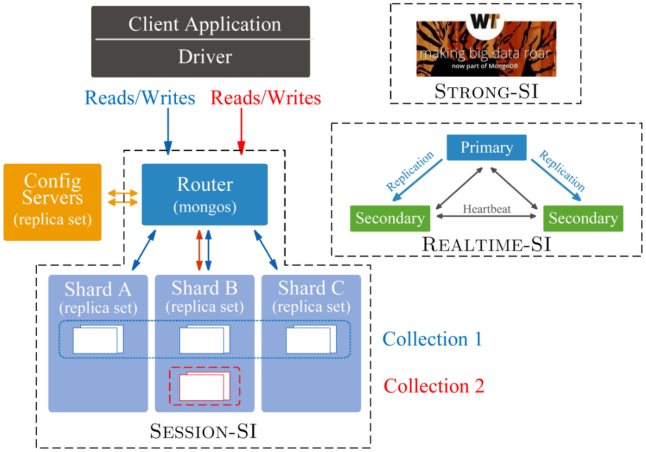
- Read Committed (RC), Snapshot Isolation (SI), Serializability (SER) offer increasing protection against concurrency anomalies

MongoDB transactions history

V3.2 (Dec'15): single-document txns in one node via MVCC WiredTiger (WT) storage

V4.0 (Jun'18): multi-document txns in a replicaset/shard

V4.2 (Aug'19): distributed multi-doc txns across shards



Layered implementation of transactions

Replicaset transactions workflow: All txn operations are first performed on the primary using WiredTiger transaction workflow

Before commit of txn, all txn updates are Raft-replicated with secondaries using the assigned timestamp

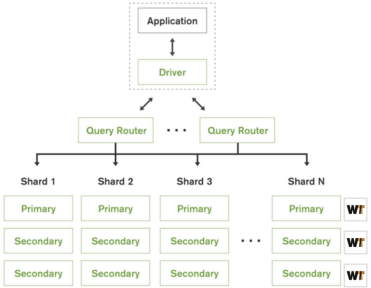
- ReadConcern:Snapshot ensures that data has been committed to the majority of the replica set, and read is done on a consistent time cut
- WriteConcern:Majority ensures that when a write is acked, it's been replicated (flushed to disk) to a majority of the replicaset nodes

Distributed transactions

MongoDB txns are general interactive txns

Client connects to *mongos*, txn-router, which sets the read-ts of the txn using its cluster time, and routes ops to shards

Shard primary sets WT read-ts on first op as supplied, delegates op handling to ReplicaSet alg, and aborts on conflict informing *mongos*



Mongos asks first shard to coordinate 2-phase commit (2PC)

Coordinator sends prepare to all participant shards

Each participant computes a prepare timestamp & Raft-replicates prepare

Coordinator picks the max prepare timestamp as the global commit timestamp and sends it in a commit message to all participants

Participants replicate and commit the commit oplog entry

Coordinator waits for acks from all participants

Execute || Validate, Order || Persist

Execute and validate overlap. Execution stages operations at a shard; validation checks for w-w and prepare conflict to ensure atomic visibility.

Ordering uses the global commit timestamp sent by the coordinator. Shards use the commit-ts for persisting the txn and making it visible.

Reasoning & Verification is challenging

Distributed multi-doc txns was developed incrementally over several years

Sources of complexity include: aligning time across clusters, cross-layer interactions with WiredTiger, speculative majority reads, recovery protocol upon router failure, chunk migration by the catalog, interactions with DDL operations, fault-tolerance, etc.

TLA+ modeling of MongoDB transactions

First modeling of multi-shard database txns at scale

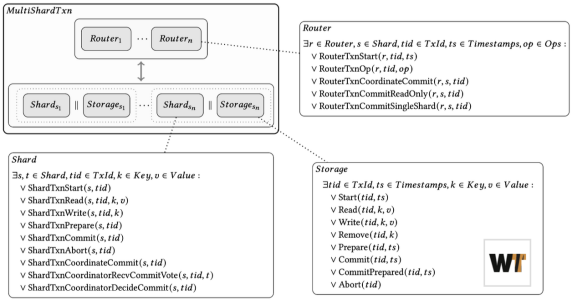
Specifies the txn behavior and isolation guarantees precisely & succinctly

Our modular approach enables automated model-based conformance testing of the WiredTiger storage implementation

Two modules

MultiShardTxn models the sharded transaction protocol

Storage models the underlying replication/storage layer at each shard



Initial states

Router

```
Init ≜
  ∧ rtxn = [r ∈ Router |-> [t ∈ TxId |-> 0]]
  ∧ rParticipants = [r ∈ Router |-> [t ∈ TxId |-> <<>>]]
  ∧ rTxnReadTs = [r ∈ Router |-> [t ∈ TxId |-> NoValue]]
  ∧ rInCommit = [r ∈ Router |-> [t ∈ TxId |-> FALSE]]
```

Shard

```
  ∧ shardTxnReqs = [s ∈ Shard |-> [t ∈ TxId |->  <<>>]]
  ∧ shardTxns = [s ∈ Shard |-> {}]
  ∧ shardPreparedTxns = [s ∈ Shard |-> {}]
  ∧ aborted = [s ∈ Shard |-> [t ∈ TxId |-> FALSE]]
  ∧ coordInfo = [s ∈ Shard |-> [t ∈ TxId |-> [self |-> FALSE, …]
  ∧ coordCommitVotes = [s ∈ Shard |-> [t ∈ TxId |-> {}]]
  ∧ shardOps = [s ∈ Shard |-> [t ∈ TxId |-> <<>>]]
```

Network

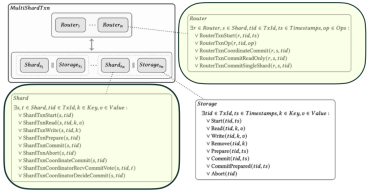
```
  ∧ msgsPrepare = {}
  ∧ msgsVoteCommit = {}
  ∧ msgsAbort = {}
  ∧ msgsCommit = {}
```

Global

```
  ∧ catalog ∈ [Keys -> Shard]
  ∧ ops = [s ∈ TxId |-> <<>>]
```

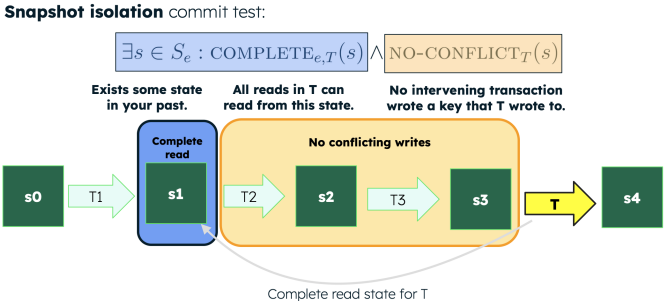
(Static mapping from keys to shards)

(Stores global transaction histories for isolation checking)



Isolation Checking

Transactions record operations into global history (ops), which is checked by state-based isolation checker Crooks et al. PODC'17 via Soethout's TLA+ lib.



RC:maj WC:maj returns fractured read t2:<k1=⊥, k2=t1>

1: Initial predicate
2: RouterTxnStart: t1 0
3: RouterTxnStart: t2 0
4: RouterTxnOp(r1 s1 t1 k1 "write")
5: RouterTxnOp(r1 s2 t1 k2 "write")
6: RouterTxnOp(r1 s1 t2 k1 "read")
7: RouterTxnOp(r1 s2 t2 k2 "read")
8: ShardTxnStart(s1 t1)
9: ShardTxnStart(s2 t1)
10: ShardTxnStart(s1 t2)
11: ShardTxnRead(s1 t2 k1)
12: ShardTxnWrite(s1 t1 k1)
13: ShardTxnWrite(s2 t1 k2)

Interactive trace exploration link on the browser

14: RouterTxnCoordCommit(r1 s1 t1)
15: ShardTxnCoordCommit(s1 t1)
16: ShardTxnPrepare(s1 t1)
17: ShardTxnPrepare(s2 t1)
18: ShardTxnCoordRecvCommitVote(s1 t1 s1)
19: ShardTxnCoordRecvCommitVote(s1 t1 s2)
20: ShardTxnCoordDecideCommit(s1 t1)
21: ShardTxnCommit(s1 t1)
22: ShardTxnCommit(s2 t1)
23: ShardTxnStart(s2 t2)
24: ShardTxnRead(s2 t2 k2)
25: RouterTxnCommitReadOnly(r1 s1 t2)
26: ShardTxnCommit(s1 t2)
27: ShardTxnCommit(s2 t2)

Precise WT timestamp/txn interactions are key for correctness

T2 should wait to observe T1's pending commit/update to k1!

1: start(t1 ts=0)
2: write(t1 k1)
3: prepare(t1 ts=1)
4: start(t2 ts=3)
5: commit(t1 ts=2)
6: read(t2 k1)

Model Checking (EC2 m6g.2xlarge)

- $Shard = \{s_1, s_2\}, Router = \{r_1\},$ $TxId = \{t_1, t_2\}, Key = \{k_1, k_2\}$
- **MaxStmts**=2
- **RC** = "snapshot"
- **INVARIANT** SnapshotIsolation

Depth = 35
8,408,701 distinct states (<10 minutes) ✓

- $Shard = \{s_1, s_2\}, Router = \{r_1\},$ $TxId = \{t_1, t_2\}, Key = \{k_1, k_2\}$
- **MaxStmts**=2
- **RC** = "local"
- **INVARIANT** ReadCommitted

Depth = 35
1,950,582 distinct states (<10 minutes) ✓

Modular Specification

Formally connect our high level protocol specification to lower level storage layer

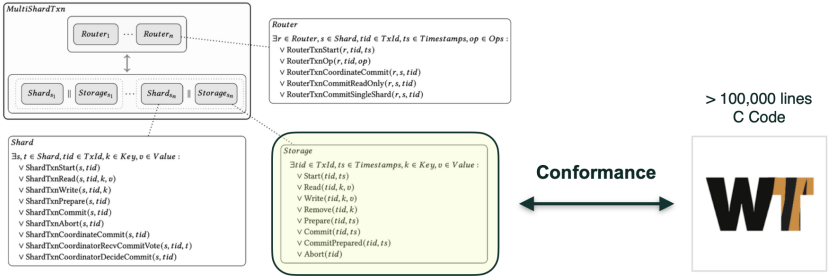
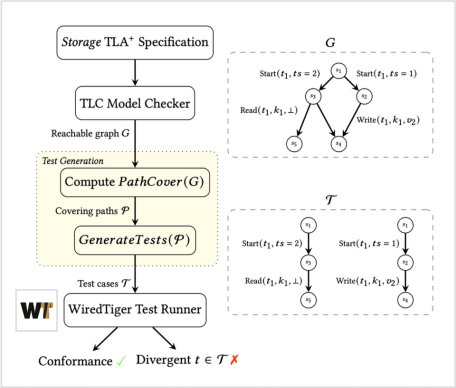


Figure 3: High level overview of our *MultiShardTxn* transactions specification, showing each logical component. Actions of the *Shard* component compose synchronously (indicated by ||) with corresponding, lower level actions of the *Storage* model, while actions of the *Router* and *Shard* interact asynchronously.

Test case generation

Our Python tool

- processes the TLC-output state graph
- computes path coverings
- translates each path into unit test code that drives WiredTiger API
- checks for conformance of model and implementation values



Model-based Verification

Model	$Keys = \{k1, k2\}$ $TxIds = \{t1, t2\}$ $Timestamp = \{1 \ldots 3\}$
States	490,360
States (symmetry)	132,981
Tests	87,143
Mean Depth	15
State graph generation (TLC)	1m 11s
Test Generation	29m 10s
Test Execution	13m 49s
Conformance	✓

Generate and run test cases against WiredTiger

Permissiveness

The degree of concurrency a txn protocol allows under a given isolation level

Read Concern	Write Conflicts	Prepare Conflicts	Permissiveness
SI definition	-	-	1.0
“snapshot”	Yes	Yes	0.81

Snapshot Isolation

Read Concern	Write Conflicts	Prepare Conflicts	Permissiveness
RC definition	-	-	1.0
“local”	No	No	0.792
“local”	Yes	No	0.790
“local”	Yes	Yes	0.76

Read Committed

MongoDB default

Permissiveness

Schedule prevented by prepare conflict blocking, permitted under read committed.

```
t1 : << [op |-> "read", key |-> k2, value |-> NoValue],
        [op |-> "read", key |-> k2, value |-> t2]>> @@

t2 : << [op |-> "write", key |-> k2, value |-> t2]>>
```

(Non-repeatable read)

Read Concern	Write Conflicts	Prepare Conflicts	Permissiveness
RC definition	-	-	1.0
“local”	No	No	0.792
“local”	Yes	No	0.790
“local”	Yes	Yes	0.76

Read Committed

MongoDB default

Future Work

Add catalog modeling for correctness under chunk migration

Extend multigrain modular modeling to more protocols

Generate test cases from TLA+ to bridge spec and code

Check transaction permissiveness to guide protocol optimization