Verifying Transactional Consistency of MongoDB

(submitted to VLDB'2022)

Hengfeng Wei

hfwei@nju.edu.cn

December 27, 2021

mongoDB

MongoDB 的三种经典部署架构

MongoDB 3.0	MongoDB 3.2	MongoDB 3.4	MongoDB 3.6	MongoDB 4.0	MongoDB 4.2
New Storage engine (WiredTiger)	Enhanced replication protocol: stricter consistency & durability	Shard membership awareness	Consistent secondary reads in sharded clusters	Replica Set Transactions	Distributed Transactions
	WiredTiger default storage engine		Logical sessions	Make catalog timestamp-aware	Oplog applier prepare support
	Config server manageability improvements		Retryable writes	Snapshot reads	Distributed commit protocol
	Read concern "majority"		Causal Consistency	Recoverable rollback via WT checkpoints	Global point-in-time reads
			Cluster-wide logical clock	Recover to a timestamp	More extensive WiredTiger repair
			Storage API to changes to use timestamps	Sharded catalog improvements	Transaction manager
			Read concern majority feature always available		
			Collection catalog versioning		
			UUIDs in sharding		
			Fast in-place updates to large documents in WT		

MongoDB 事务的三阶段发展过程

A Fundamental Question:

What transactional consistency guarantee do MongoDB transactions in each deployment provide?

挑战一：MongoDB 官方规约不清楚, SI 有多种变体

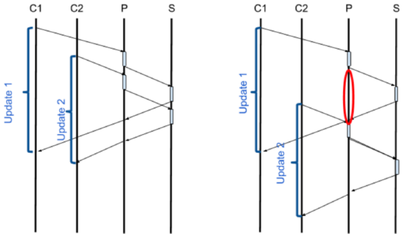
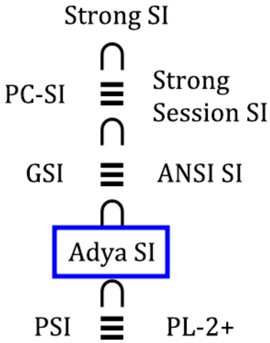
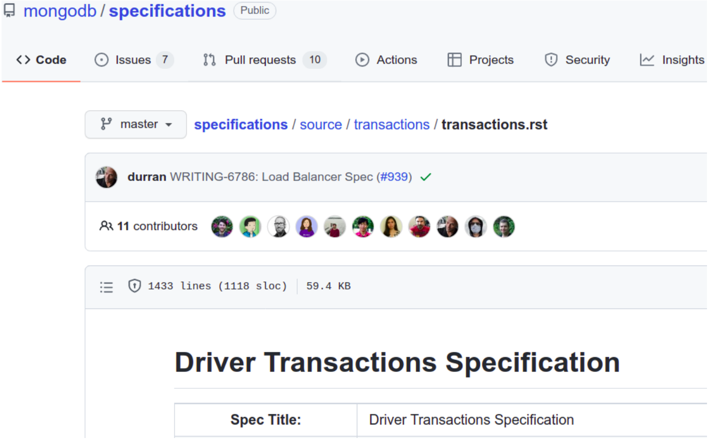


Figure 3: Back-to-Back Transactions with and without Speculative Snapshot Isolation



挑战二：MongoDB 缺少精简的事务协议描述，更没有严格证明

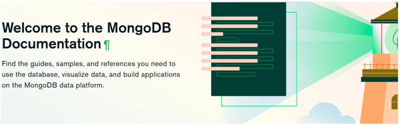


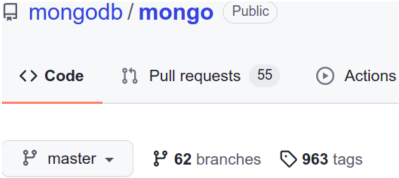
挑战三: SI 检测问题是 NP-complete 问题, 复杂度高

THEOREM 3.2. *For any criterion* $C \in \{$PREFIX CONSISTENCY, SNAPSHOT ISOLATION, SERIALIZABILITY$\}$ *the problem of checking whether a given history satisfies* C *is NP-complete.*

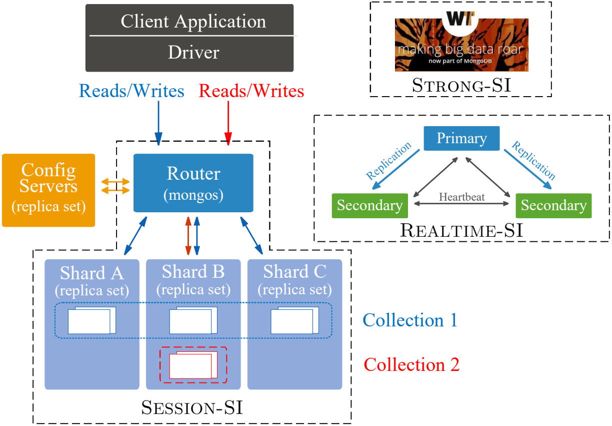
贡献一: 使用 (VIS, AR) 框架, 为多种 SI 变体提供形式化规约

贡献二：为 MongoDB 事务一致性协议提供精简的伪代码描述





贡献三：证明 WiredTiger、ReplicaSet、ShardedCluster 事务协议分别满足 StrongSI、RealtimeSI、SessionSI 变体



贡献四: 设计并评估了多项式时间 SI 变体白盒检测算法

JEPSEN

1. 事务 $T : (E, \mathsf{po})$
   - po : Program Order
   - $start(T)$: 事务开始时间
   - $commit(T)$: 事务提交时间

2. 历史 $\mathcal{H} : (\mathbb{T}, \textsc{so})$
   - $\mathbb{T}$: 已提交事务集合
   - $\textsc{so}$: Session Order

3. 执行 $\mathcal{A} : (\mathcal{H}, \textsc{vis}, \textsc{ar})$
   - $\textsc{vis}$: 可见性 (Visibility) 偏序关系
   - $\textsc{ar}$: 仲裁 (Arbitration) 全序关系
   - $\textsc{vis} \subseteq \textsc{ar}$

一个事务一致性模型可定义为一组一致性公理的集合 Φ。

历史 $\mathcal{H}$ 满足事务一致性模型 Φ, 如果存在 $\textsc{vis}$ 与 $\textsc{ar}$ 使得

$$\exists \textsc{vis}, \textsc{ar}.\ (\mathcal{H}, \textsc{vis}, \textsc{ar}) \models \Phi。$$

$\forall (E, \mathsf{po}) \in \mathcal{H}.\ \forall e \in \mathsf{Event}.\ \forall key, val.\ \big(\mathsf{op}(e) = \mathsf{read}(key, val) \land \{f \mid (\mathsf{op}(f) = \_(key, \_) \land f \xrightarrow{\mathsf{po}} e\} \neq \emptyset\big) \implies \mathsf{op}(\max_{\mathsf{po}}\{f \mid \mathsf{op}(f) = \_(key, \_) \land f \xrightarrow{\mathsf{po}} e\}) = \_(key, val)$ (INT)	
$\forall T \in \mathcal{H}.\ \forall key, val.\ T \vdash \mathsf{read}(key, val) \implies \max_{\mathrm{AR}}(\mathrm{VIS}^{-1}(T) \cap \mathsf{WriteTx}_{key}) \vdash \mathsf{write}(key, val)$ (EXT)	
$\mathrm{SO} \subseteq \mathrm{VIS}$ (SESSION)	$\mathrm{AR} \,;\, \mathrm{VIS} \subseteq \mathrm{VIS}$ (PREFIX)
$\mathrm{RB} \subseteq \mathrm{VIS}$ (RETURNBEFORE)	$\mathrm{CB} \subseteq \mathrm{AR}$ (COMMITBEFORE)
$\mathrm{VIS} \subseteq \mathrm{RB}$ (REALTIMESNAPSHOT)	$\forall S, T \in \mathcal{H}.\ S \bowtie T \implies (S \xrightarrow{\mathrm{VIS}} T \lor T \xrightarrow{\mathrm{VIS}} S)$ (NOCONFLICT)

$$\text{SI} = \textsc{Int} \land \textsc{Ext} \land \textsc{Prefix} \land \textsc{NoConflict}$$

$$\textsc{SessionSI} = \text{SI} \land \textsc{Session}$$

$$\textsc{SessionSI} = \text{SI} \land \textsc{Session}$$

$$\textsc{RealtimeSI} = \text{SI} \land \textsc{ReturnBefore} \land \textsc{CommitBefore}$$

$$\textsc{SessionSI} = \text{SI} \land \textsc{Session}$$

$$\textsc{RealtimeSI} = \text{SI} \land \textsc{ReturnBefore} \land \textsc{CommitBefore}$$

$$\text{GSI} = \text{SI} \land \textsc{RealtimeSI} \land \textsc{CommitBefore}$$

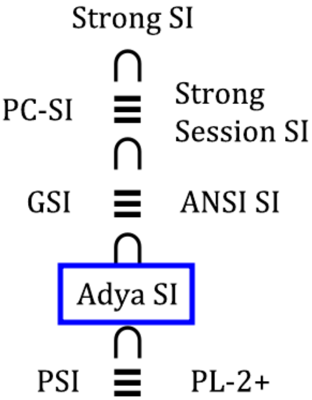
$$\textsc{SessionSI} = \text{SI} \land \textsc{Session}$$

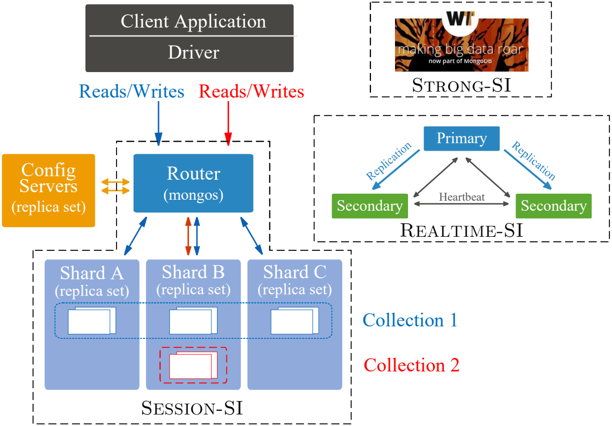
$$\textsc{RealtimeSI} = \text{SI} \land \textsc{ReturnBefore} \land \textsc{CommitBefore}$$

$$\text{GSI} = \text{SI} \land \textsc{RealtimeSI} \land \textsc{CommitBefore}$$

$$\textsc{StrongSI} = \text{GSI} \land \textsc{ReturnBefore}$$

- ANSI-SI
- SI
- GSI
- StrongSI
- StrongSessionSI
- PSI
- WriteSI
- NMSI
- PCSI





重点在于如何确定每个事务的 “读快照” (Read Snapshot),

也就是对该事务可见的所有事务构成的集合

$\textsc{WiredTiger} \models \textsc{StrongSI}$

每个 WiredTiger 事务 $txn \in \mathsf{WT\_TXN}$ 有一个唯一标识号

$$txn.\mathsf{tid} \in \mathsf{TID} = \mathbb{N} \cup \{-1, \perp_{\mathsf{tid}}\}$$

- 事务开始时 (WT_START), $txn.\mathsf{tid} = 0$
- 事务第一个写操作成功执行后 (WT_UPDATE), $txn.\mathsf{tid} > 0$
- 事务因写冲突回滚时 (WT_ROLLBACK), $txn.\mathsf{tid} = -1$

每个 WiredTiger 事务 $txn \in \mathsf{WT\_TXN}$ 有一个唯一标识号

$$txn.\mathsf{tid} \in \mathsf{TID} = \mathbb{N} \cup \{-1, \bot_{\mathsf{tid}}\}$$

- 事务开始时 (WT_START), $txn.\mathsf{tid} = 0$
- 事务第一个写操作成功执行后 (WT_UPDATE), $txn.\mathsf{tid} > 0$
- 事务因写冲突回滚时 (WT_ROLLBACK), $txn.\mathsf{tid} = -1$
- $\bot_{\mathsf{tid}}$ 表示不存在这个事务

每个 WiredTiger 事务 $txn \in \textsf{WT\_TXN}$ 有一个唯一标识号

$$txn.\textsf{tid} \in \textsf{TID} = \mathbb{N} \cup \{-1, \bot_{\textsf{tid}}\}$$

- 事务开始时 (WT_START), $txn.\textsf{tid} = 0$
- 事务第一个写操作成功执行后 (WT_UPDATE), $txn.\textsf{tid} > 0$
- 事务因写冲突回滚时 (WT_ROLLBACK), $txn.\textsf{tid} = -1$
- $\bot_{\textsf{tid}}$ 表示不存在这个事务

对于只读事务 txn, 始终有 $txn.\textsf{tid} = 0$

- 客户端通过会话 (Session) 与 WiredTiger 进行交互
- 每个会话有一个唯一会话标识号 $wt\_sid \in \mathsf{WT\_SID} = \mathbb{N}$

- 客户端通过会话 (Session) 与 WiredTiger 进行交互
- 每个会话有一个唯一会话标识号 $wt\_sid \in \mathsf{WT\_SID} = \mathbb{N}$
- WiredTiger 维护数据结构
  $\mathsf{wt\_txn\_global} \in [\mathsf{current\_tid} : \mathsf{TID}, \mathsf{states} : \mathsf{WT\_SID} \to \mathsf{TID}]$

  current_tid : 当前分配的最大事务标识号

  states : 会话与会话之上当前事务之间的映射关系

- 客户端通过会话 (Session) 与 WiredTiger 进行交互
- 每个会话有一个唯一会话标识号 $wt\_sid \in \mathsf{WT\_SID} = \mathbb{N}$
- WiredTiger 维护数据结构
  $\mathsf{wt\_txn\_global} \in [\mathsf{current\_tid} : \mathsf{TID}, \mathsf{states} : \mathsf{WT\_SID} \to \mathsf{TID}]$

  current_tid : 当前分配的最大事务标识号

  states : 会话与会话之上当前事务之间的映射关系

事务 txn 提交或回滚时, $\mathsf{wt\_txn\_global.states}[wt\_sid] \leftarrow \bot_{\mathsf{txn}}$

每个事务只能观察到在它开始之前提交的事务

$$\textsc{WiredTiger} \models \textsc{RealtimeSI}$$

每个事务只能观察到在它开始之前提交的事务

WiredTiger $\models$ RealtimeSI

每个事务在开始时 (wt_start) 根据

wt_txn_global 维护的信息确定它的 “读快照”

WiredTiger 事务协议从反面入手计算, 排除不可见事务集合

每个事务 txn 维护以下信息:

txn.snapshot : 正在进行的、已获取事务标识号的事务集合

txn.snap_max : txn 开始时, 当前最大的事务标识号

$$\text{wt\_txn\_global} \in [\text{current\_tid} : \text{TID}, \text{states} : \text{WT\_SID} \to \text{TID}]$$

procedure TXN_VISIBLE(txn, tid)
return $\neg(tid = -1 \lor tid \in txn.\mathsf{snapshot} \lor (tid \geq txn.\mathsf{snap\_max} \land tid \neq txn.\mathsf{tid}))$

对于事务 txn, 满足以下条件的、标识号等于 tid 的事务对 txn 不可见:

- $tid = -1$: 该事务已回滚
- $tid \in txn.snapshot$: 该事务与 txn 并发
- $\mathsf{tid} \geq txn.\mathsf{snapshot} \land \mathsf{tid} \neq txn.\mathsf{tid}$:
  - $\mathsf{tid} \geq txn.\mathsf{snapshot}$: 该事务开始得比 txn 晚
  - $\mathsf{tid} \neq txn.\mathsf{tid}$: 允许 txn 观察到自身

每个事务观察到在它开始之前提交的事务

Definition (Visibility Relation)

$$\text{VIS}_{\text{WT}} \triangleq \textsc{ReturnBefore}.$$

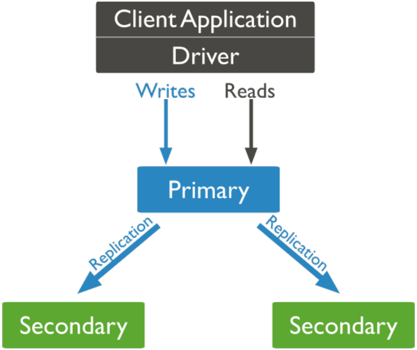
在逻辑上, 所有事务按实时序依次提交

Definition (Arbitration Relation)

$$\text{AR}_{\text{WT}} \triangleq \textsc{CommitBefore}.$$

Lemma (冲突事务的提交顺序)

$$\forall txn, txn' \in \mathsf{WT\_TXN}.$$
$$txn \bowtie txn' \implies \left(txn \xrightarrow{AR_{WT}} txn' \iff txn.\mathsf{tid} < txn'.\mathsf{tid}\right).$$



$$\textsc{ReplicaSet} \models \textsc{RealtimeSI}$$

$$readConcern = \text{“}snapshot\text{”} \text{ and } writeConcern = \text{“}majority\text{”}$$

$readConcern = \text{“}snapshot\text{”}$: 保证事务读取到一致性的、被多数节点提交的快照

$writeConcern = \text{“}majority\text{”}$: 保证事务中的写操作以及读到的数据被多数节点提交 (majority committed)

主节点维护 oplog, 并负责决定事务的提交顺序

主节点维护 oplog, 并负责决定事务的提交顺序

每个事务 txn 被赋予一个唯一的逻辑提交时间戳 txn.commit_ts

这些逻辑时间戳确定了事务在 ReplicaSet 层的 (逻辑) 提交顺序

当事务 txn 开始时 (OPEN_WT_SESSION),

计算它的 “读时间戳” (txn.read_ts)

条件: oplog 中所有提交时间戳小于 txn.read_ts 的事务均已提交

(txn.read_ts 保证可见的事务在 oplog 中不造成空洞)

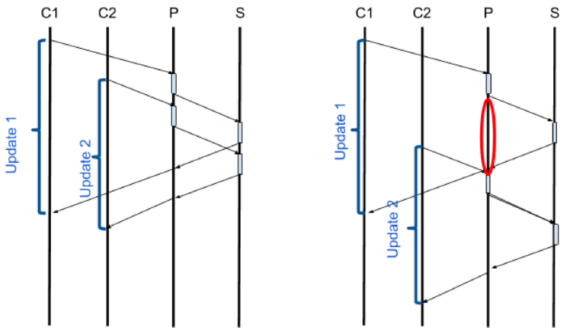


Figure 3: Back-to-Back Transactions with and without Speculative Snapshot Isolation

- 事务读取 WiredTiger 层当前最新的数据, 而不限于已被多数节点提交的数据
- 当事务提交时, 等待读取到的数据被多数节点提交

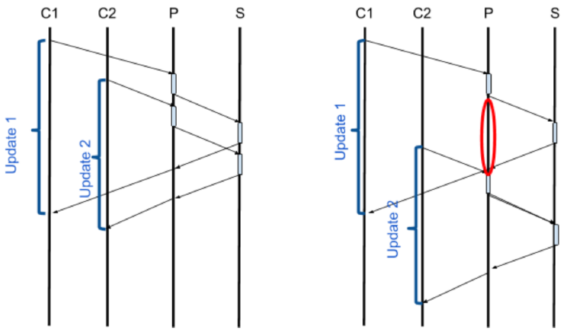


Figure 3: Back-to-Back Transactions with and without Speculative Snapshot Isolation

- 事务读取 WiredTiger 层当前最新的数据，而不限于已被多数节点提交的数据
- 当事务提交时，等待读取到的数据被多数节点提交

只读事务在提交时发起 “noop” 操作，然后等待该操作被多数节点提交

(physical component, logical component)

Coarse synchronization of time intervals using NTP

Order within coarse intervals with a monotonic counter

---

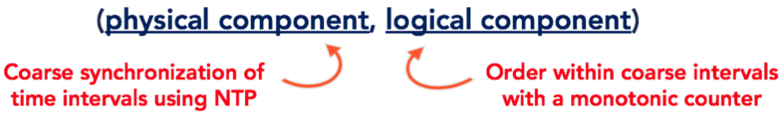
procedure TICK()

    if $\mathsf{ct}.sec \geq \mathsf{clock}$ **then**

        return $\langle \mathsf{ct}.sec, \mathsf{ct}.counter + 1 \rangle$

    else

        return $\langle \mathsf{clock}, 0 \rangle$




```
procedure TICK()
    if ct.sec ≥ clock then
        return ⟨ct.sec, ct.counter + 1⟩
    else
        return ⟨clock, 0⟩
```

- 主节点的 cluster time (ct) 仅在产生新的 oplog 项时增加
- 每个消息都携带发送方的 ct
- 每个节点 (包括客户端) 维护它已知的最大 ct

visibility

Definition (Visibility Relation)

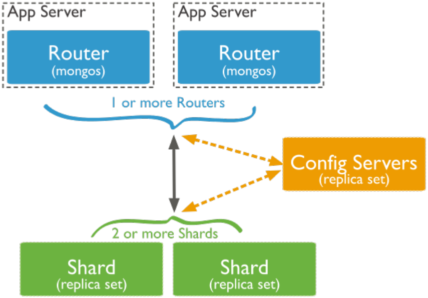
$$\forall txn_1, txn_2 \in \mathsf{RS\_TXN}.\ txn_1 \xrightarrow{\text{VIS}_{\text{RS}}} txn_2 \iff txn_1.\mathsf{commit\_ts} \le txn_2.\mathsf{read\_ts}.$$

Definition (Arbitration Relation)

Order all update transactions in $\textsc{ar}_{\text{RS}}$ by their `commit_ts`. For each client, we insert its read-only transactions one by one in session order: each read-only transaction txn on session $rs\_session$ is placed immediately after the later (in $\textsc{ar}_{\text{RS}}$) of

(1) the previous (in $\textsc{so}_{\text{RS}}$) transaction of txn on session $rs\_session$, if any; and

(2) the update transactions txn' (in $\textsc{ar}_{\text{RS}}$) such that $txn' \xrightarrow{\text{VIS}_{\text{RS}}} txn$.



$$\textsc{ShardedCluster} \models \textsc{SessionSI}$$

Conclusion

Hengfeng Wei (hfwei@nju.edu.cn)