

Verifying Transactional Consistency of MongoDB

(submitted to VLDB'2022)

Hengfeng Wei

hfwei@nju.edu.cn

December 28, 2021





mongo

Transaction

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

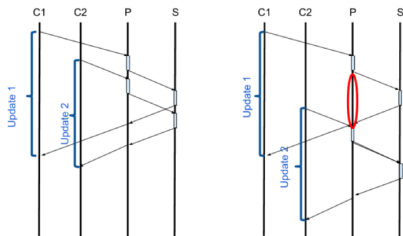
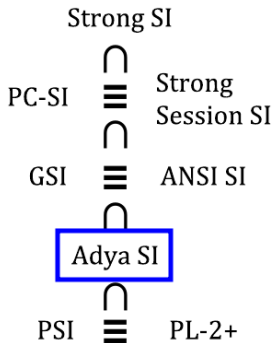


Figure 3: Back-to-Back Transactions with and without **Speculative Snapshot Isolation**



挑战二: MongoDB 缺少精简的事务协议描述, 更没有严格证明

The screenshot displays the GitHub interface for the `mongodb/specifications` repository. The repository is public and has 7 issues, 10 pull requests, and various actions, projects, security checks, and insights. The current view is the `specifications / source / transactions / transactions.rst` file, which is part of the `master` branch. The file is titled `durran WRITING-6786: Load Balancer Spec (#939)` and has 11 contributors. The file size is 59.4 KB and contains 1433 lines (1118 sloc). The main heading of the document is **Driver Transactions Specification**. Below the heading, there is a table with the following structure:

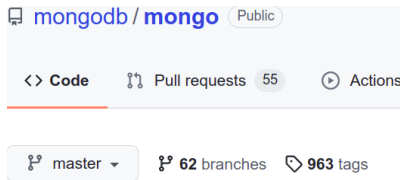
| Spec Title: | Driver Transactions Specification |
|-------------|-----------------------------------|
|-------------|-----------------------------------|

挑战三: SI 检测问题是 NP-complete 问题, 复杂度高

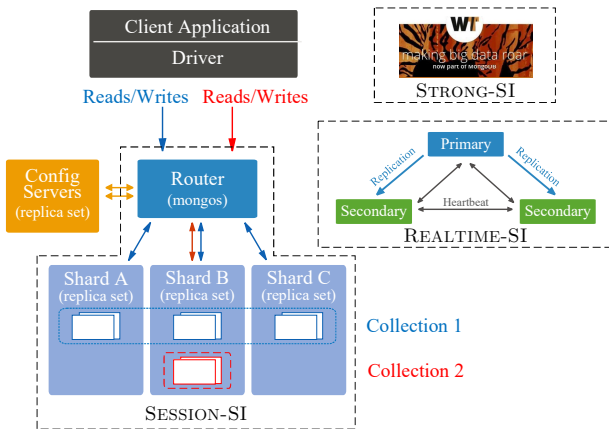
THEOREM 3.2. *For any criterion $C \in \{\text{PREFIX CONSISTENCY}, \text{SNAPSHOT ISOLATION}, \text{SERIALIZABILITY}\}$ the problem of checking whether a given history satisfies C is NP-complete.*

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、REALTIME-SI、SESSION-SI 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 $T : (E, po)$

- ▶ po : 程序序 (Program Order)
- ▶ $start(T)$: 事务开始时间
- ▶ $commit(T)$: 事务提交时间

2. 历史 $\mathcal{H} : (\mathbb{T}, so)$

- ▶ \mathbb{T} : 已提交事务集合
- ▶ so : 会话序 (Session Order)

3. 执行 $\mathcal{A} : (\mathcal{H}, vis, ar)$

- ▶ vis : 可见 (Visibility) 偏序关系
- ▶ ar : 仲裁 (Arbitration) 全序关系
- ▶ $vis \subseteq ar$

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 VIS 与 AR 使得

$$\exists \text{VIS, AR. } (\mathcal{H}, \text{VIS, AR}) \models \Phi$$

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

$$S \xrightarrow{\text{RB}} T \iff \text{commit}(S) < \text{start}(T)$$

$$S \xrightarrow{\text{CB}} T \iff \text{commit}(S) < \text{commit}(T)$$

$$SI = INT \wedge EXT \wedge PREFIX \wedge NoCONFLICT$$

$$\text{SESSIONSI} = \text{SI} \wedge \text{SESSION}$$

$$\text{SESSIONSI} = \text{SI} \wedge \text{SESSION}$$

$$\text{REALTIMESI} = \text{SI} \wedge \text{RETURNBEFORE} \wedge \text{COMMITBEFORE}$$

$$\text{SESSIONSI} = \text{SI} \wedge \text{SESSION}$$

$$\text{REALTIMESI} = \text{SI} \wedge \text{RETURNBEFORE} \wedge \text{COMMITBEFORE}$$

$$\text{GSI} = \text{SI} \wedge \text{REALTIMESI} \wedge \text{COMMITBEFORE}$$

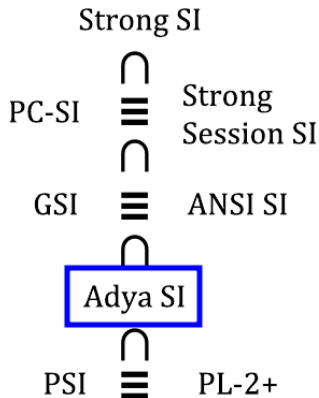
$$\text{SESSIONSI} = \text{SI} \wedge \text{SESSION}$$

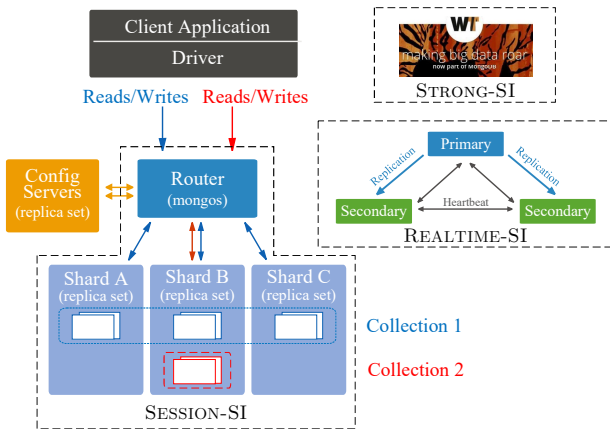
$$\text{REALTIMESI} = \text{SI} \wedge \text{RETURNBEFORE} \wedge \text{COMMITBEFORE}$$

$$\text{GSI} = \text{SI} \wedge \text{REALTIMESI} \wedge \text{COMMITBEFORE}$$

$$\text{STRONGSI} = \text{GSI} \wedge \text{RETURNBEFORE}$$

- ▶ 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



$\text{WIREDTIGER} \models \text{STRONGSI}$

每个 WIREDTIGER 事务 $txn \in \text{WT_TXN}$ 有一个唯一标识号

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

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

每个 WIREDTIGER 事务 $txn \in \text{WT_TXN}$ 有一个唯一标识号

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

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

每个 WIREDTIGER 事务 $txn \in \text{WT_TXN}$ 有一个唯一标识号

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

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

对于只读事务 txn , 始终有 $txn.tid = 0$

- ▶ 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 $wt_sid \in WT_SID = \mathbb{N}$

- ▶ 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 $wt_sid \in WT_SID = \mathbb{N}$
- ▶ WiredTiger 维护数据结构
 $wt_txn_global \in [current_tid : TID, states : WT_SID \rightarrow TID]$
 $current_tid$: 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 $wt_sid \in WT_SID = \mathbb{N}$
- ▶ WiredTiger 维护数据结构
$$wt_txn_global \in [current_tid : TID, states : WT_SID \rightarrow TID]$$

$current_tid$: 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, $wt_txn_global.states[wt_sid] \leftarrow \perp_{txn}$

每个事务只能观察到在它开始之前提交的事务

$$\text{WIREDTIGER} \models \text{REALTIMESI}$$

每个事务只能观察到在它开始之前提交的事务

$$\text{WIREDTIGER} \models \text{REALTIMESI}$$

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

$wt_txn_global \in [current_tid : TID, states : WT_SID \rightarrow TID]$

```
1: procedure TXN_VISIBLE(txn, tid)
2:   return  $\neg(tid = -1 \vee tid \in txn.snapshot \vee (tid \geq$   
    $txn.snap\_max \wedge tid \neq txn.tid))$ 
```

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
- ▶ $tid \geq txn.snapshot \wedge tid \neq txn.tid$:
 - ▶ $tid \geq txn.snapshot$: 该事务开始得比 txn 晚
 - ▶ $tid \neq txn.tid$: 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 (可见关系 VIS_{WT})

$$\text{VIS}_{\text{WT}} \triangleq \text{RETURNBEFORE}$$

在逻辑上, 所有事务按实时序依次提交

Definition (仲裁关系 AR_{WT})

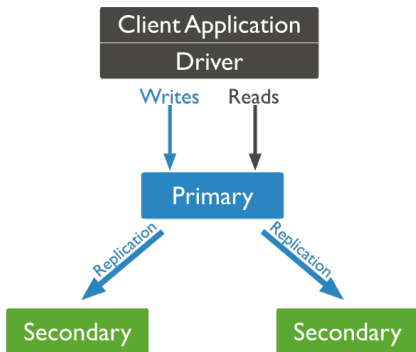
$$\text{AR}_{\text{WT}} \triangleq \text{COMMITBEFORE}$$

Lemma (冲突事务的提交顺序)

$$\forall txn, txn' \in WT_TXN.$$

$$txn \bowtie txn' \implies (txn \xrightarrow{AR_{WT}} txn' \iff txn.tid < txn'.tid).$$

该引理对 STRONGSI 检测算法有辅助作用



$\text{REPLICASET} \models \text{REALTIMESI}$

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

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

$writeConcern = \text{“majority”}$: 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)

```
1: procedure TXN_VISIBLE( $txn, tid, ts$ )  
2:   return  $\neg(tid = -1 \vee tid \in txn.snapshot \vee (tid \geq$   
    $txn.snap\_max \wedge tid \neq txn.tid)) \wedge (ts \neq \perp_{ts} \wedge ts \leq txn.read\_ts)$ 
```

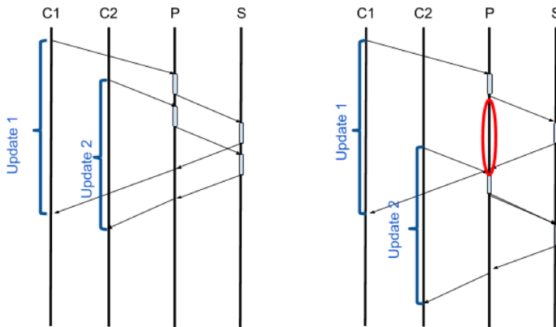


Figure 3: Back-to-Back Transactions with and without **Speculative Snapshot Isolation**

- ▶ 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

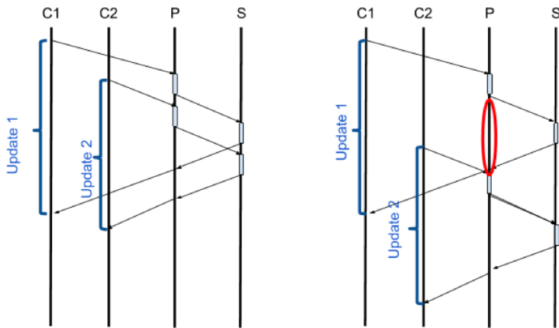


Figure 3: Back-to-Back Transactions with and without Speculative Snapshot Isolation

- ▶ 事务读取 WiredTiger 层当前最新的数据, 而不同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

```
1: procedure TICK()  
2:   if  $ct.sec \geq clock$  then  
3:     return  $\langle ct.sec, ct.counter + 1 \rangle$   
4:   else  
5:     return  $\langle clock, 0 \rangle$ 
```

(physical component, logical component)

Coarse synchronization of
time intervals using NTP



Order within coarse intervals
with a monotonic counter

```
1: procedure TICK()  
2:   if  $ct.sec \geq clock$  then  
3:     return  $\langle ct.sec, ct.counter + 1 \rangle$   
4:   else  
5:     return  $\langle clock, 0 \rangle$ 
```

- ▶ 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

Definition (可见关系 VIS_{RS})

$$\forall txn_1, txn_2 \in \text{RS_TXN}. \quad txn_1 \xrightarrow{\text{VIS}_{\text{RS}}} txn_2 \iff \\ txn_1.\text{commit_ts} \leq txn_2.\text{read_ts}.$$

Lemma (WiredTiger 与 ReplicaSet 层可见关系)

$$VIS_{RS} \subseteq VIS_{WT}$$

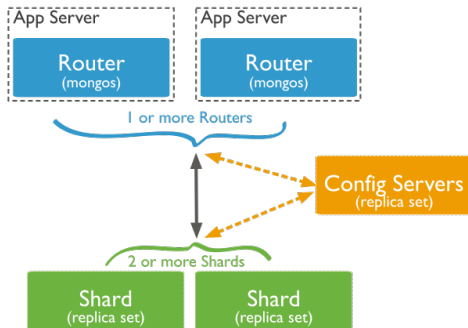
REPLICASET 使用逻辑时间戳“覆写”(override)了
WIREDTIGER 中的事务标识号

Definition (仲裁关系 AR_{RS})

- ▶ 非只读事务按照它们的 $commit_ts$ 在 AR_{RS} 中排序
- ▶ 对于每个客户端, 将只读事务按照会话序依次插入 AR_{RS} 。

具体而言, 会话 $rs_session$ 上的只读事务 txn 紧随以下事务之后:

- ▶ txn 在 so_{RS} 序下的前一个事务 (如果有),
- ▶ 非只读事务 txn' 满足 $txn' \xrightarrow{VIS_{RS}} txn$ 。

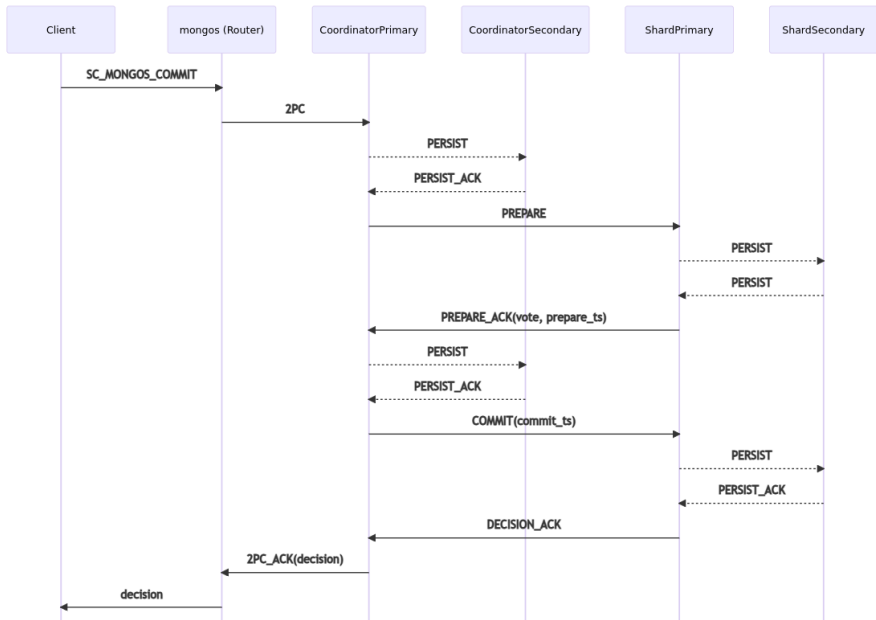


$\text{SHARDEDCLUSTER} \models \text{SESSIONSI}$

分布式事务

不同分区间使用 2PC 协议, 保证原子性

同一分区间使用共识复制协议, 提高容错性



monogs 为事务指定快照时间戳 (read_ts)

逻辑时间戳是松散同步的, 在某些分区上该快照尚不可得

monogs 为事务指定快照时间戳 ($read_ts$)

逻辑时间戳是松散同步的, 在某些分区上该快照尚不可得

CASE-CLOCK-SKEW: 如果 $ct < read_ts$, 则先推进 ct

CASE-PENDING-COMMIT-READ: 如果可见的写操作是
WT_PREPARED 状态, 则重试读

CASE-PENDING-COMMIT-UPDATE 如果可见的写操作是
WT_PREPARED 状态, 则先重试读

CASE-HOLES: 等待直到 oplog 中 $commit_ts \leq read_ts$ 的事务均提交

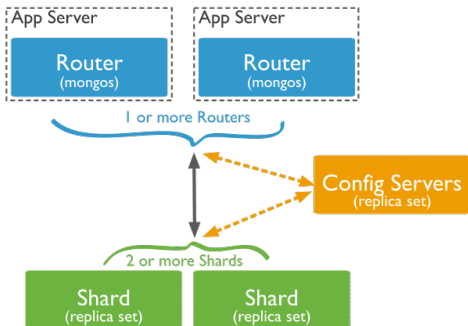
Definition (可见关系 VIS_{SC})

$$\forall txn_1, txn_2 \in \text{SC_TXN}. \quad txn_1 \xrightarrow{\text{VIS}_{\text{SC}}} txn_2 \iff \\ txn_1.\text{commit_ts} \leq txn_2.\text{read_ts}.$$

Definition (仲裁关系 AR_{SC})

- ▶ 非只读事务按照它们的 $commit_ts$ 在 AR_{SC} 中排序
 - ▶ 相同的 $commit_ts$ 根据事务协调者所在的分区 $shard_id$ 定序
- ▶ 对于每个客户端, 将只读事务按照会话序依次插入 AR_{SC} 。
具体而言, 会话 $sc_session$ 上的只读事务 txn 紧随以下事务之后:
 - ▶ txn 在 SO_{SC} 序下的前一个事务 (如果有),
 - ▶ 非只读事务 txn' 满足 $txn' \xrightarrow{VIS_{SC}} txn$ 。

$\text{SHARDEDCLUSTER} \models \text{SESSIONSI}$



$$\text{SO}_{\text{SC}} \subseteq \text{VIS}_{\text{SC}}$$

由节点维护并分发逻辑时间戳的机制保障

Conclusion



Hengfeng Wei (hfwei@nju.edu.cn)