

Verifying Transactional Consistency of MongoDB

(submitted to VLDB'2022)

Hengfeng Wei

hfwei@nju.edu.cn

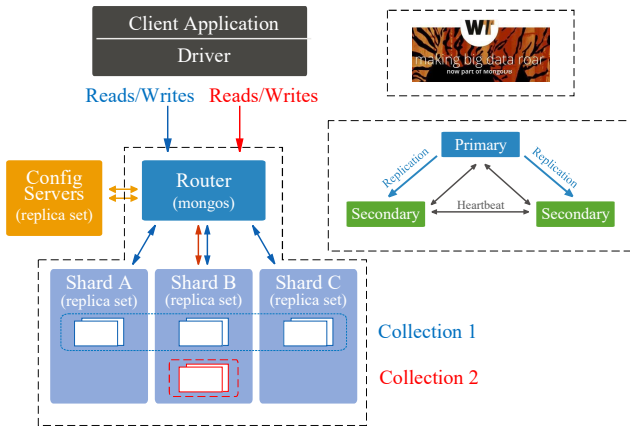
December 29, 2021





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

一个基本问题:

每种部署下, MongoDB 事务协议提供怎样的事务一致性?

挑战一：MongoDB 官方规约不清楚, SI 有多种变体

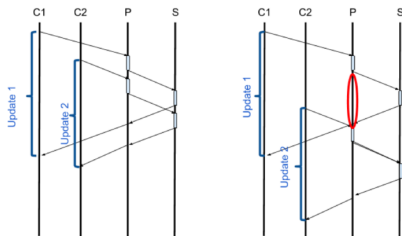
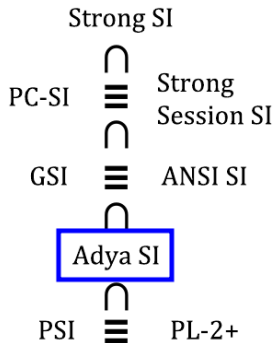


Figure 3: Back-to-Back Transactions with and without **Speculative Snapshot Isolation**



挑战二: MongoDB 缺少精简的事务协议描述, 更没有严格证明

The screenshot shows the GitHub interface for the `mongodb/specifications` repository. The breadcrumb navigation indicates the path: `specifications / source / transactions / transactions.rst`. The commit message is "durran WRITING-6786: Load Balancer Spec (#939)" with a green checkmark. Below the commit message, there are 11 contributor avatars. The file statistics show 1433 lines (1118 sloc) and 59.4 KB. The main heading of the document is "Driver Transactions Specification". At the bottom, a table lists the specification title.

mongodb / specifications Public

<> Code Issues 7 Pull requests 10 Actions Projects Security Insights

master specifications / source / transactions / transactions.rst

durran WRITING-6786: Load Balancer Spec (#939) ✓

11 contributors

1433 lines (1118 sloc) | 59.4 KB

Driver Transactions Specification

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

挑战三: 在 Version Order 未知的情况下,
SI 检测问题是 NP-complete 的

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



SESSIONSI

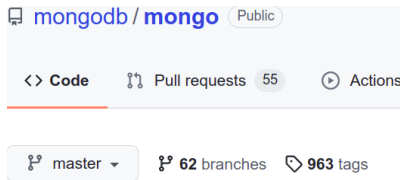
REALTIMESI

GSI

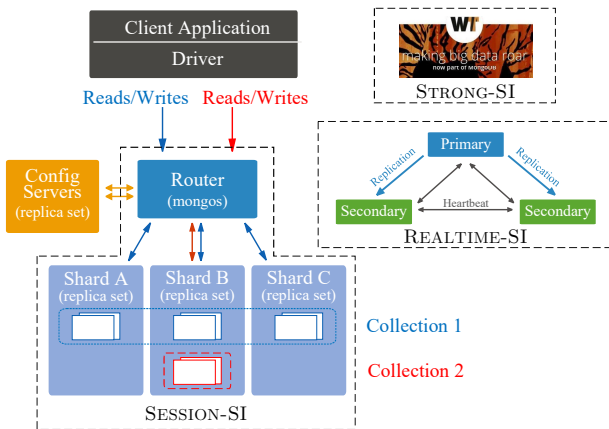
STRONGSI

...

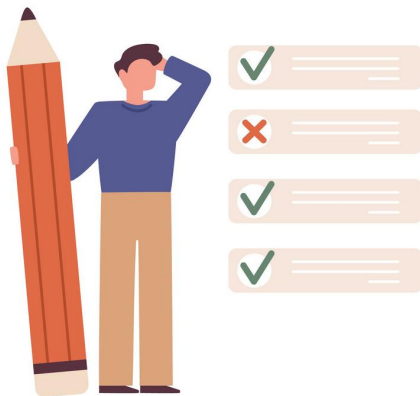
贡献二：为 MongoDB 事务一致性协议提供精简的伪代码描述



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





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

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

$$\text{READATOMIC} = \text{INT} \wedge \text{EXT}$$

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

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

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

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

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

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

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

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

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

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

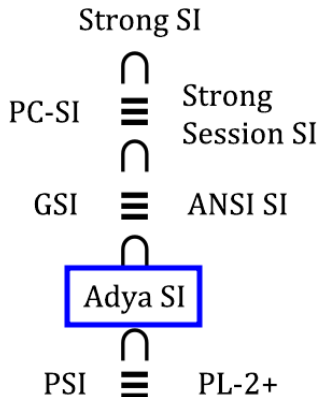
$$\text{SESSIONSI} = \text{SI} \wedge \text{SESSION}$$

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

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

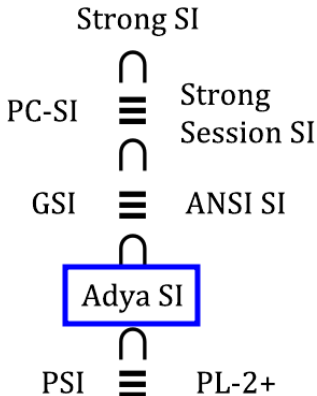
$$\text{STRONGSI} = \text{GSI} \wedge \text{RETURNBEFORE}$$

- ▶ ANSI-SI
- ▶ SI
- ▶ GSI
- ▶ STRONGSI
- ▶ STRONGSESSIONSI
- ▶ PSI
- ▶ WRITE SI
- ▶ NMSI
- ▶ PC-SI



$$\text{TRANSVIS} \triangleq (\text{VIS}^+ = \text{VIS})$$

- ▶ ANSI-SI
- ▶ SI
- ▶ GSI
- ▶ STRONGSI
- ▶ STRONGSESSIONSI
- ▶ PSI
- ▶ WRITESI
- ▶ NMSI
- ▶ PC-SI



$$\text{PSI} = \text{INT} \wedge \text{EXT} \wedge \text{TRANSVIS} \wedge \text{NOCONFLICT}$$

Non-Monotonic Snapshot Isolation: scalable and strong consistency for geo-replicated transactional systems

Masoud Saeida Ardekani
Université Pierre-et-Marie Curie (UPMC-LIP6)
Paris, France
masoud.saeida-ardekani@lip6.fr

Pierre Sutra
University of Neuchâtel
Neuchâtel, Switzerland
pierre.sutra@unine.ch

Marc Shapiro
INRIA & UPMC-LIP6
Paris, France
http://lip6.fr/Marc.Shapiro/

SI/PSI 扩展性差, 作者提出四种可扩展性性质:

Wait-Free Queries: 只读事务不被阻塞且总是被成功提交

Genuine Partial Replication: 仅被读写的节点参与分布式事务协议

Minimal Commitment Synchronization: 仅在发生写冲突时等待

Forward Freshness: 允许读在当前事务开始后提交的数据

Non-Monotonic Snapshot Isolation: scalable and strong consistency for geo-replicated transactional systems

Masoud Saeida Ardekani
Université Pierre-et-Marie Curie (UPMC-LIP6)
Paris, France
masoud.saeida-ardekani@lip6.fr

Pierre Sutra
University of Neuchâtel
Neuchâtel, Switzerland
pierre.sutra@unine.ch

Marc Shapiro
INRIA & UPMC-LIP6
Paris, France
<http://lip6.fr/Marc.Shapiro/>

Definition (NMSI)

$$\text{NMSI} = \text{ACA} \wedge \text{CONS} \wedge \text{WCF}$$

Avoiding Cascading Aborts (ACA): 避免读未提交数据

Consistent Snapshot (CONS): 不遗漏所依赖事务的写操作

Write-Conflict Freedom (WCF): 没有 (并发) 写冲突

A Critique of Snapshot Isolation

Daniel Gómez Ferro Maysam Yabandeh *

Yahoo! Research
Barcelona, Spain


{danielgf,maysam}@yahoo-inc.com

WRITESI 避免读-写冲突, 而不是写-写冲突 (SI: READSI)

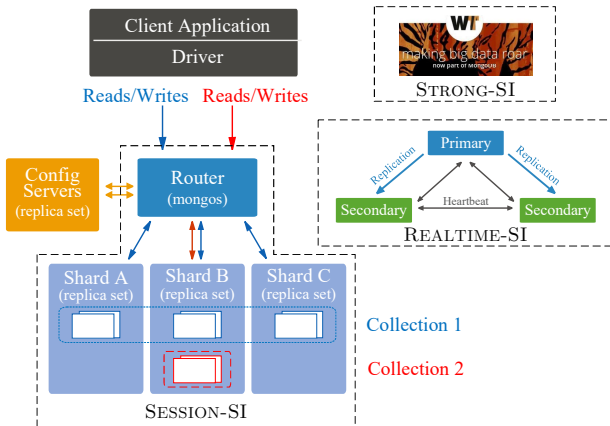
$$\text{WRITESI} \subseteq \text{SER} \subset \text{SI}$$

(暂时存疑)

PROTOCOL



proof



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), $txn.tid = -1$
- ▶ \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})

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

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})

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

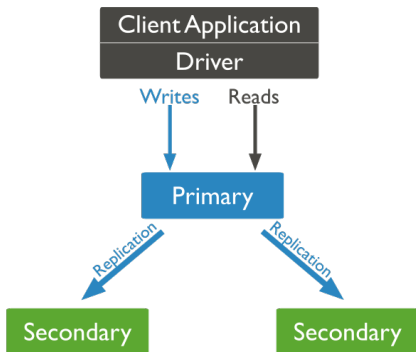
Lemma (冲突事务的提交顺序)

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

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

该引理对 STRONGSI 检测算法有辅助作用


$$\text{STRONGSI} = \text{SI} \wedge \text{REALTIME_SNAPSHOT} \wedge \text{RETURN_BEFORE} \\ \wedge \text{COMMIT_BEFORE}$$



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

$readConcern = \text{“snapshot”}$ 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

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

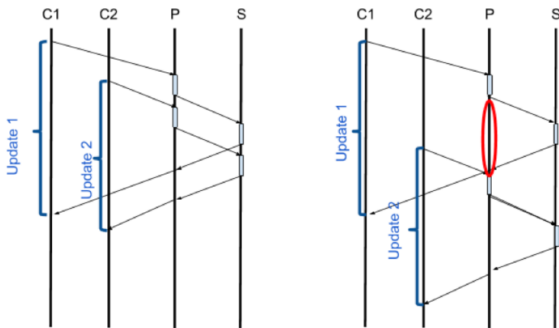


Figure 3: Back-to-Back Transactions with and without **Speculative Snapshot Isolation**

- ▶ 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

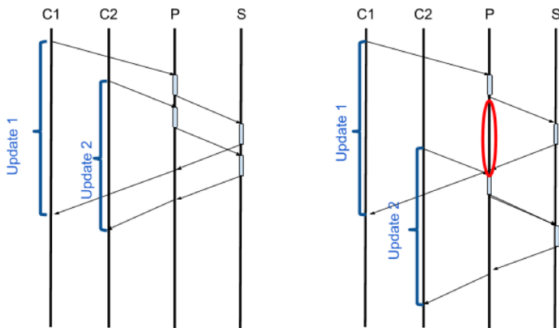


Figure 3: Back-to-Back Transactions with and without **Speculative Snapshot Isolation**

- ▶ 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

```
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)

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

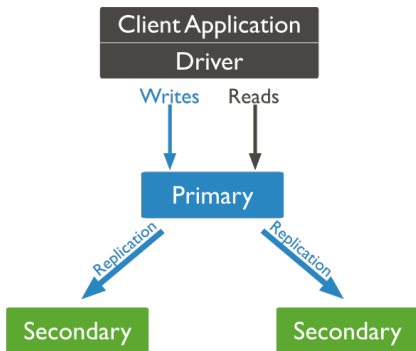
REPLICASET 使用逻辑时间戳“覆写” (override) 了
WIREDTIGER 中的事务标识号

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} 。

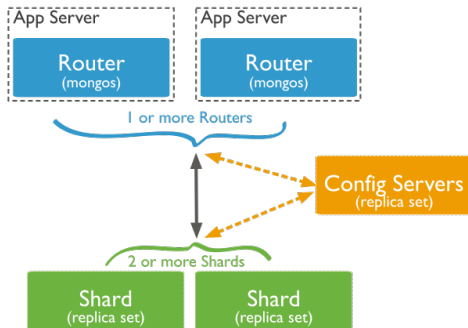
具体而言, 会话 $rs_session$ 上的只读事务 txn 紧随以下事务之后:

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



$$\text{REALTIME SI} = \text{SI} \wedge \text{RETURN BEFORE} \wedge \text{COMMIT BEFORE}$$

由主从架构、复制与多数节点提交机制、读时间戳设置策略共同保障

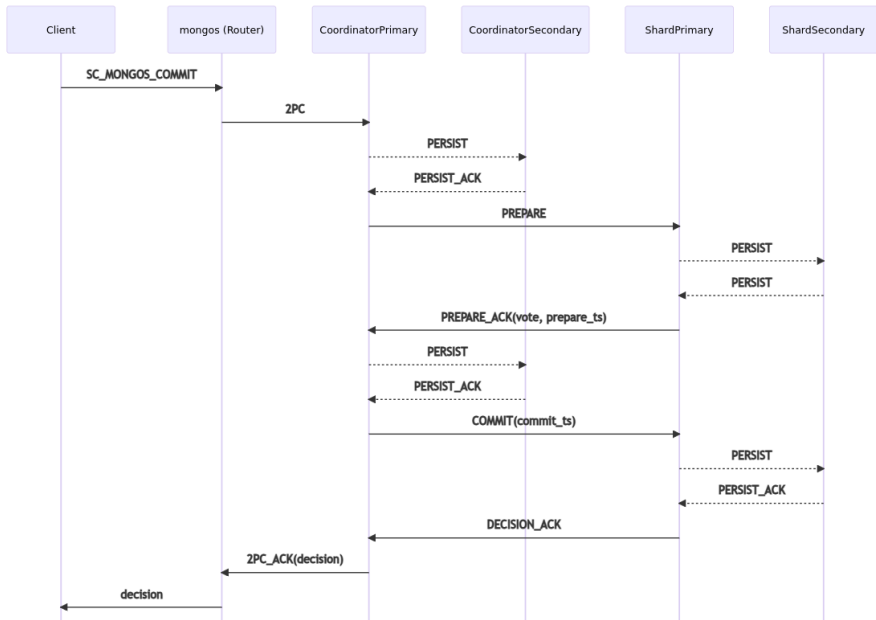


$\text{SHARDEDCLUSTER} \models \text{SESSIONSI}$

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

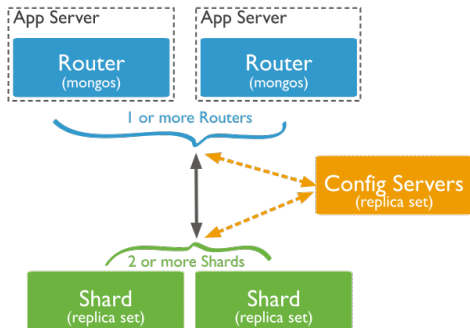
CASE-HOLES: 等待直到 oplog 中 $commit_ts \leq read_ts$ 的事务均提交

Definition (可见关系 VIS_{SC})

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

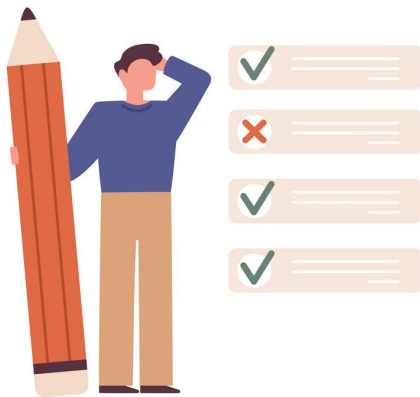
Definition (仲裁关系 AR_{SC})

- ▶ 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 $txn' \xrightarrow{VIS_{SC}} txn$ 。



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

由逻辑时间戳的维护与分发机制保障



多项式时间 SI 变体白盒检测算法



- (1) 收集事务协议产生的历史执行
- (2) 根据证明中的定义构造 VIS 与 AR 关系
- (3) 检查 VIS 与 AR 关系是否满足相应的 SI 变体所需的公理



- (1) 收集事务协议产生的历史执行
- (2) 根据证明中的定义构造 VIS 与 AR 关系
- (3) 检查 VIS 与 AR 关系是否满足相应的 SI 变体所需的公理

第 (2) 中构造的 AR 是全序关系, 提供了 Version Order,
第 (3) 步可以在多项式时间内完成

如何从历史执行中获取必要的信息, 以构造 VIS 与 AR 关系:

WIREDTIGER: 记录事务的开始与提交时间, 利用前述引理校正

REPLICASET: 分别从 `mongod.log` 与 `oplog.rs` 中获取
事务的读时间戳与提交时间戳

SHARDEDCLUSTER: 分别从 `mongod.log` 与分区主节点 `oplog` 中获取
事务的读时间戳与提交时间戳

Lemma (冲突事务的提交顺序)

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

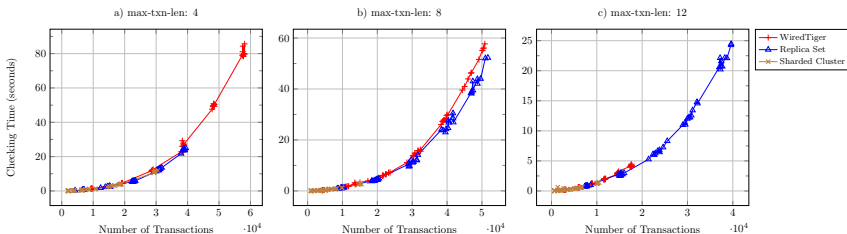
$txn \bowtie txn' \implies (txn \xrightarrow{AR_{WT}} txn' \iff txn.tid < txn'.tid).$

Deployment	Version	Configuration	OS
Standalone	WiredTiger 3.3.0	A WiredTiger storage engine	Ubuntu 20.04
Replica Set	MongoDB 4.2.8	A replica set of 5 nodes	Each database node runs in a Docker container built with the Debian 10 image.
Sharded Cluster		<p>A cluster consists of 1 config server and 2 shards.</p> <p>Each shard is a replica set of 3 nodes.</p> <p>Each node has a mongos instance.</p>	

MongoDB 部署配置

Parameter	Value(s)	Description
key-count	10	There are 10 distinct keys at any point for generation.
key-dist	exponential	Probability distribution on keys.
max-txn-length	{4, 8, 12}	Each transaction contains at most 4/8/12 operations.
max-updates-per-key	128	There are at most 128 updates on each key.
read:update ratio	1 : 1	The default (and fixed) read:update ration in Jepsen.
wt-duration	{10s, 20s, 30s, 40s, 50s, 60s}	The duration of an experiment on WiredTiger.
rs-duration	{1min, 2min, 3min, 4min, 5min}	The duration of an experiment on replica set .
sc-duration	{1min, 2min, 3min, 4min, 5min, 8min}	The duration of an experiment on sharded cluster.
wt/rs/sc-timeout	5s/10s/30s	Transaction timeout for experiments on WiredTiger, replica set, and sharded cluster, respectively.

事务生成器相关参数 (Jepsen 支持)



- ▶ 13ms 的时钟误差内, 可认为 $\text{WIREDTIGER} \models \text{STRONGSI}$
- ▶ 10s 检测包含 30,000 个事务的历史执行
 - ▶ Intel Core i5-9500 CPU @ 3.00GHZ and 16GB RAM

Conclusion



Hengfeng Wei (hfwei@nju.edu.cn)