

# Verifying Transactional Consistency of MongoDB

(submitted to VLDB'2022)

Hengfeng Wei

hfwei@nju.edu.cn

December 27, 2021





## 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

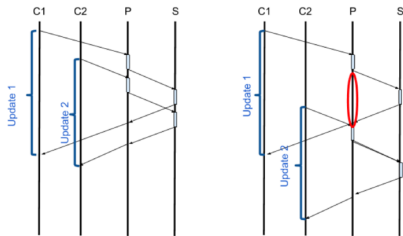
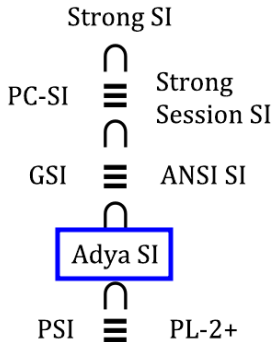


Figure 3: Back-to-Back Transactions with and without **Speculative Snapshot Isolation**



## 挑战二: MongoDB 缺少精简的事务协议描述, 更没有严格证明

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
-------------	-----------------------------------

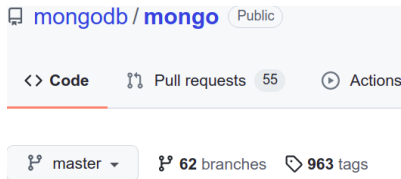
挑战三: SI 检测问题是 NP-complete 问题, 复杂度高

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.

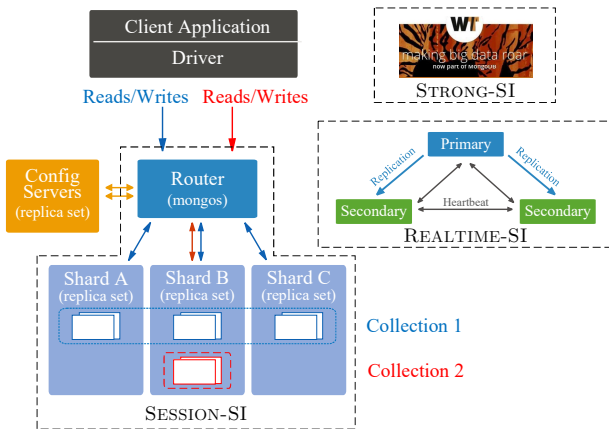


贡献一: 使用 (VIS, AR) 框架, 为多种 SI 变体提供形式化规约

## 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 $T : (E, \text{po})$

- ▶  $\text{po}$  : Program Order
- ▶  $\text{start}(T)$  : 事务开始时间
- ▶  $\text{commit}(T)$  : 事务提交时间

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

- ▶  $\mathbb{T}$  : 已提交事务集合
- ▶  $\text{so}$  : Session Order

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

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

一个事务一致性模型可定义为一组一致性公理的集合  $\Phi$ 。

历史  $\mathcal{H}$  满足事务一致性模型  $\Phi$ , 如果存在 VIS 与 AR 使得

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

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

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



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

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

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

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

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

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

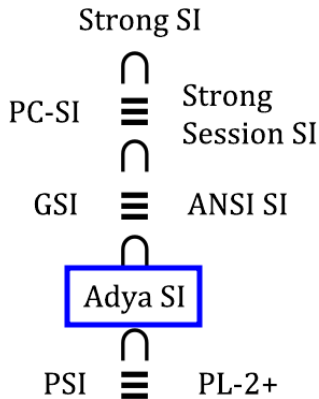
$$\text{SESSIONSI} = \text{SI} \wedge \text{SESSION}$$

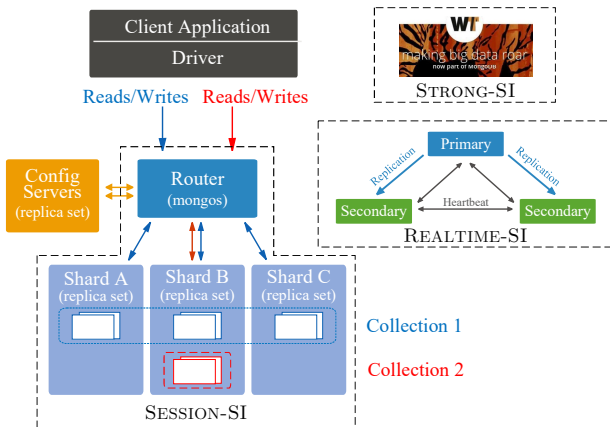
$$\text{REALTIMESI} = \text{SI} \wedge \text{RETURNBEFORE} \wedge \text{COMMITBEFORE}$$

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

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

- ▶ 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),  $txn.tid = -1$
- ▶  $\perp_{\text{tid}}$  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),  $txn.tid = -1$
- ▶  $\perp_{\text{tid}}$  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 (Visibility Relation)

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

在逻辑上, 所有事务按实时序依次提交

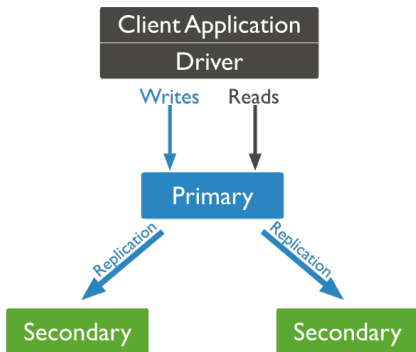
### Definition (Arbitration Relation)

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

Lemma (冲突事务的提交顺序)

$\forall txn, txn' \in \text{WT\_TXN}.$

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



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





# Conclusion



Hengfeng Wei (hfwei@nju.edu.cn)