Verifying Transactional Consistency of MongoDB

(submitted to VLDB'2022)

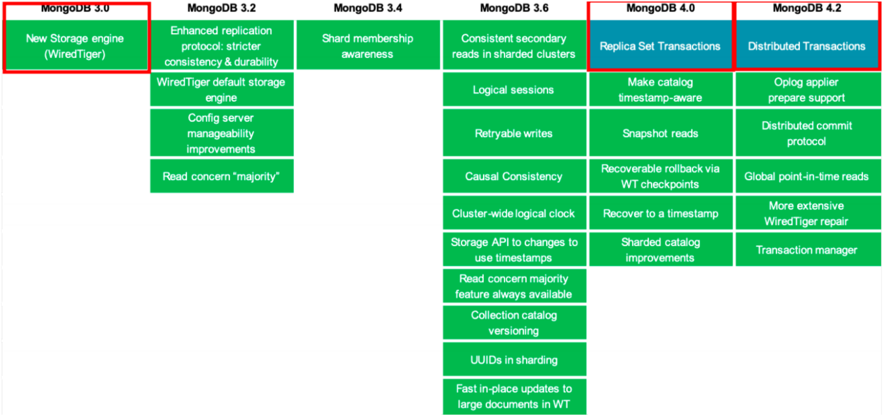
Hengfeng Wei

hfwei@nju.edu.cn

December 27, 2021

mongoDB

MongoDB 的三种经典部署架构



MongoDB 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

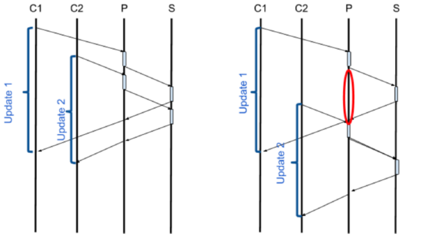
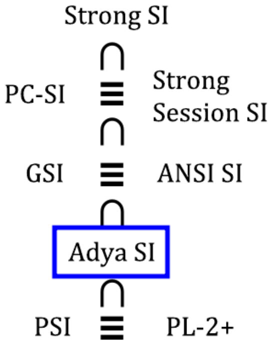
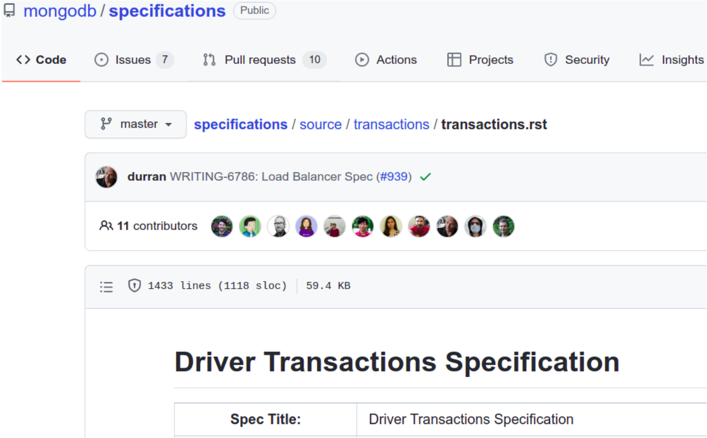


Figure 3: Back-to-Back Transactions with and without Speculative Snapshot Isolation



挑战二：MongoDB 缺少精简的事务协议描述，更没有严格证明

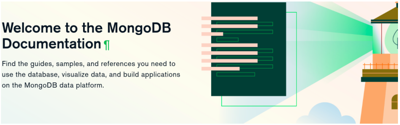


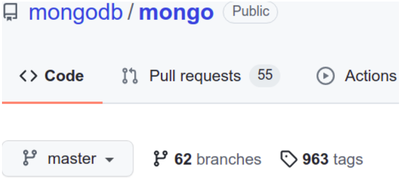
挑战三: SI 检测问题是 NP-complete 问题, 复杂度高

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.*

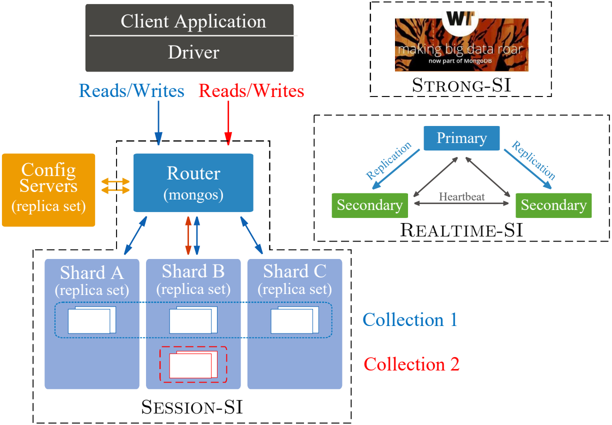
贡献一: 使用 (VIS, AR) 框架, 为多种 SI 变体提供形式化规约

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)

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

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

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

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

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

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

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

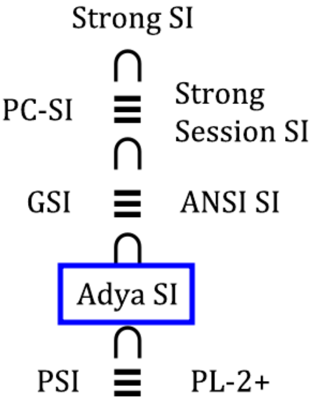
$$\textsc{SessionSI} = \mathrm{SI} \wedge \textsc{Session}$$

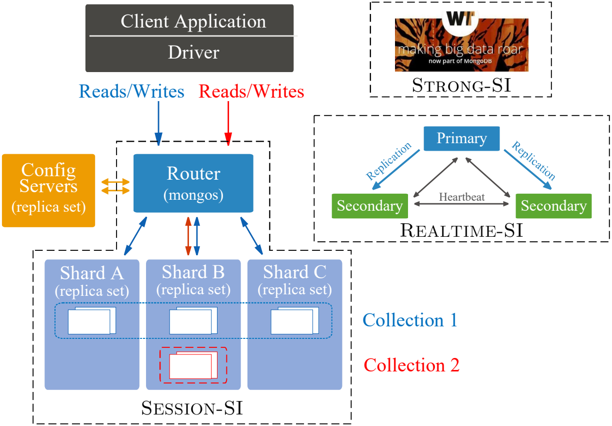
$$\textsc{RealtimeSI} = \mathrm{SI} \wedge \textsc{ReturnBefore} \wedge \textsc{CommitBefore}$$

$$\mathrm{GSI} = \mathrm{SI} \wedge \textsc{RealtimeSI} \wedge \textsc{CommitBefore}$$

$$\textsc{StrongSI} = \mathrm{GSI} \wedge \textsc{ReturnBefore}$$

- 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

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

- 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), $txn.\mathsf{tid} = -1$

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

- 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 $wt\_sid \in \mathsf{WT\_SID} = \mathbb{N}$
- 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

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

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

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

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

1: **procedure** TXN_VISIBLE(txn, tid)

2: **return** $\neg\big(tid = -1 \vee tid \in txn.\mathsf{snapshot} \vee (tid \geq txn.\mathsf{snap\_max} \wedge tid \neq txn.\mathsf{tid})\big)$

Conclusion

Hengfeng Wei (hfwei@nju.edu.cn)