DTCC

# 第十一届中国数据库技术大会

DATABASE TECHNOLOGY CONFERENCE CHINA 2020

## 架构革新 高效可控

IT168.com ChinaUnix.net ITPUB

北京国际会议中心 | 2020/12/21-12/23

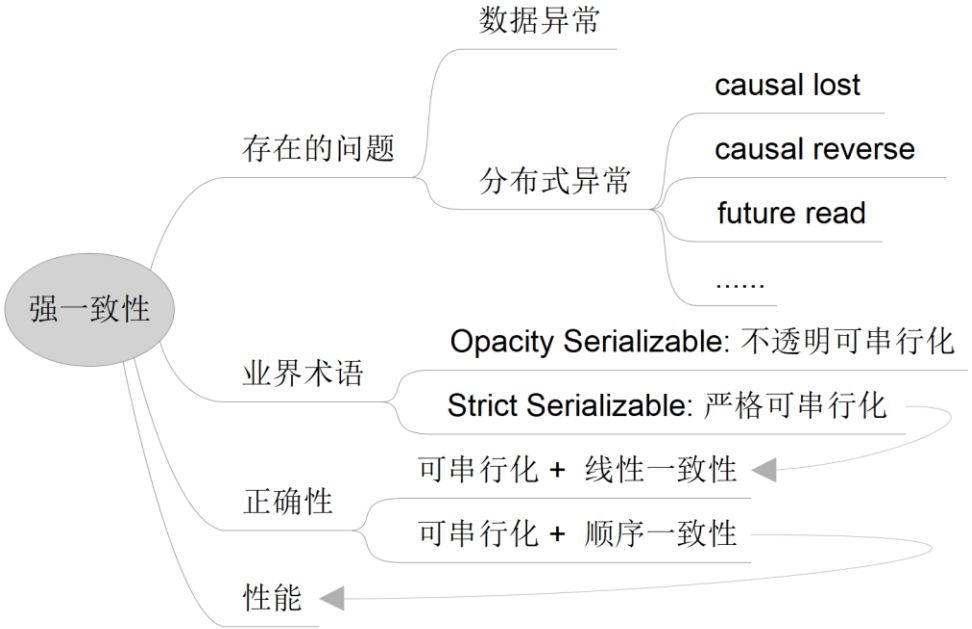
# 腾讯TDSQL 分布式事务处理

## 多级一致性技术（强一致性）

李海翔 @那海蓝蓝

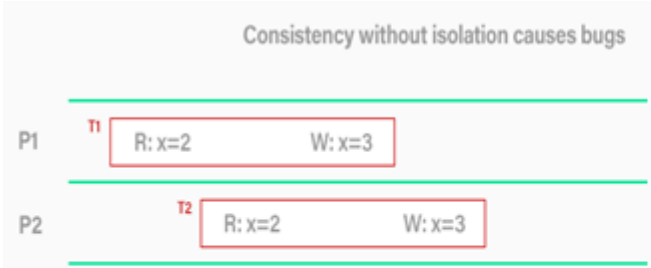
# “强一致”的 分布式事务型数据库



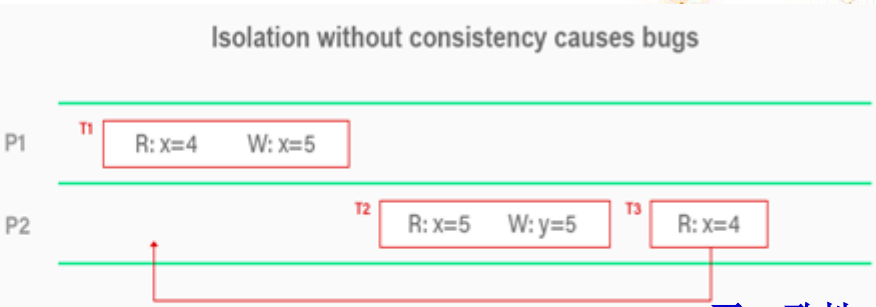


# @分布式一致性+事务一致性的问题—挑战：缺少一致性带来的问题






无事务保障：脏写数据异常




无一致性保障：违反因果序

分布式事务型数据库，
两种一致性缺一不可

# @分布式一致性+事务一致性的问题— PostgreSQL 中一个神奇的例子



**数据准备**

```
create table control ( deposit_no int not null ); insert into control values (1);
create table receipt ( receipt_no serial primary key, deposit_no int not null, payee text not null, amount money not null );
insert into receipt (deposit_no, payee, amount) values ((select deposit_no from control), 'Crosby', '100′);
insert into receipt (deposit_no, payee, amount) values ((select deposit_no from control), 'Stills', '200′);
insert into receipt (deposit_no, payee, amount) values ((select deposit_no from control), 'Nash', '300′);
```

理论分析时的并发情况：

| T1 | T2 | T3 |
|---|---|---|
| R1-x0 | | |
| W1-y1 | | |
| | R2-x0 | |
| | W2-x1 | |
| | C2 | |
| | | R3-y0 |
| C1 ？ | | |

事实并发情况：

| T1 | T2 /T3 |
|---|---|
| R1-x0 | |
| W1-y1 | |
| | R2-x0 |
| | W2-x1 |
| | C2 |
| | |
| | R3-y0 |
| C1 ？ | |

# @分布式一致性+事务一致性的问题— PostgreSQL 中一个神奇的例子



| P1 | P2 | |
|---|---|---|
| T1 | T2 | T3 |
| begin; -- T1<br>insert into receipt (deposit_no, payee, amount) values ( (select deposit_no from control), 'Young', '100' ); | | |
| select * from receipt;<br>receipt_no \| deposit_no \| payee \| amount<br>----------+----------+--------+--------<br>1 \| 1 \| Crosby \| $100.00<br>2 \| 1 \| Stills \| $200.00<br>3 \| 1 \| Nash \| $300.00<br>4 \| 1 \| Young \| $100.00<br>(4 rows) | begin; -- T2<br>select deposit_no from control;<br>deposit_no<br>----------<br>1 | |
| | update control set deposit_no = 2;<br>commit; | |
| | | begin; -- T3<br>select * from receipt where deposit_no = 1;<br>receipt_no \| deposit_no \| payee \| amount<br>----------+----------+--------+--------<br>1 \| 1 \| Crosby \| $100.00<br>2 \| 1 \| Stills \| $200.00<br>3 \| 1 \| Nash \| $300.00<br>(3 rows) |
| commit;<br>ERROR: could not serialize access<br>due to read/write dependencies among transactions<br>DETAIL: Cancelled on identification as a pivot, during commit attempt.<br>HINT: The transaction might succeed if retried. | | |

https://wiki.postgresql.org/wiki/SSI#Deposit_Report

# @分布式一致性+事务一致性的问题— PostgreSQL 中一个神奇的例子



**可串行化**：T3-T1-T2
**依赖图**：无环
**SSI**：无环
**Pg SSI**：无环，但有“RW→ T1 →RW”

**可串行化**：T3-T1-T2 <------> **因果序**：T2-T3
结论：
单机系统的事务排序，需要考虑因果序(可串行化的缺陷)
**DTA：T3的lower >= 符合读已提交的数据的提交时间戳**
**多级一致性**：T3-T1-T2-T3，故有环

**Pg事实执行逻辑：**

```
T3 -> T1   因为T1没有提交，T3根据MVCC读旧版本
R1(control)[deposit_no=1]
R3(receipt) -> W1(receipt)[deposit_no=1]

T1 -> T2
R1(control) -> R2(control)  W2(control)

T2->T3(session order)  没有判断
```

T3 →RW→ T1 →RW→ T2
发生了Stale read，在事务型DB中，是否合理？

**按事务原理推理的执行逻辑：**

```
T1 -> T3   按real-time排定逻辑关系
R1(control)[deposit_no=1]
W1(receipt)[deposit_no=1] -> R3(receipt)

T1 -> T2
R1(control) -> R2(control)  W2(control)

T2->T3(session order)  没有判断
```

无环

**按事务+分布式一致性原理推理的执行逻辑：**

```
T1 -> T3   按real-time排定逻辑关系
R1(control)[deposit_no=1]
W1(receipt)[deposit_no=1] -> R3(receipt)

T1 -> T2
R1(control) -> R2(control)  W2(control)

T2->T3(session order)  应该判断
```

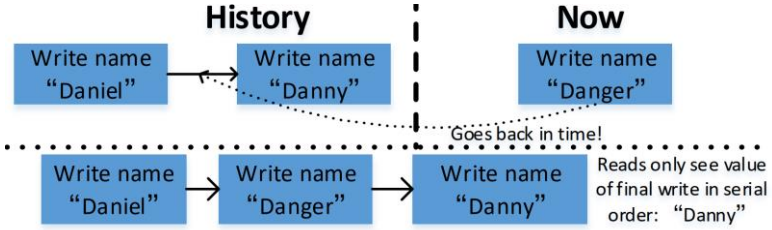
有环

事务：可串行化，需要考虑因果一致性

https://wiki.postgresql.org/wiki/SSI#Deposit_Report

# @分布式一致性+事务一致性的问题—业界观点：（异步）复制数据库



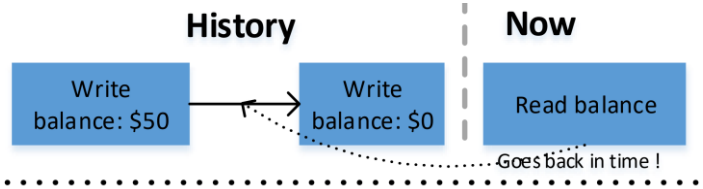



## Immortal write

In multi-master asynchronously replicated database systems, where writes are allowed to occur at either replica, it is possible for conflicting writes to occur across the replicas.

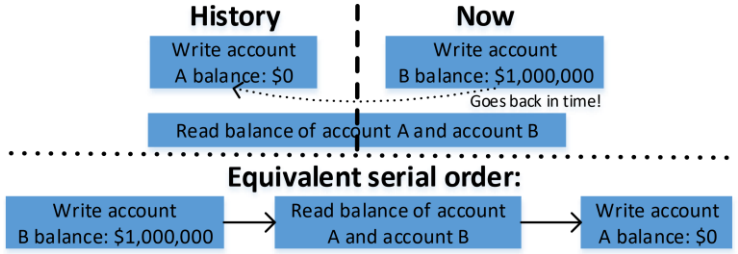
| | T1 | T2 | T3 | T4 |
|---|---|---|---|---|
| t0 | | | | |
| t1 | W-Rep1(t1) | | | |
| t2 | | W-Rep1(t2) | | |
| t3 | | | W-Rep3(t1) | |
| t4 | | | | R-Rep1(t1) |

复制型架构中的问题




## Causal reverse

| | T1 | T2 | T3 | |
|---|---|---|---|---|
| t0 | | | | |
| t2 | W-Rep1(t2) | | | |
| t3 | | W-Rep2(t3) | | (T1->T2) |
| t4 | | | R-Rep1(t0) R-Rep2(t3) | T2\T3 in the same Rep |




http://dbmsmusings.blogspot.com/2019/08/an-explanation-of-difference-between.html

## Stale reads

Stale reads do not violate serializability. The system is simply time travelling the read transaction to a point in time in the equivalent serial order of transactions before the new writes to this data item occur. Therefore, asynchronously replicated database systems can allow stale reads without giving up its serializability (or even one-copy serializability) guarantee.

| | T1 | T2 |
|---|---|---|
| t0 | | |
| t2 | W-Rep1(t2) | |
| t3 | | R-Rep2(t1) |

# @分布式一致性+事务一致性的问题—分布式一致性异常总结



## 总结：

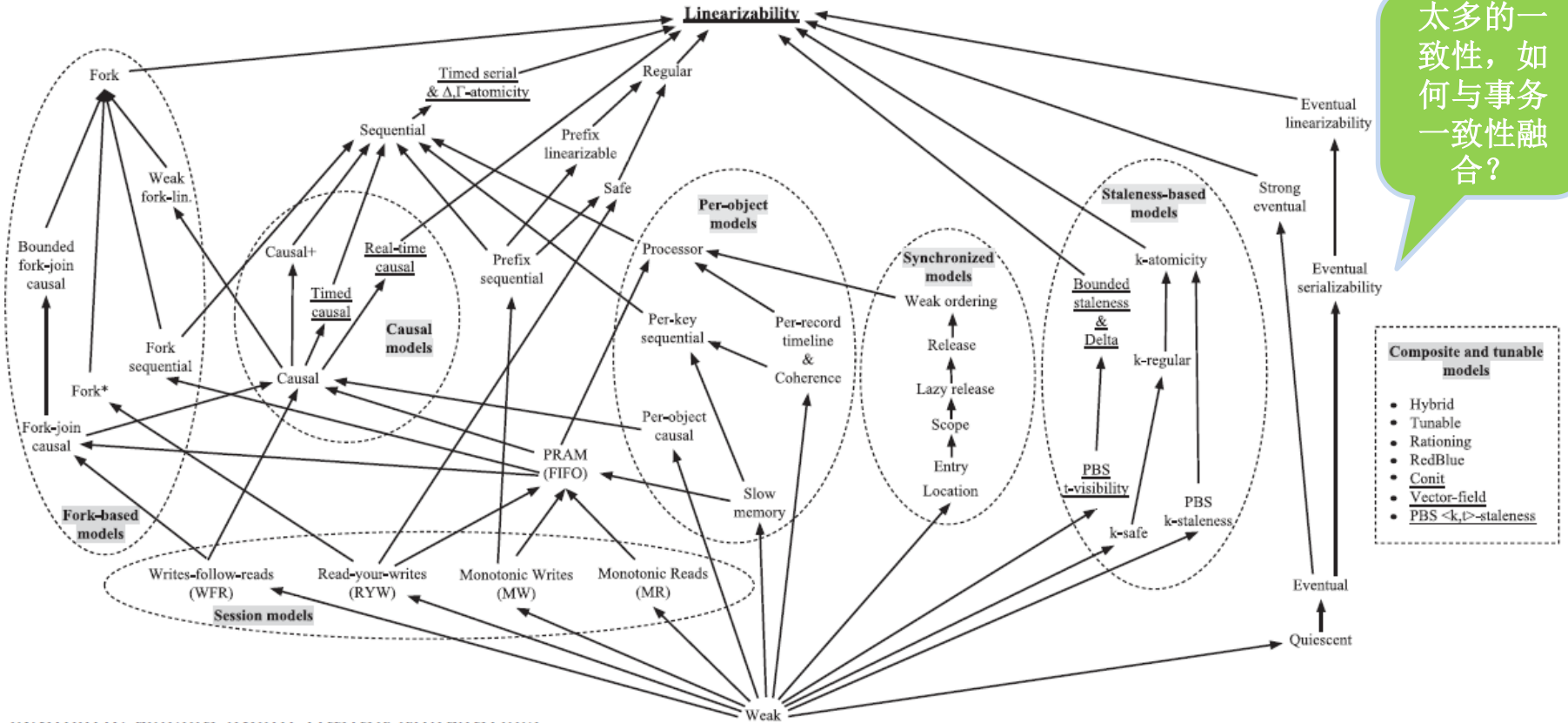
1. **并发事务之间，属于事务/数据的一致性问题**
2. **非并发事务之间，存在如下问题**（借用了Daniel的词，但含义可能不同）

| 原因/异常 | 示例号 | 场景/现象 | 特点 |
|---|---|---|---|
| **causal lost（绑匪问题）** | S1 | P1 W1-x-t1<br>P2 R2-x-t0<br>不同session之间不存在显式的因果关系 | **因果关系是隐式无法知道的** |
| **causal reverse**（Stale read（不能读到最新的数据，读到一个旧数据） | S2 | 不同session之间存在显式的因果关系<br>P1 W1-x-t1<br>P2 R2-x-t1 R3-x-t0（**时间关系不确定导致因果反转**）<br>或：<br>P1 W1-x-t1 W1-y-t3<br>P2 R3-x-t0（**读已提交导致因果反转**） | **因果关系是显式可知道的** |
| | S4 | P1 W1-x-t1 R1-x-t0<br>相同session之间存在显式的因果关系 | |
| **未来读<br>读漂移** | **S5** | **P1 W1-x1-t1 C1<br>P2 R2-x1-t0 C2（读后于自己在real-time上发生的数据，无因果关系）** | 时钟漂移到未来，无因果关系 |

# @分布式一致性的问题—挑战：问题复杂度增加



太多的一致性，如何与事务一致性融合？



# @分布式一致性+事务一致性的问题—业界观点



Table 1: *Ordering guarantees.* We use result visibility to define a range of coherence and consistency models. Checkmarks indicate constraints that are enforced in the model. We assume that programs, sessions and transactions are equivalent. Note that, because result visibility is an invariant, the constraints are additive and there is no need to mention executions. As a result, we can compactly represent a wide range of ordering guarantees using only result visibility to construct the columns of this table.

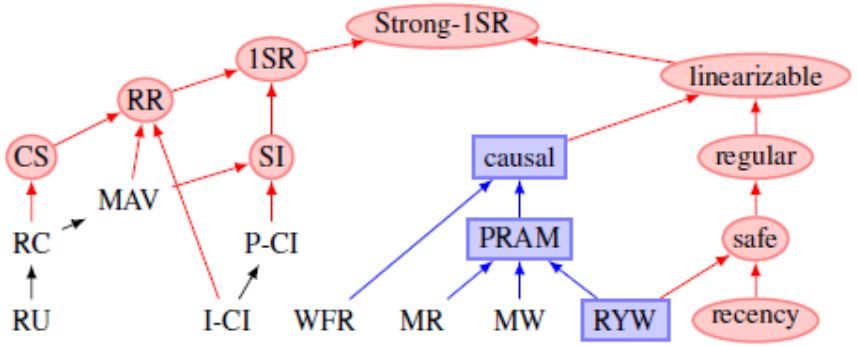
| | Coherence (same object) | | | | | | Consistency (different objects) | | | | | | Session Guarantees | | Isolation (transactional) | | | | | | | Real-time |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Program ordering $\alpha(x) \prec_p \beta(x) \Longrightarrow \alpha(x) \lhd \beta(x)$ | | | | Write Atomicity $w_1(x) \prec w_2(x) \lhd \gamma \Longrightarrow w_1(x) \lhd \gamma$ | | Program ordering $\alpha(x) \prec_p \beta(y) \Longrightarrow \alpha(x) \lhd \beta(y)$ | | | | Write Atomicity $w_1(x) \prec w_2(y) \lhd \gamma \Longrightarrow w_1(x) \lhd \gamma$ | | Monotonic Views $w \lhd r_1 \prec_p r_2 \Longrightarrow w \lhd r_2$ | Dependent Writes $w_1 \lhd r \prec_p w_2 \lhd \gamma \Longrightarrow w_1 \lhd \gamma$ | Dependency cycle-free $\alpha \in t_1 \lhd \beta \in t_2 \Longrightarrow S(t_1) \lhd S(t_2)$ | | | Consistent Reads $w \in t_1 \lhd r \in t_2 \Longrightarrow WS(t_1) \lhd RS(t_2)$ | Repeatable Reads $w \in t_1 \lhd r \in t_2 \Longrightarrow w \lhd RS(t_2)$ | Non-inter. Reads $w_1 \prec_p w_2 \in t_1$ $w_1 \lhd r \in t_2 \Longrightarrow w_2 \lhd r \in t_2$ | No Aborted Reads $WS(t_1) \lhd RS(t_2)$ $t_2 \in T_C \Longrightarrow t_1 \in T_C$ | $\alpha \prec_r \beta \Longrightarrow \alpha \lhd \beta$ |
| | $w \prec r$ | $w \prec w$ | $r \prec r$ | $r \prec w$ | $\prec_p$ | $\lhd$ | $w \prec r$ | $w \prec w$ | $r \prec r$ | $r \prec w$ | $\prec_p$ | $\lhd$ | | | $WS \lhd WS$ | $RS \lhd WS$ | $WS \lhd RS$ | | | | | |
| **Strict Serializability** | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| **Serializability [7]** | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | |
| **Snapshot Isolation [6]** | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓[1] | ✓ | ✓ | ✓ | ✓ | |
| **Parallel SI [20]** | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓[1] | ✓ | ✓ | ✓ | ✓ | |
| **Non-monotonic SI[4]** | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | | ✓ | ✓ | ✓ | ✓[1] | ✓ | ✓ | ✓ | ✓ | |
| **Monotonic Atomic Views [5]** | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | | ✓ | ✓ | ✓ | ✓ | |
| **Repeatable Reads** | ✓ | ✓ | ✓ | ✓ | | | ✓ | ✓ | ✓ | ✓ | | | | | ✓ | ✓ | | | ✓ | ✓ | ✓ | |
| **Read Committed [14]** | ✓ | ✓ | ✓ | ✓ | | | ✓ | ✓ | ✓ | ✓ | | | | | ✓ | ✓ | | | | ✓ | ✓ | |
| **Read Uncommitted [14]** | ✓ | ✓ | ✓ | ✓ | | | ✓ | ✓ | ✓ | ✓ | | | | | ✓ | | | | | | | |
| **Linearlizability [15]** | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | | | | | | | | ✓ |
| **Sequential Consistency [16]** | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | | | | | | | | |
| **Total Store Order [1]** | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | | | | | | | | |
| **Partial Store Order [1]** | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | | | ✓ | ✓ | | ✓ | ✓ | ✓ | | | | | | | | |
| **Weak Ordering [12]** | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | | | | | | ✓ | ✓ | ✓ | | | | | | | | |
| **Processor Consistency [13]** | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | | ✓ | ✓ | ✓ | ✓ | | ✓ | ✓ | | | | | | | | |
| **PRAM [17]** | ✓ | ✓ | ✓ | ✓ | ✓ | | | ✓ | ✓ | ✓ | ✓ | | ✓ | ✓ | | | | | | | | |
| **Cache Consistency [13]** | | | | | | ✓ | | | | | | | | | | | | | | | | |
| **Read-Your-Writes [21]** | ✓ | | | | | | ✓ | | | | | | | | | | | | | | | |
| **Monotonic Reads [21]** | | | ✓ | | | | | | ✓ | | | | ✓ | | | | | | | | | |
| **Monotonic Writes [21]** | | ✓ | | | ✓ | | | ✓ | | | ✓ | | | | | | | | | | | |
| **Writes-Follow-Reads [21]** | | | | ✓ | | | | | | ✓ | | | | ✓ | | | | | | | | |

*Making Consistency More Consistent: A Unified Model for Coherence, Consistency and Isolation*

# @问题与背景—挑战：业界研究可改进不？



- 两个体系：
  - 分布式一致性
  - 事务一致性
- 一个世界：
  - 分布式事务型数据库




Bailis P, Davidson A, Fekete A, et al. Highly available transactions: Virtues and limitations. In Proc. of VLDB. 2013. 181-192.

| 产品名称 | 可串行化 | 多级一致 |
| --- | --- | --- |
| Oxxxxxxxx | 2.2版本 支持 | 因果一致性 |
| CorckroachDB | 支持 | 顺序一致性、因果一致性 |
| TxXx | 不支持 | 因果一致性 |
| Spanner | 支持（性能差） | 严格可串行化 |
| **TDSQL** | **支持** | **严格可串行化（正确性）**<br>**.....**<br>**顺序可串行化（性能）** |

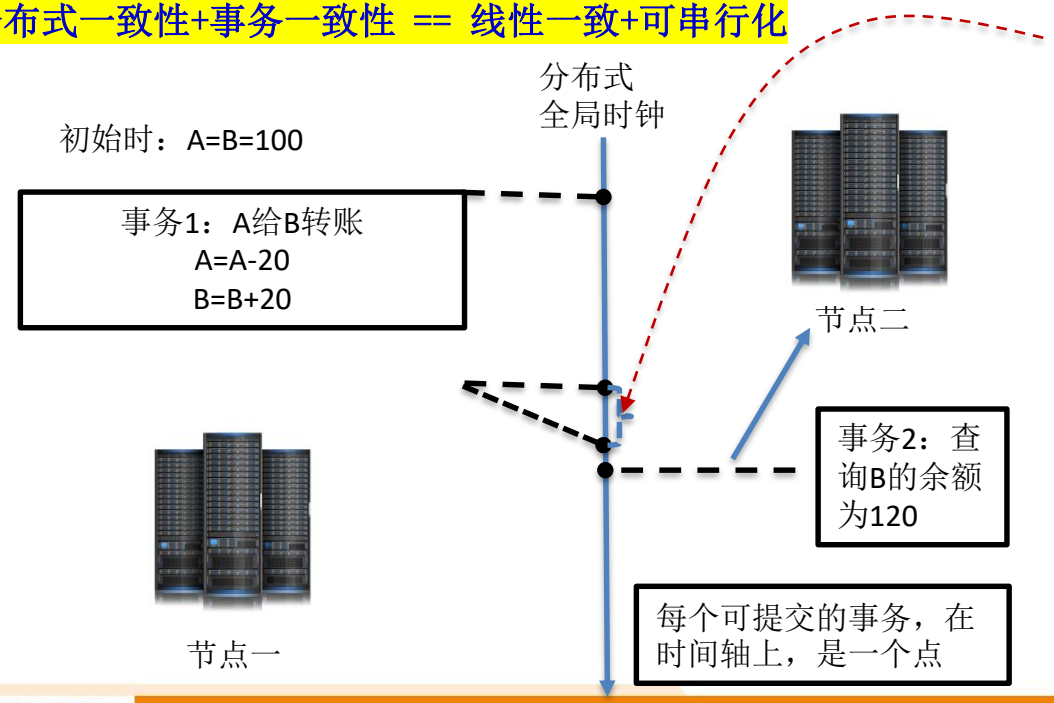
# @分布式一致性+事务一致性的问题—业界典型方案



**Spanner，严格可串行化，存在性能瓶颈**

原子钟+TrueTime延迟提交（Google Spanner）

分布式一致性+事务一致性 == 线性一致+可串行化




延迟提交时间（TrueTime）
>
节点时钟差异的上限（原子钟）

从 time master 同步时的网络延迟, 导致的误差≈ 1ms

local clock 的漂移, 误差是时间的锯齿函数， 需校准

**问题：**
单分片写事务最大发生数：
1s/2∈ = 125个事务/每秒
∈是延迟提交平均值，约为4ms

# @分布式一致性+事务一致性的问题—业界观点



## 分布式一致性+事务一致性

1. One-copy serializability guarantees sequential consistency where no thread of execution can process more than one transaction.
2. Strong session serializability guarantees sequential consistency where each session corresponds to a separate thread of execution.
3. Strong write serializability corresponds to a linearizability guarantee for write transactions, but only sequential consistency for read-only transactions.
4. Strong partition serializability guarantees linearizable consistency within a partition, but only sequential consistency across partitions.
5. And strict serializability guarantees linearizable consistency at all times for all reads and writes (across non-concurrent transactions).

http://dbmsmusings.blogspot.com/2019/08/an-explanation-of-difference-between.html

## 强一致性/弱一致性

In a hybrid consistent system, *strong* operations are guaranteed to be seen in some sequential order by all processes (as in sequential consistency), while *weak* operations are designed to be fast, and they eventually become visible by all processes (much like in eventual consistency). Weak operations are only guaranteed to be ordered according to their interleaving with strong operations: if two operations belong to the same session and one of them is strong, then their relative order of invocation is respected and visible by all processes.

Similarly to hybrid consistency (see Section 3.8), strong operations are ordered according to processor or sequential consistency, whereas the ordering of weak operations is just restricted by the relative ordering with respect to the strong operations invoked by the same process.

Viotti P, Vukolić M. Consistency in Non-Transactional Distributed Storage Systems. ACM Comput. Surv., 2016, 49(1): 19:1-19:34.

# @学术界的方案—各种一致性的差异：线性一致性，定义



- 线性一致性：任何2个写操作，串行执行；读、写可并行；写操作的输出值总是全序

$$\textsc{Linearizability}(\mathcal{F}) \triangleq \textsc{SingleOrder} \wedge \textsc{RealTime} \wedge \textsc{RVal}(\mathcal{F}), \tag{7}$$

where

$$\textsc{SingleOrder} \triangleq \exists H' \subseteq \{op \in H : op.oval = \nabla\} : vis = ar \setminus (H' \times H) \tag{8}$$

and

$$\textsc{RealTime} \triangleq rb \subseteq ar. \tag{9}$$

- Regular线性一致性：任何2个写操作，串行执行；读、写可并行；但获取到的值可能是最近并发写操作影响到的的一个值
- Safe线性一致性：任何2个写操作，串行执行；读、写可并行；但获取到的值可能是副本中的任何一个值（某个副本可能没有收到最近的写操作的影响而改变值）

$$\textsc{Regular}(\mathcal{F}) \triangleq \textsc{SingleOrder} \wedge \textsc{RealTimeWrites} \wedge \textsc{RVal}(\mathcal{F}) \tag{10}$$

$$\textsc{Safe}(\mathcal{F}) \triangleq \textsc{SingleOrder} \wedge \textsc{RealTimeWrites} \wedge \textsc{SeqRVal}(\mathcal{F}), \tag{11}$$

where

$$\textsc{RealTimeWrites} \triangleq rb|_{wr \to op} \subseteq ar \tag{12}$$

is a restriction of real-time ordering only for writes (preceding reads or other writes), and

$$\textsc{SeqRVal}(\mathcal{F}) \triangleq \forall op \in H : Concur(op) = \emptyset \Rightarrow op.oval \in \mathcal{F}(op, cxt(A, op)), \tag{13}$$

which restricts the return value consistency only to read operations that are not concurrent with any write.

Viotti P, Vukolić M. Consistency in Non-Transactional Distributed Storage Systems. ACM Comput. Surv., 2016, 49(1): 19:1-19:34

理论上：

线性一致可以细分为多种级别

The difference between the three resides in the allowed set of return values for a read operation concurrent with a write

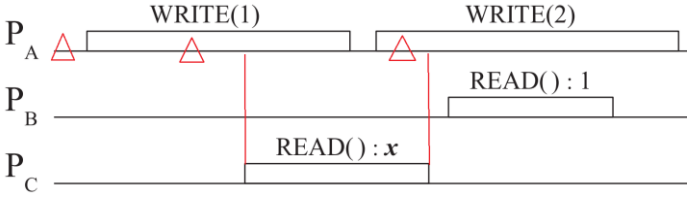



Fig. An execution exhibiting read-write concurrency (real-time flows from left to right). The register is initialized to 0. **Atomic (linearizable) semantics allow *x* to be 0 or 1**. Regular semantics allow *x* to be 0, 1, or 2. With safe semantics *x* may be any value

# @学术界的方案—采用的技术：顺序一致性



- **顺序一致性：对session内的写操作排序；不同session间的写操作，不用全局有序；任何读，都按照session排定的序读取（读操作读取的结果对于一个session写出的结果是稳定的）**

$$\text{PRAM} \triangleq so \subset vis. \tag{20}$$

$$\text{SequentialConsistency}(\mathcal{F}) \triangleq \text{SingleOrder} \wedge \text{PRAM} \wedge \text{RVal}(\mathcal{F}). \tag{21}$$

$so$ ($session\ order$) is a partial order defined as $so \triangleq rb \cap ss$.

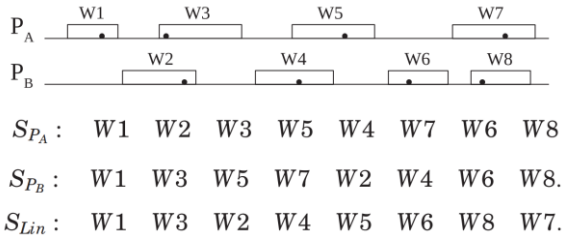
Pipeline RAM (**PRAM** or FIFO) consistency [Lipton and Sandberg 1988] prescribes that all processes see write operations issued by a given process in the same order as they were invoked by that process

In a storage system implementing **sequential** consistency, all operations are serialized in the same order on all replicas, and the ordering of operations determined by each process is preserved.

Viotti P, Vukolić M. Consistency in Non-Transactional Distributed Storage Systems. ACM Comput. Surv., 2016, 49(1): 19:1-19:34

In **sequential consistency**, all writes --- no matter which thread made the write, and no matter which data item was written to --- are **globally ordered**. Every single thread of execution must see the writes occurring in this order.

https://dbmsmusings.blogspot.com/2019/07/overview-of-consistency-levels-in.html
http://dbmsmusings.blogspot.com/2019/08/an-explanation-of-difference-between.html




An execution with processes issuing write operations on a shared object. Black spots are the chosen linearization points.

**前2个，顺序一致；第三个，线性一致性**

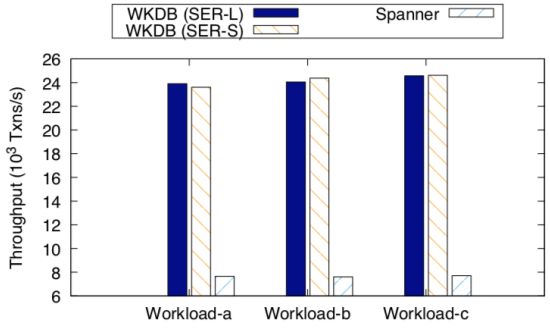
## 原始定义

A high-speed processor may execute operations in a different order than is specified by the program. The correctness of the execution is guaranteed if the processor satisfies the following condition: the result of an execution is the same as if the operations had been executed in the order specified by the program. A processor satisfying this condition will be called *sequential*. Con-

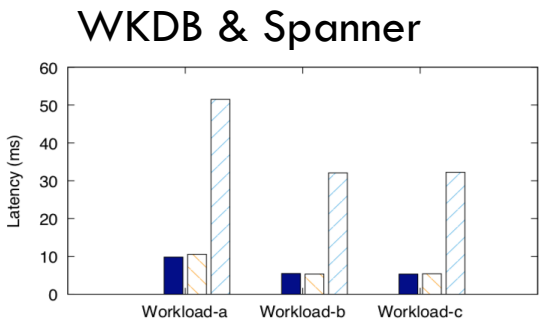
Leslie Lamport. 1979. How to make a multiprocessor computer that correctly executes multiprocess programs. IEEE Transactions on Computers 28, 9 (1979), 690–691.

此处的“**globally ordered**”非对所有的源自不同session的写操作一同排序，而是对他们产生的结果使之在全局保持确定的顺序且符合so序

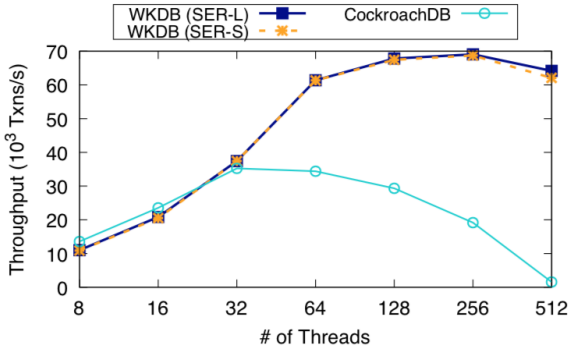
在YCSB基准测试集上，WKDB显著优于谷歌的Spanner及其开源实现CockroachDB。



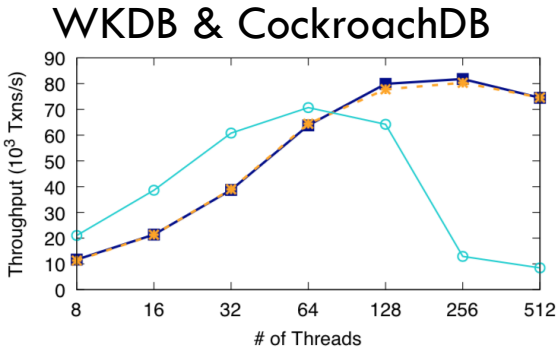

(a) Throughput



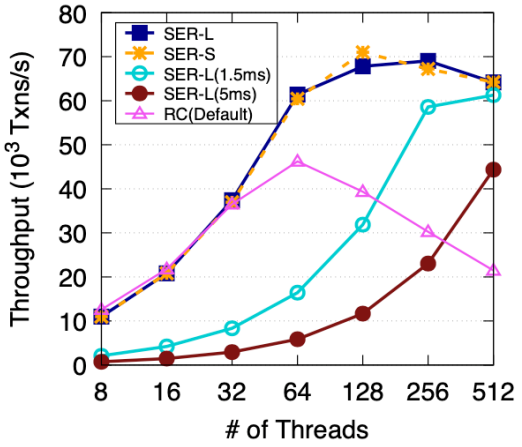

(b) Latency



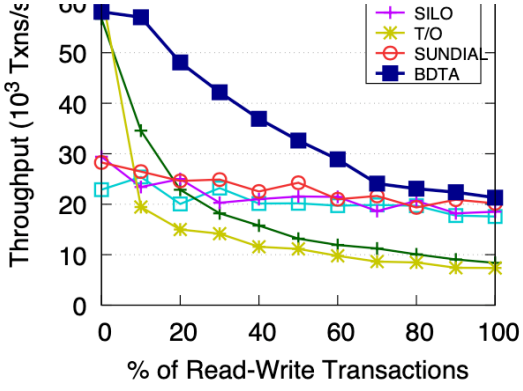

(a) YCSB-Workload-a




(b) YCSB-Workload-b



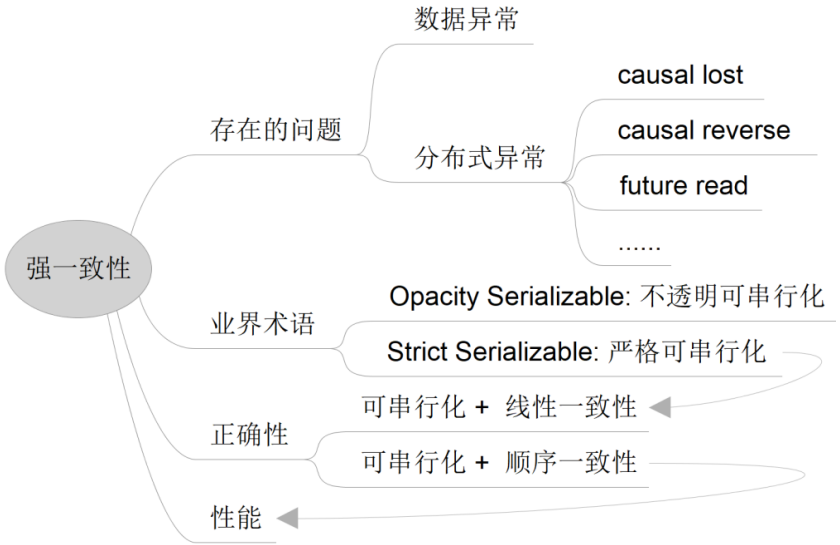

(a) Multi-level Serializability




(b) Concurrency Control Evaluations

# @小结 “强一致”的 分布式事务型数据库






3TS
腾讯事务处理技术验证系统

**1 第一个实现多种强一致性的工程系统**

**2 第一个高性能的强一致性工程系统**

**3 序幕拉开，真正的强一致高性能性系统将如雨后春笋.....**

## 作为一个事务技术相关的研究框架，3TS致力于探索事务的本质问题：

1. 分布式数据库的事务处理技术，会受到哪些因素的影响（可用性？可靠性？安全性？一致性？扩展性？功能？性能？架构？新硬件？AI？……）？
2. 分布式事务型数据库系统的评价、评测体系应该如何建立？
3. 世界上有多少种数据异常？数据异常的体系化研究方法如何建立？
4. 为什么并发访问控制算法会有很多种？各种并发访问控制算法之间，有没有的本质的关联关系？

# THANKS