2019
05
08-10
北京新云南皇冠假日酒店

数据风云 十年变迁

DTCC 第十届中国数据库技术大会

DATABASE TECHNOLOGY CONFERENCE CHINA 2019

IT168 ChinaUnix ITPUB

基于HLC的分布式事务实现深度剖析

阿里云智能事业群-数据库产品事业部
蔡乐、何登成

何登成，花名圭多。阿里巴巴资深技术专家

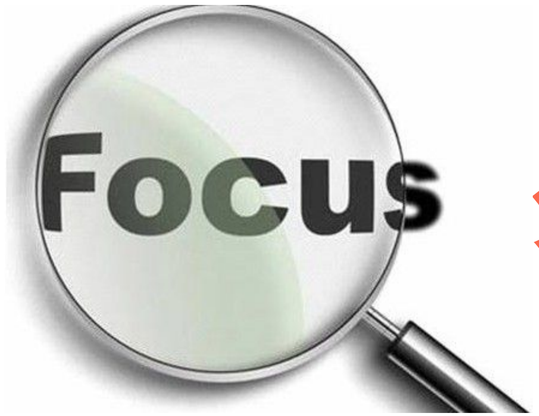
- 浙江大学计算机学院本科、研究生，师从陈刚老师。2005年至今，一直专注在数据库领域，先后在神州通用、网易、阿里从事数据库研发和管理工作
- 连续多年阿里巴巴双11、双12、支付宝新春红包大型活动数据库总负责人
- 目前负责阿里巴巴分布式数据库POLARDBX的研发工作

分布式事务关键技术点

节点内并发处理

单节点的优化

时钟方案



分布式事务管理

新硬件的使用

隔离级别支持
和演进技术

数据库为什么需要时钟： 为事务排序

- 通过事务对外提供数据相关操作的ACID
- 数据库对事务顺序的标识决定事务的原子性和隔离性（A和I）
- 时钟：日志LSN，事务ID，时间戳

数据库为什么需要时钟：支持MVCC

- 许多商业和开源数据库产品都支持MVCC
- MVCC通过支持数据的多版本，允许读写相同数据可以并发，在读多写少的场景下极大提升性能
- 数据的多版本需要对数据进行时钟标识

Single Value

Key	Value
A	"current_value"
B	"value_of_b"

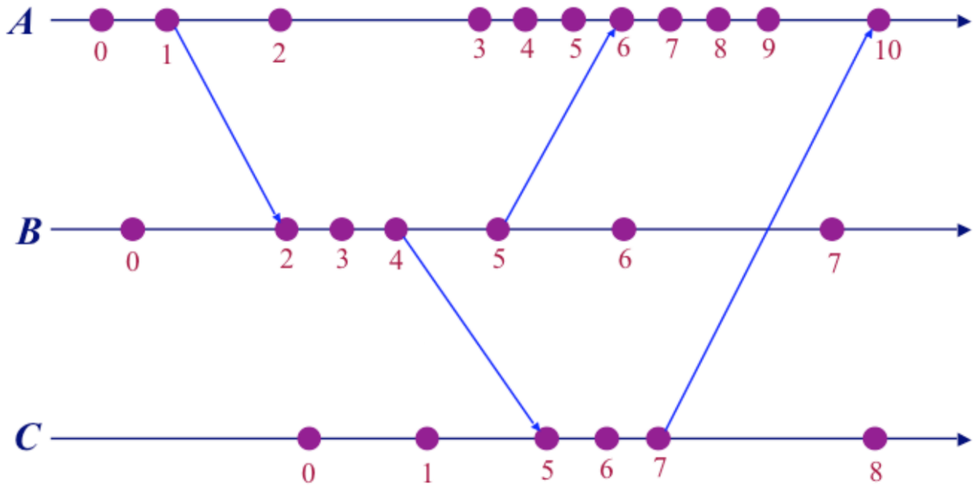
Multi-versioned Values

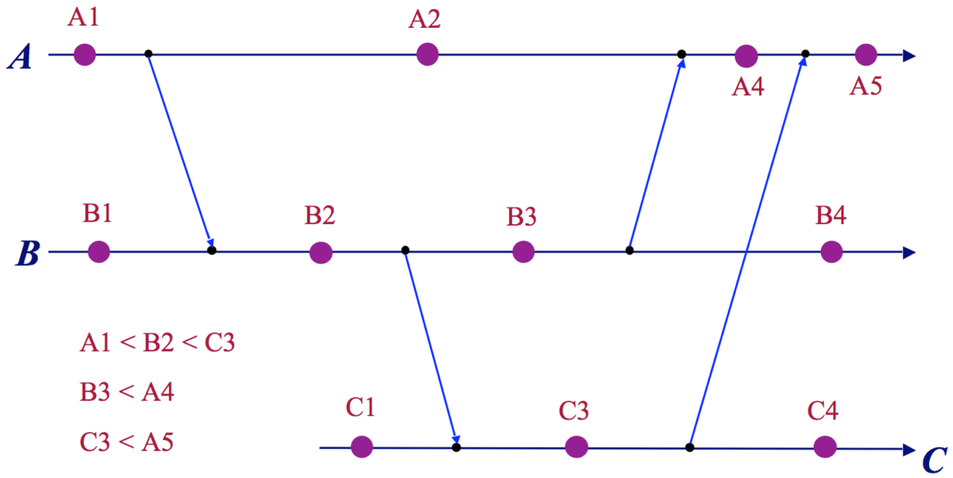
Key	Timestamp	Value
A	400	"current_value"
A	322	"old_value"
A	50	"original_value"
B	100	"value_of_b"

分布式数据库下的时钟

- 单机数据库：日志LSN或服务器的时钟
- 分布式数据库：数据库实例运作在多台服务器上
- 每个数据库实例有独立的时钟或日志（LSN），不能反映全局的顺序，比如服务器间有时钟偏移。
- 解决方案：引入类似单机系统的中心时钟

Lamport 时钟（LC）





- 分布式时钟：每个实例本地维护，单调递增counter
- Counter递增：本地事件发生时； 收到其它实例的消息时
- LC决定事件的happen-before关系，比如A1，B2和C3

```
# event is known
   time = time + 1;
# event happens
   send(message, time);
#receiving a message
   (message, time_stamp) = receive();
   time = max(time_stamp, time) + 1;
```

混合逻辑时钟（HLC）

HLC	l.j	和j节点有通讯的所有节点的时间戳的最大值
Logic Time	c.j	随因果事件发生而递增
物理时间	pt.j	机器wall time

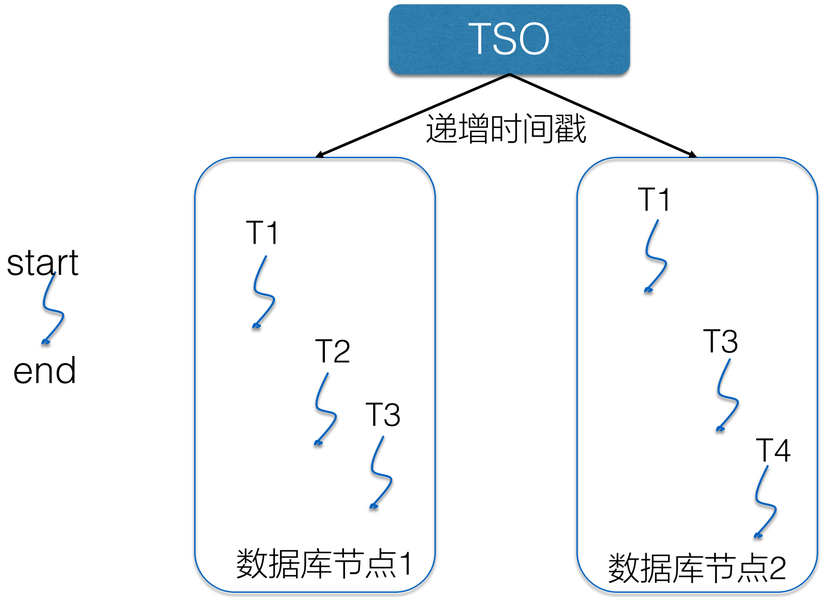
Initial l.j := 0; c.j :=0
- Send or local event
  l'.j := l.j;
  l.j := max(l'.j, pt.j);
  IF (l.j == l'.j) THEN c.j := c.j + 1
  ELSE c.j := 0;
  Timestmap with l.j, c.j

- Receive event of message m
  l'.j := l.j;
  l.j := max(l'.j, l.m, pt.j);
  IF (l.j == l'.j == l.m) THEN c.j :=max(c.j, c.m) + 1
  ELSEIF (l.j == l'.j) THEN c.j:= c.j + 1
  ELSEIF (l.j == l.m) THEN c.j:= c.m + 1
  ELSE c.j := 0
  Timestamp with l.j, c.j

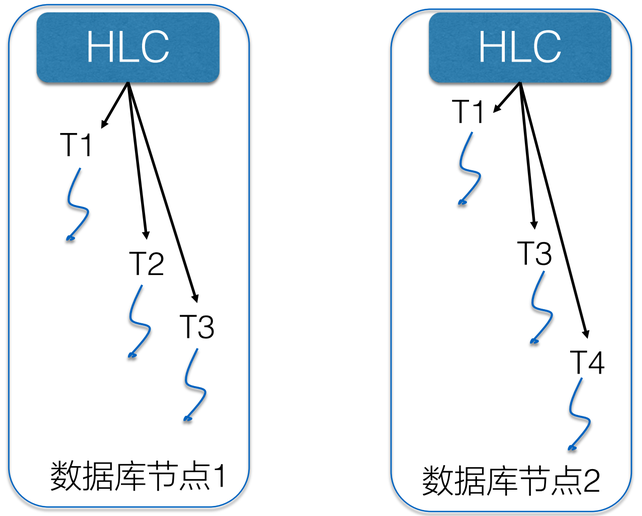
- 当物理时钟推进时， 逻辑时钟部分被置零
- HLC记录事件的因果关系，保证和物理时钟的偏差是bounded

HLC和中心时钟的差别

- 中心时钟（TSO）：为所有事务排序
- 分布式数据库中的HLC： 为具备数据库定义的因果关系的事务排序
  - 因果关系（happen before）：一个事务在另外一个事务开始前已经提交，并且它们访问了或发生在相同的节点；



T1,T2,T4的提交时间：t1 < t2 < t4



T1,T2,T3的提交时间：th1 < th2 ； th1 < th3
th2 和 th4 ?

时钟方案

TiDB

中心式时钟（TSO）

Postgres-XL

中心式事务列表（GTM）

ORACLE DATABASE

SCN

Cockroach DB

混合逻辑时钟
（HLC）

Google Cloud Spanner

Truetime

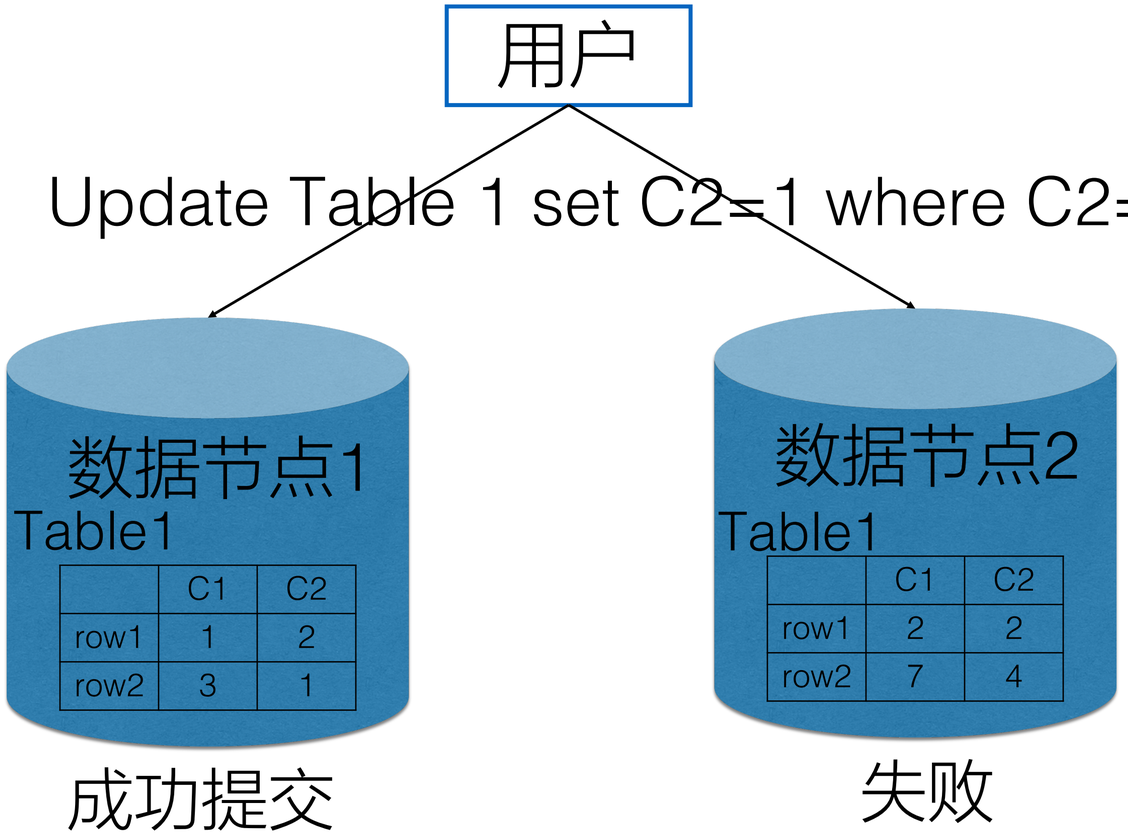
中心式 VS. 分布式 VS. Truetime

	中心式	分布式	Truetime
优点	全局一致的时间 实现简洁	无中心化的性能和HA瓶颈 HLC可以替代wall time使用	全局一致时间 简化应用开发
缺点	• 时钟获取都涉及网络延时 • 跨分区跨region下性能损耗大 • 中心时钟服务成为性能和HA的瓶颈	• 时钟和DB逻辑耦合， • 缺乏中心时钟，增加调试复杂度 • 外部一致性的实现性能受时钟skew影响	• 需要专有硬件 • 性能受时钟skew影响 • 需要配套的并发协议

分布式事务管理

保证全局读写原子性和隔离性

分布式事务的原子性例子（1）

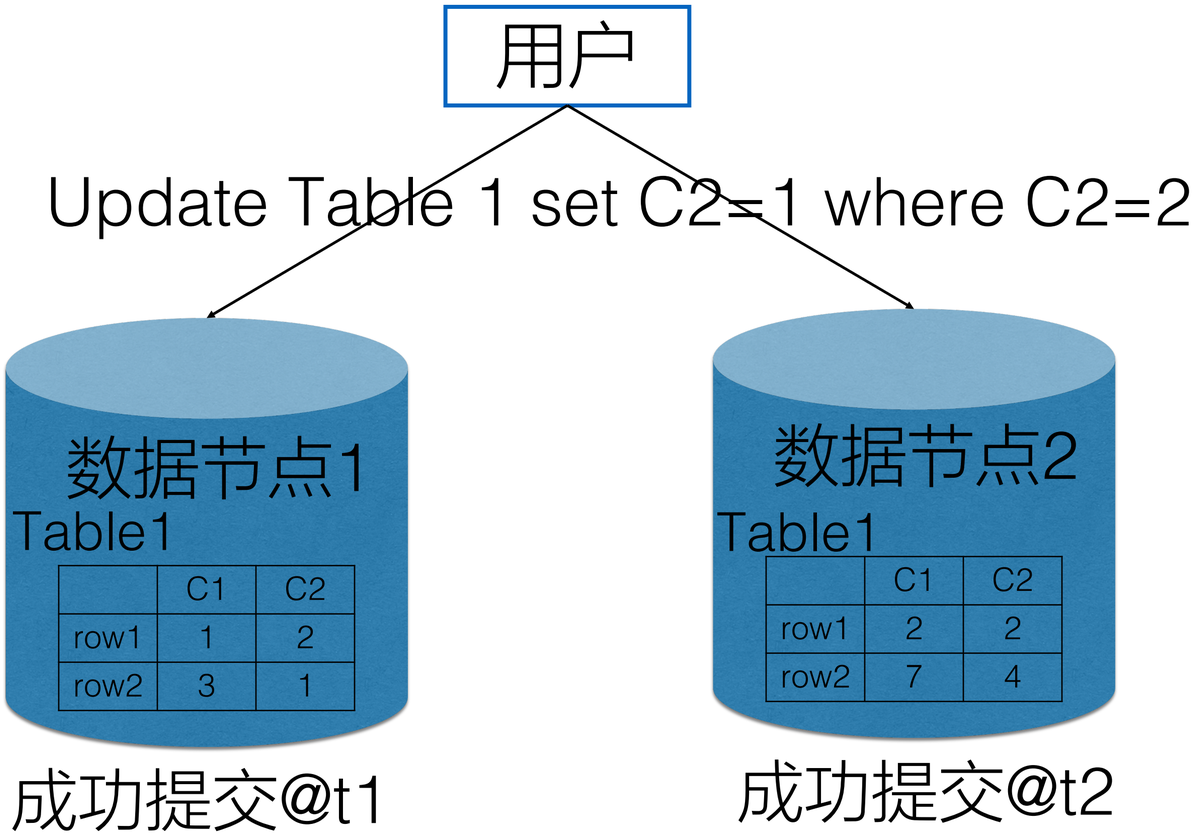


最后的结果是部分的

- 数据节点1上，row1是（1，1）
- 数据节点2上，row1是（2，2）

丧失事务的原子性

分布式事务的原子性例子（2）

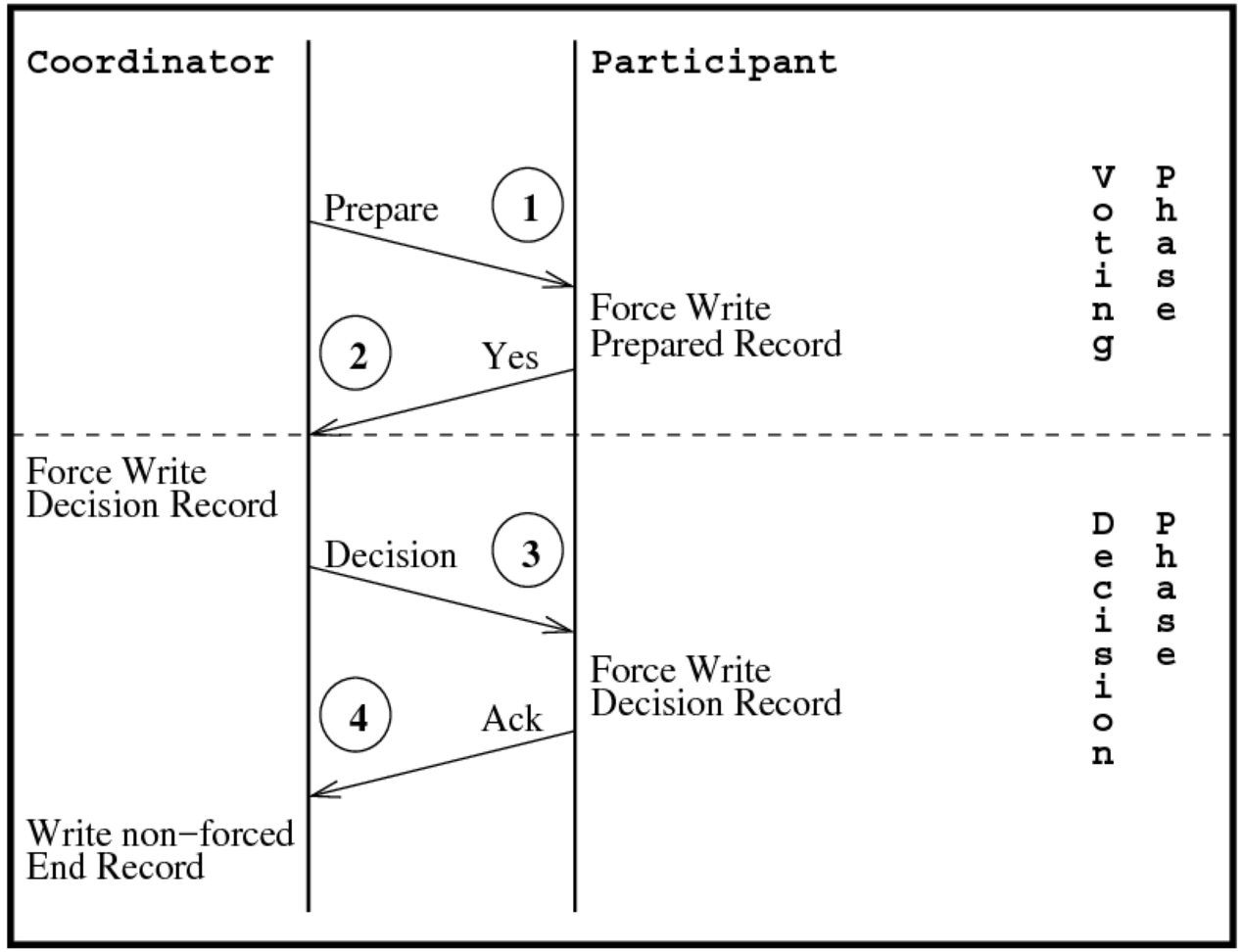


最后的结果是部分可见

- 虽然数据节点1和数据节点2都成功提交，但是提交的时间点不一样
- 如果按照本地的提交时间点决定可见性，则导致事务结果是部分可见

丧失事务的原子性

两阶段提交



From "Atomic commit protocols, their integration, and their optimisations in distributed database systems", International Journal of Intelligent Information and Database Systems 4(4):373-412 · September 2010

- Prepare阶段
  - 协调节点通知所有参与的数据节点去prepare事务
  - 数据节点持久化日志，返回成功
- 提交阶段
  - 当协调节点收到所有参与数据节点的prepare成功后，通知数据节点提交
  - 如果有数据节点没有成功prepare，通知所有参与数据节点去abort事务

分布式事务管理

TiDB

Postgres-XL

Google Cloud Spanner

FOUNDATIONDB

FAUNA

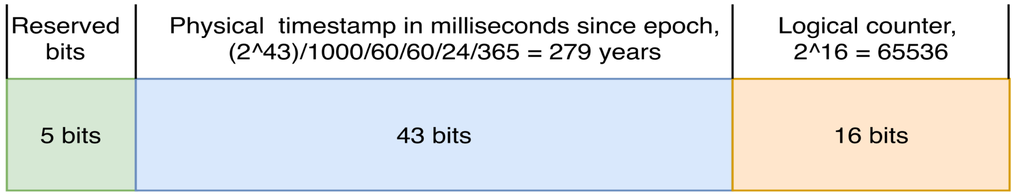
两阶段提交

MVOCC

确定性事务

HLC和两阶段提交

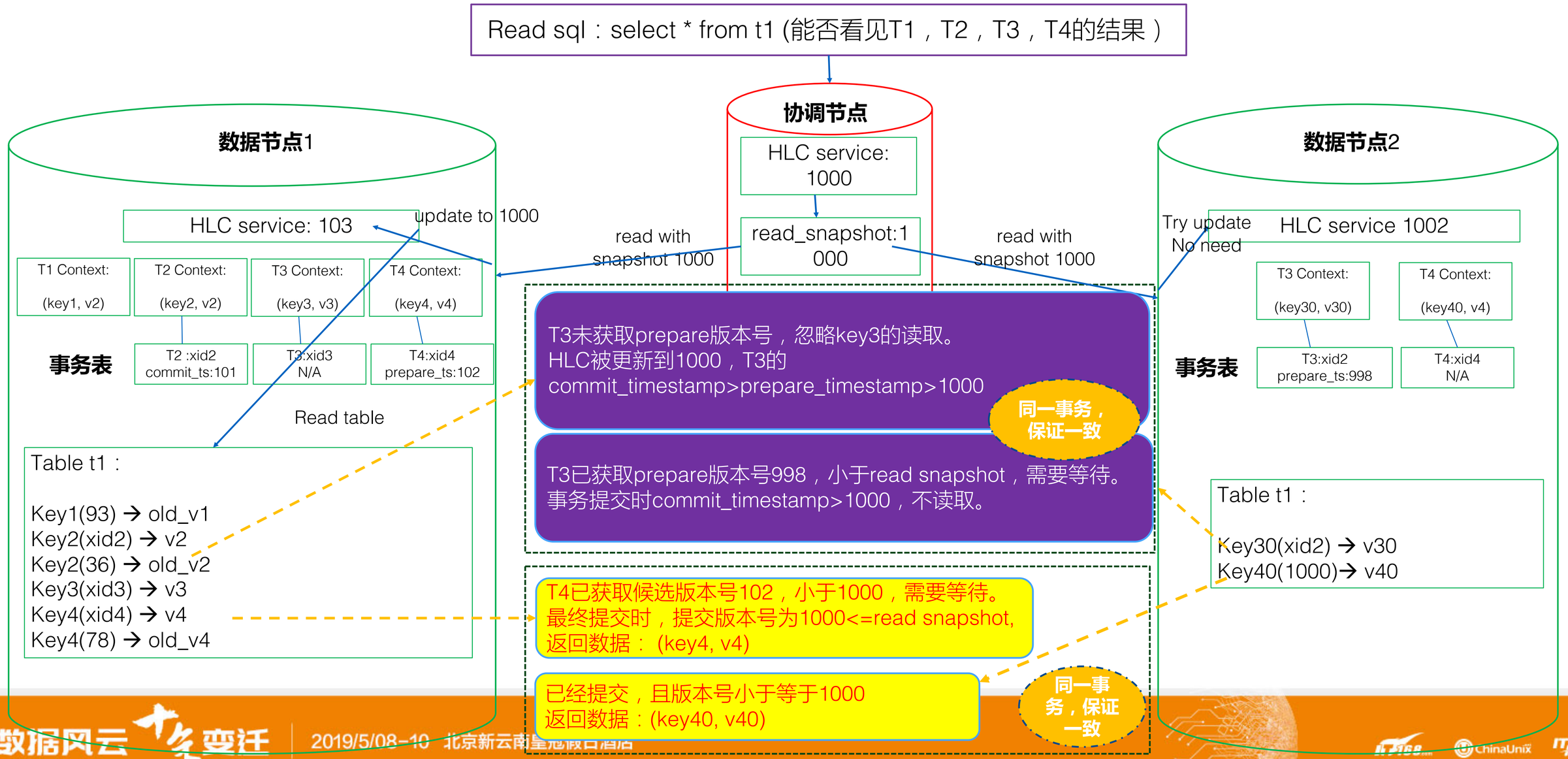
HLC的格式



HLC更新和两阶段提交的例子

事务操作	协调节点 HLC	数据节点1 HLC	数据节点2 HLC
begin	1000，0	1001，10	999，20
Select * from t1；		1001，10	1000，0
Insert into t1 values（数据节点1 only）			
Insert into t1 values（数据节点2 only）			
Prepare数据节点1		1002，0（协调者）	
Prepare数据节点2			1001，0
Commit数据节点1		1002，0（1002，0）是提交时间	
Commit数据节点2			1002，0
返回协调节点	1002，0		

分布式下的一致性读取


Read sql：select * from t1 (能否看见T1，T2，T3，T4的结果）
协调节点
HLC service: 1000
read_snapshot:1000
read with snapshot 1000
read with snapshot 1000
数据节点1
HLC service: 103
update to 1000
T1 Context: (key1, v2)
T2 Context: (key2, v2)
T3 Context: (key3, v3)
T4 Context: (key4, v4)
事务表
T2 :xid2 commit_ts:101
T3:xid3 N/A
T4:xid4 prepare_ts:102
Read table
Table t1：
Key1(93) → old_v1
Key2(xid2) → v2
Key2(36) → old_v2
Key3(xid3) → v3
Key4(xid4) → v4
Key4(78) → old_v4
T3未获取prepare版本号，忽略key3的读取。
HLC被更新到1000，T3的commit_timestamp>prepare_timestamp>1000
同一事务，保证一致
T3已获取prepare版本号998，小于read snapshot，需要等待。
事务提交时commit_timestamp>1000，不读取。
T4已获取候选版本号102，小于1000，需要等待。
最终提交时，提交版本号为1000<=read snapshot,
返回数据：(key4, v4)
已经提交，且版本号小于等于1000
返回数据：(key40, v40)
同一事务，保证一致
数据节点2
Try update
No need
HLC service 1002
T3 Context: (key30, v30)
T4 Context: (key40, v4)
事务表
T3:xid2 prepare_ts:998
T4:xid4 N/A
Table t1：
Key30(xid2) → v30
Key40(1000)→ v40

HLC时钟偏移的问题

- HLC和peak TPS的关系
  - 最大TPS=logical counter的最大值/单事务刷新HLC次数/物理时钟部分的最小单元或时钟偏移
  - 没有时钟偏移的情况：Peak TPS = 2^16 / 2 / 0.001 = 3 千万
  - 时钟偏移 5ms： Peak TPS = 2^16 / 2 / 0.005 = 6百万
  - 时钟偏移将导致peak TPS的大幅下降

- 解决方案
  - 设定最大允许时钟偏移
    - 如果有节点的时钟偏移大于最大值：
      - 强制下线时钟偏移大的节点（或switchover到备机）
      - 不让数据库实例在节点上启动
  - 允许logic counter overflow到物理时钟部分

其他问题

- 超时异常和故障
- 节点宕机，本地恢复
  - 参与者宕机恢复
  - 协调者宕机恢复
- 多副本下的协调者节点高可用
    - 切换到参与者的新主
    - 异地重建

总结

- 混合逻辑时钟（HLC）
  - 本地获取，避免中心时钟的性能瓶颈和单点故障
  - 维护跨实例的事务/事件的因果关系（happen-before）
- 分布式事务
  - 为分布式数据库提供一致性数据访问的支持：原子性，隔离性
  - 一体化分布式数据库的用户体验

扫码获取阿里云数据库代金券

数量有限 先到先得

欢迎关注
阿里巴巴数据库技术 公众号


THANKS