PingCAP

Transaction

dongxu@PingCAP

PingCAP

隔离级别

- SI (Snapshot Isolation)
  - Repeatable Read
- SSI (Serialzation Snapshot Isolation)
  - Avoid Write Skew

Write Skew

A = 1 B = 2

Begin()	Begin()
Set A = B	Set B = A
Commit()	Commit()

SI :

没有写冲突
可能出现某一个事务的提交结果被无效化，或者事务提交时，依赖的读的结果已经不一样了

可能出现 A = 2 B = 1 的结果

SSI :

可以检测到冲突或序列化两个事务

只会出现 A = B = 2 或 A = B = 1

PingCAP

TiDB 的 SSI

- 默认隔离级别 SI
- 不同于 MySQL, TiDB 的实现是 SI + Lock
  - 将读显式加锁

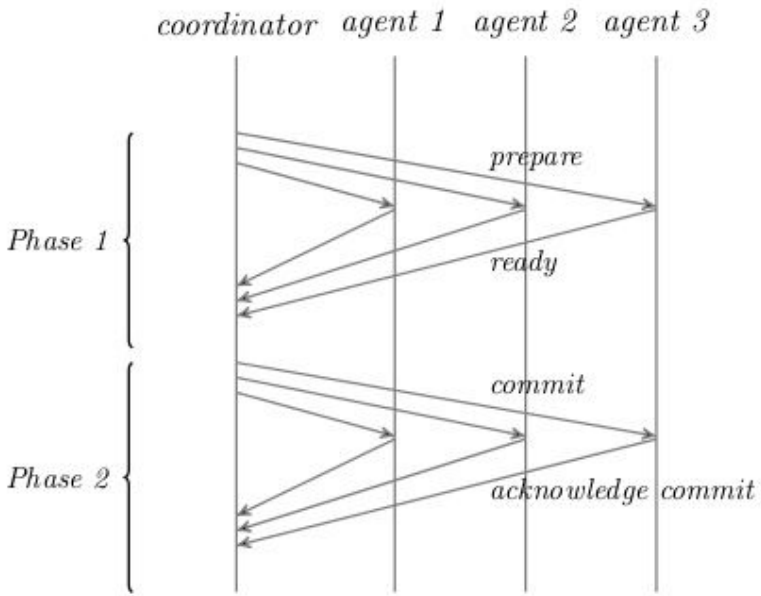
PingCAP

分布式事务

只有一种办法

两阶段提交

2-Phase Commit



PingCAP

事务管理器

- 解决读写 / 写写冲突
  - ACID 事务
  - 标记事务是否已经提交，数据是否可见
- 传统数据库中事务管理器一般是一个独立模块
  - Yahoo OMID
  - Palantir AtalsDB
- 去中心化的事务管理

PingCAP

Google Percolator

- 基于 BigTable 的事务层
  - 支持跨行事务
- 用于批量索引更新等业务
- 没有独立的事务管理器
  - 但仍有单点授时服务器（TSO）
- SI 隔离级别

PingCAP

Google Percolator

- 两级锁
  - Primary Lock
  - Secondary Lock
- 两阶段提交

PingCAP

Google Percolator

key	*bal:data*	*bal:lock*	*bal:write*
Bob	*6:* *5:* $10	*6:* *5:*	*6:* data @ 5 *5:*
Joe	*6:* *5:* $2	*6:* *5:*	*6:* data @ 5 *5:*

1. Initial state: Joe's account contains $2 dollars, Bob's $10.

Bob	*7:***$3** *6:* *5:* $10	*7:* **I am primary** *6:* *5:*	*7:* *6:* data @ 5 *5:*
Joe	*6:* *5:* $2	*6:* *5:*	*6:* data @ 5 *5:*

Column	*Use*
c:lock	An uncommitted transaction is writing this cell; contains the location of primary lock
c:write	Committed data present; stores the Bigtable timestamp of the data
c:data	Stores the data itself
c:notify	Hint: observers may need to run
c:ack_O	Observer "O" has run ; stores start timestamp of successful last run

PingCAP

Percolator 的问题

- 引入了 Shadow Column
  - Lock
  - Write
- 依赖单点 TSO
- 失效锁的清理
  - Google 内部依赖 Chubby

PingCAP

Percolator 的优势

- 几乎是去中心化的模型
  - 问题转换成将 TSO 去中心化的问题
- 实现简单，不容易出错

TiKV MVCC

- RocksDB
  - LSM-Tree

KeyA	Meta of KeyB
KeyA_Version1	aaa
KeyA_Version2	aaaa
KeyA_Version3	aaaaa
KeyB	Meta of KeyB
keyB_Version2	bbb
...	...

TiKV MVCC

```
message Meta {
    optional MetaLock lock         = 1;
    repeated MetaItem items        = 2;  // 逆序排列(版本号
较大的在前面)
}


// 每次成功提交的数据对应 Meta 中的一条 MetaItem 记录
message MetaItem {
    // start_ts 是此版本数据对应的事务号，同时用于拼接对应的 DataKey
    optional uint64 start_ts    = 1;
    // 此版本数据在 commit_ts 后可见
    optional uint64 commit_ts   = 2;
}
```

TiKV MVCC

```
message MetaLock {
    enum Type {
        // ReadOnly 对应 LockKey() 操作
        // Commit 和 Rollback 时直接清除 Lock
        ReadOnly  = 1;
        // ReadWrite 对应 Put/Delete
        // Commit 时需插入一项 MetaItem, Rollback 时需删除 Data
        ReadWrite = 2;
    }
    optional Type   type        = 1;
    optional uint64 start_ts    = 2;    // 标识锁住该 Key 的事务
    optional bytes  primary_key = 3;    // 标识锁住该 Key 的事务的 PLock 的位置
}
```

PingCAP

TiKV Transaction

- 所有参与事务的 Key 都在同一个 Region 可以简化成一阶段提交
- Batch Prewrite
  - Group keys by region
- 异步 Commit Secondaries

PingCAP

Thanks

Q & A