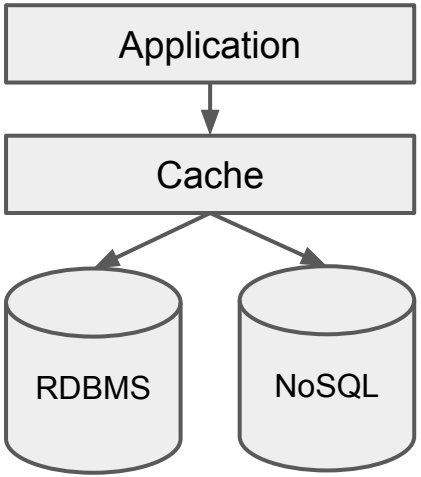
分布式 NewSQL 数据库实现

以 TiDB 为例

关于我

PingCAP 创始人兼职 CEO
微博:@goroutine
组织:github.com/pingcap
TiDB: github.com/pingcap/tidb
致力于打造开源的 Google F1

数据存储是应用的核心



数据存储的模型影响到软件开发的方方面面

- RDBMS
  - MySQL
  - PostgresSQL
- NoSQL
  - MongoDB
  - HBase
  - Cassandra

RDBMS

Prof.

- 事务
- SQL
- Schema
- 生态
  - 开发者数量
  - Language support
  - ORMs

Cons.

- Scale 能力弱
- 容量和吞吐受单机影响大
- 生产环境中 Schema 变更比较痛苦

RDBMS

- Scale Up
  - 仍然没有摆脱单机的局限
- Scale Out
  - 分库/分表
  - 水平 Sharding（Cobar, Vitess, DRDS, Kingshard...）

Scale Out 是目前大多数公司的选择，大多数业务为了追求高可用性选择了水平扩展的方案，但是这个方案带来的问题是放弃了事务和全局一致性。

NoSQL

- HBase
- Cassandra
- MongoDB

追求 Scalability
放弃 SQL
放弃跨行事务

Prof.

- 扩展能力
- 高吞吐
- 多样化的数据模型
- 一致性级别
  - 强一致（Hbase）
  - 最终一致性（C*）

Cons.

- 事务
- SQL 支持
- Schema

NoSQL => NewSQL

- SQL
- 分布式事务
  - SI/SSI
- 可扩展性

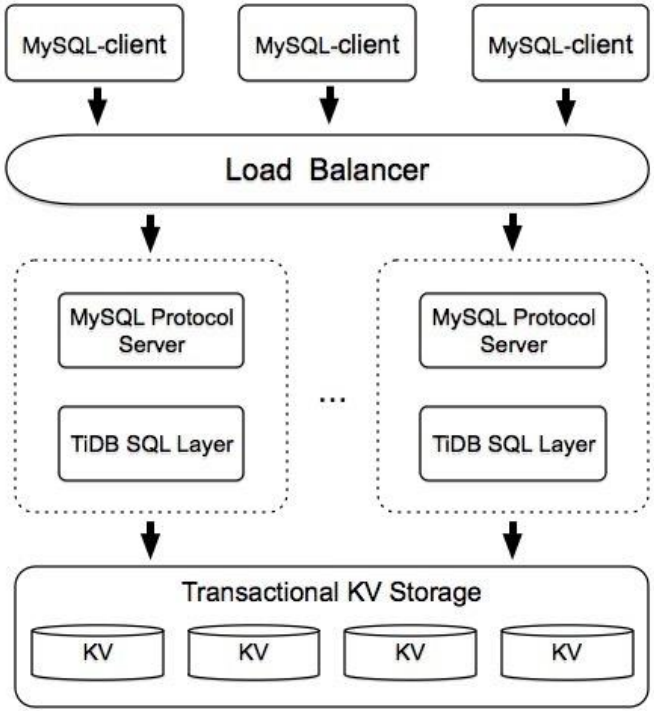
- FoundationDB
- VoltDB
- OceanBase
- Google F1

一些基本概念

- **DDL**
  - **CREATE / DROP INDEX**
  - **CREATE PRIMARY INDEX**
  - **ALTER TABLE**
  - **ADD COLUMN/REMOVE COLUMN**
- **DML**
  - **UPDATE**
  - **DELETE**
  - **INSERT**
  - **UPSERT**
  - **MERGE**

How to build a distributed SQL database from scratch

1. SQL Parser / 执行引擎
2. 事务隔离
3. 分布式 Kv


MySQL-client
MySQL-client
MySQL-client
Load Balancer
MySQL Protocol Server
TiDB SQL Layer
...
MySQL Protocol Server
TiDB SQL Layer
Transactional KV Storage
KV
KV
KV
KV

SQL Parser

- EBNF
- yacc/lex
  - cznic/ebnf2y
  - cznic/goyacc
  - cznic/golex
- 文法参考 MySQL 的 yacc 文件

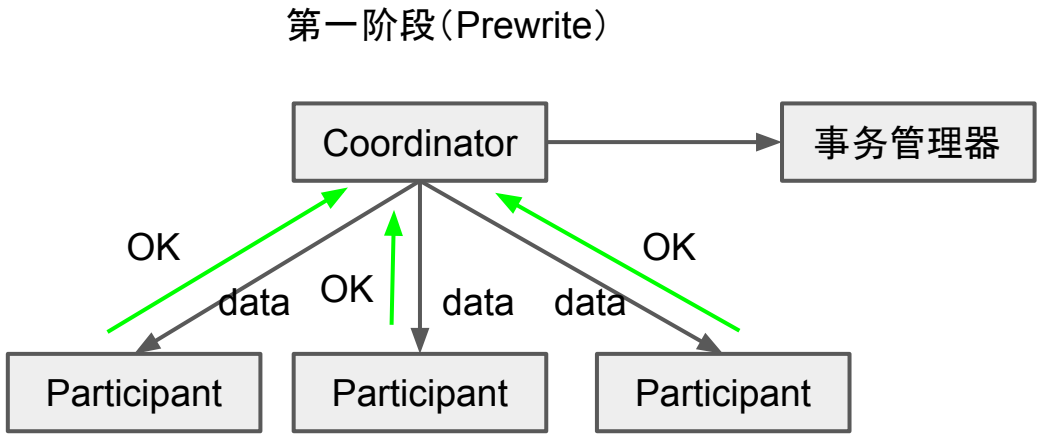
执行引擎

- Plan 生成
  - 从 AST 到执行计划树
- Plan 优化
  - 识别索引
  - 并行优化
- DDL
  - 异步 schema 变更的重要性
  - 元信息存储

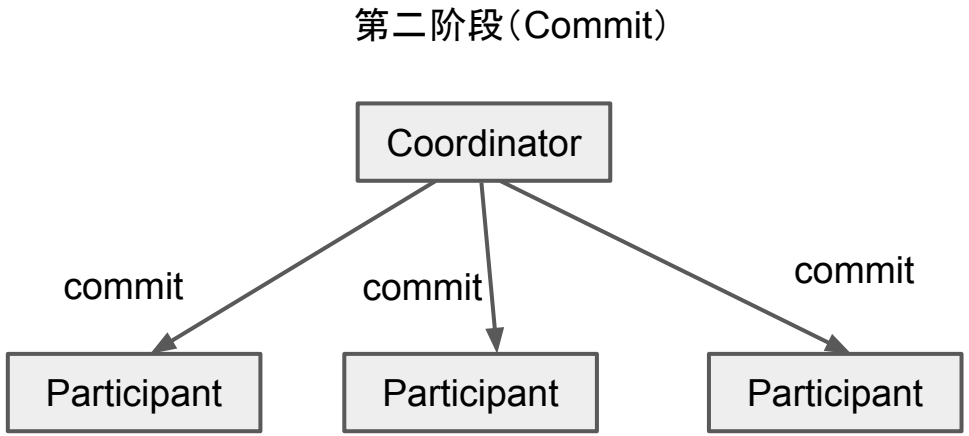
分布式事务

- 二阶段提交 (2PC)
- 隔离级别
  - SI
  - SSI
- MVCC
- 冲突处理

二阶段提交 (2PC)


第一阶段（Prewrite）
Coordinator
事务管理器
OK
data
OK
data
data
OK
Participant
Participant
Participant

二阶段提交 (2PC)


第二阶段（Commit）
Coordinator
commit
commit
commit
Participant
Participant
Participant

二阶段提交 (2PC)

- 问题
  - 协调者如何选择
    - 如果同时有多个协调者会有什么问题？
    - 去中心化的 2PC
  - 第二阶段如果参与者挂掉怎么办？
    - 如何错误恢复
  - 事务管理器
    - Paxos
  - 事务的时序问题
    - Spanner
    - 中心授时服务（percolator）

协调者选取

- 单协调者
  - 单点问题
- 多协调者
  - 一致性问题

错误恢复

- 第一阶段错误

  某个参与者写入失败、超时。协调者无法收到这个参与者的确认信息，终止事务

- 第二阶段错误

  由于已经进入第二阶段，意味着第一阶段所有的参与者已经回复 OK，并写入本地 WAL，如果此时宕机，下次恢复的时候会根据 WAL 中的 Transaction 信息去事务管理器查询是否提交或者丢弃

事务管理器

- 唯一的事务编号
  - 状态查询
  - 隔离
- 全局有序
  - 冲突处理
  - 时间戳生成策略
    - 中心授时服务器
    - 原子钟

隔离级别

- 可重复读（RR, SI）
  - MVCC 的实现
    - LMDB
    - LevelDB
- 序列化（SSI）
  - 全局事务有序
  - 分布式系统的 SSI
    - Spanner

隔离级别

Transaction 1	Transaction 2
UPDATE t SET x = x + 1;	UPDATE t SET x = x + 1;
COMMIT;	COMMIT;

初始状态：假设只有一行，x = 0
SI: 两个事务有一个成功，另外一个 ROLLBACK，然后重试，成功，结果为 2

隔离级别　select for update 支持

Transaction 1	Transaction 2
select x from t for update; select x from t; 用户逻辑代码 update t set x = x + 1; commit	select x from t for update; select x from; 用户逻辑代码 update t set x = x + 1; commit

初始状态:假设只有一行，x ＝ 0
SSI: 只有一个会成功，另外一个会 abort，需要用户自己检测并重试

隔离级别　select for update 支持

Transaction 1	Transaction 2
select x from t for update; select x from t; 用户逻辑代码 update t set x = x + 1 where x = 1; commit	select x from t for update; select x from; 用户逻辑代码 update t set x = x + 1 where x = 1; commit

SSI: 只有一个会成功，另外一个会 abort，需要用户自己检测并重试

隔离级别　select for update 支持

Transaction 1	Transaction 2
select x from t for update; select x from t; 用户逻辑代码 update t set x = x + 1 where x = 1; commit	select x from t for update; select x from; 用户逻辑代码 update t set x = x + 1 where x = 1; commit

SSI: 只有一个会成功，另外一个会 abort，需要用户自己检测并重试

冲突处理

- 悲观锁
  - 先上锁后操作
- 乐观锁
  - 先操作，提交时检查版本
  - TiDB 默认采用乐观锁

TiDB 计划

- 完善 SQL 层，多本地引擎 (1 month)
  - BoltDB，RocksDB，LMDB，LevelDB...
  - 尽量与 MySQL 接近
  - 支持主流 ORM，不同语言的 MySQL Driver
- 异步 Schema 变更 (2 months)
- HBase 引擎（6 months）
  - Percolator 模型
  - 真正的分布式数据 库
- 自建 KV 引擎
  - 针对 SQL 优化的 Transactional Distributed KV
- 多租户/容器化