# 分布式数据库模式与反模式

黄东旭

# 关于我

- 黄东旭
- PingCAP, CTO / Cofounder
- MSRA / 网易 / 豌豆荚
- Infrastructure Engineer / Gopher / Hacker / Entrepreneur
- Coding for 15yrs
- Author and architect of Codis / TiDB / TiKV

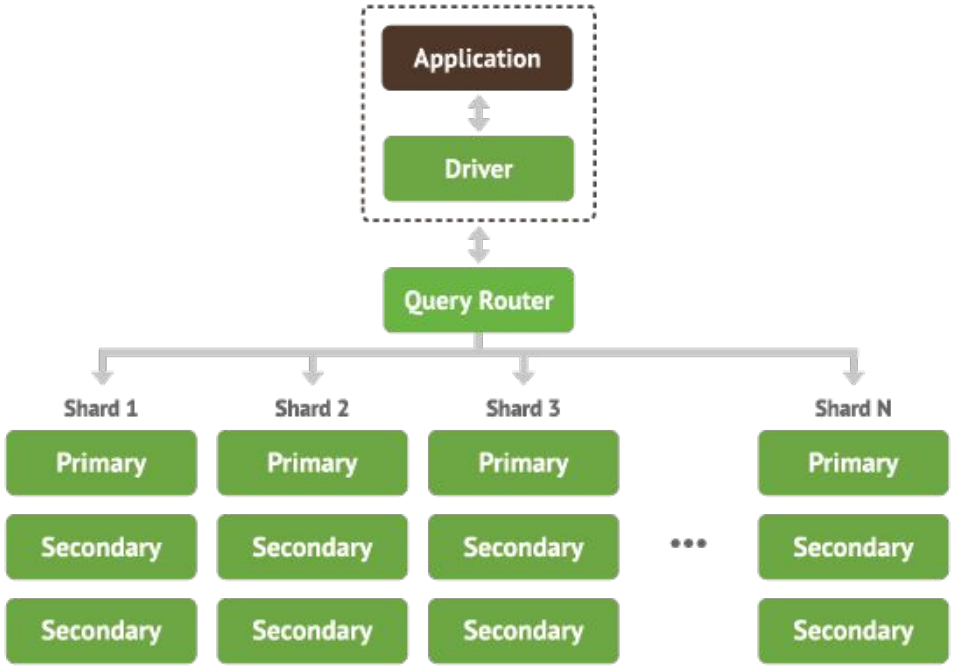
# 基础软件的现状

- 开源化是大趋势
- 分布式渐成主流
- 基础软件碎片化
- 微服务模式兴起

# 存储系统痛点日渐浮现

- 大数据量下如何弹性扩展？
- 分布式系统可用性如何定义和保证？
- 可维护性？
- 面向业务的开发复杂度？
  - 跨行事务
  - SQL
- ...

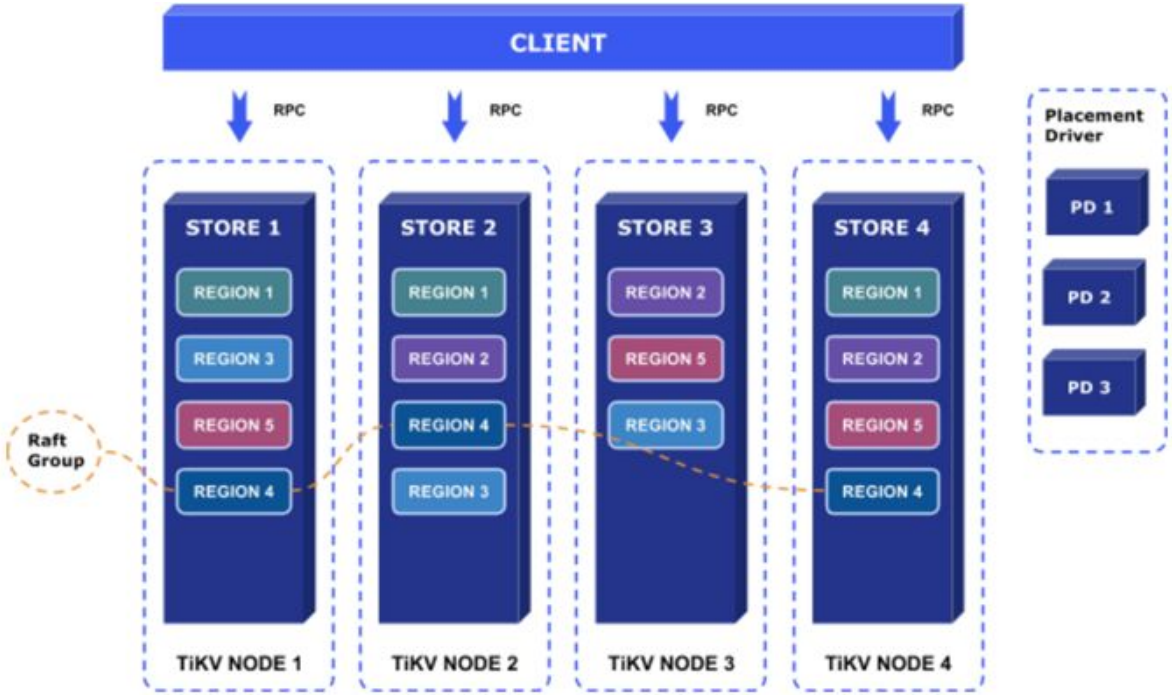
# 扩展模型 - Sharding



# 扩展模型 - Sharding

- 优势
  - 实现简单
  - 对于简单的业务场景兼容性好
- 劣势
  - 对业务层有侵入性
  - 分片固定，自动化程度低，扩展性差，很难实现按需弹性扩展
  - 无法实现复杂查询优化及高效透明的事务
  - 维护成本高

# 扩展模型 - Region


CLIENT
RPC
RPC
RPC
RPC
Placement Driver
PD 1
PD 2
PD 3
STORE 1
REGION 1
REGION 3
REGION 5
REGION 4
STORE 2
REGION 1
REGION 2
REGION 4
REGION 3
STORE 3
REGION 2
REGION 5
REGION 3
STORE 4
REGION 1
REGION 2
REGION 5
REGION 4
Raft Group
TiKV NODE 1
TiKV NODE 2
TiKV NODE 3
TiKV NODE 4

# 扩展模型 - Region

- 优势
  - 可以实现弹性扩展
  - 高度去中心化
  - 具有一定的自动 Failover 能力
- 劣势
  - 实现相对复杂
  - 业务层兼容性

# 可用性级别

高可用的幻觉：主从模型

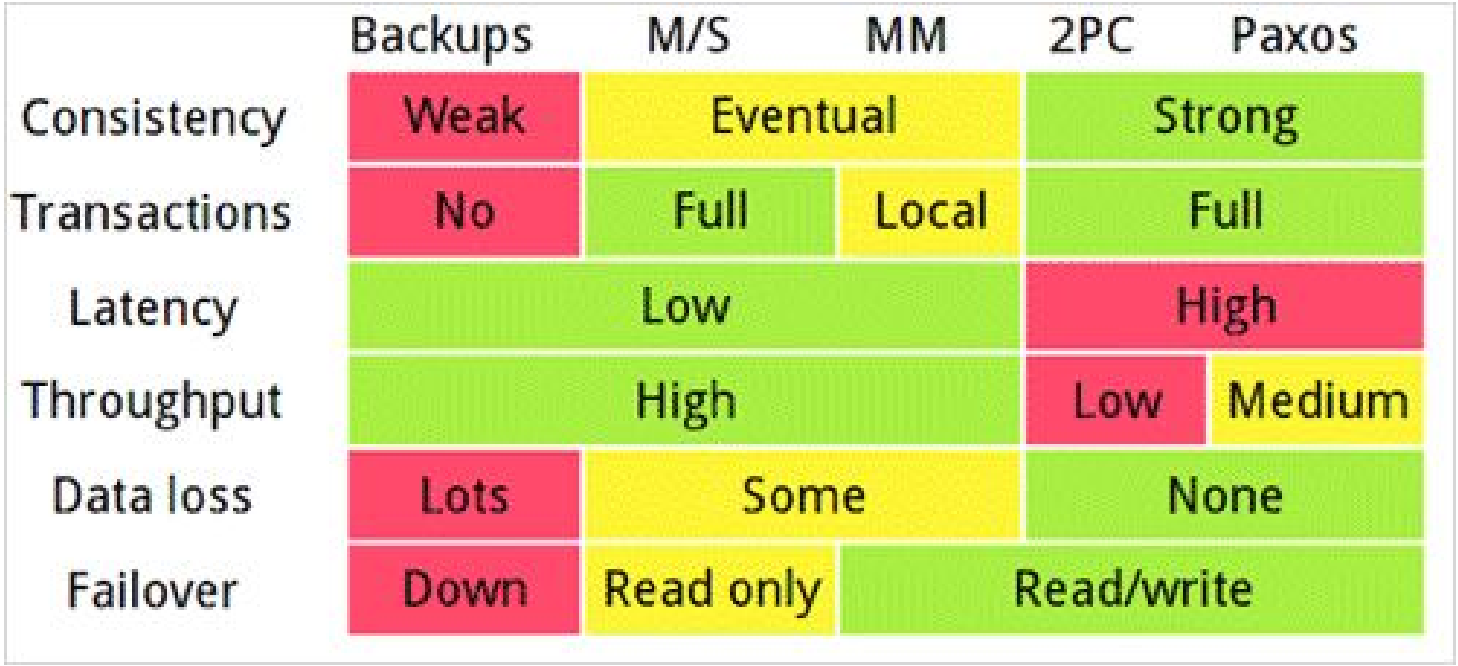
- 主从模型不能同时满足强一致性和高可用性
  - 由于集群脑裂的存在
- 说好的异地多活呢

# 可用性级别

唯一的高可用模型：分布式选举算法

- Multi-Paxos / Raft
- 强一致，自动的故障转移及数据恢复，可接受的延迟

# 一致性级别


Backups
M/S
MM
2PC
Paxos
Consistency
Weak
Eventual
Strong
Transactions
No
Full
Local
Full
Latency
Low
High
Throughput
High
Low
Medium
Data loss
Lots
Some
None
Failover
Down
Read only
Read/write

# NewSQL

- Scalability
- SQL
- ACID Transaction
- High Availability / Auto-Failover

# 案例：Google Spanner / F1

- Globally-distributed
  - Paxos
- SQL above NoSQL
- ACID Transaction support
  - TrueTime API
- Designed for Google AdWords, originally
  - became the successors of BigTable

# 业务场景

- 高吞吐，大容量
- Workload 相对分散
  - 反例：秒杀
- 一致性，可用性和延迟的取舍

# 典型场景: MySQL Sharding

特点:

- 高吞吐
- 海量并发小事务
- 模型相对简单
- 没有复杂查询

痛点:

- Scale 和 DDL
- 跨 Shard 事务

# 典型场景: Cross-datacenter HA

特点:

- 数据极端重要, 不能容忍数据不一致
- 0 downtime, 即使整个数据中心宕机都不能影响线上业务
- 异地多活

痛点:

- 目前没有一个数据库解决
- 主从复制方式不可靠, 而且再跨数据中心的场景下延迟过大
- 人工运维

# 反模式一：滥用传统关系模型

- 大量使用存储过程，外键，视图等
- 大表与大表的 JOIN
  - 网络通信的代价
  - OLAP 会更适合

# 反模式二：没有利用好并发

- 延迟和吞吐
- 查询间依赖关系过强

# 反模式三:不均匀的设计

- 业务中存在单点
  - 计数器
  - 秒杀
  - 队列
- 索引设计不合理
  - 全表扫描
  - 过多无效索引

# 反模式四:错误的一致性模型

- 滥用强一致分布式事务
- 根据冲突的频繁程度使用不同的锁策略
  - 悲观事务
  - 乐观事务