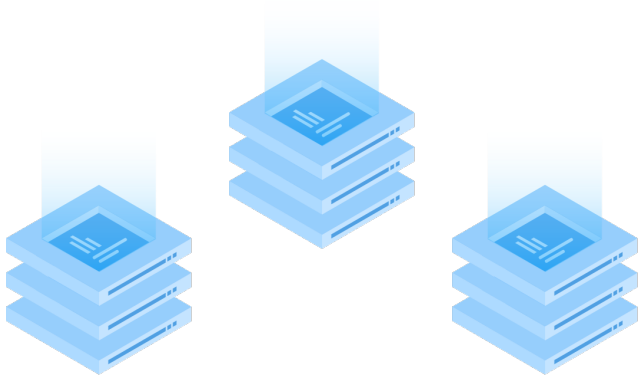
TiDB HTAP 的架构演进进展及实践

马晓宇

PingCAP TiDB



关注 InfoQ Pro 服务号

你将获得：

- InfoQ 技术大会讲师 PPT 分享
- 最新全球 IT 要闻
- 一线专家实操技术案例
- InfoQ 精选课程及活动
- 定期粉丝专属福利活动

About me

马晓宇

实时分析产品负责人@PingCAP

Big Data Infra 组 Tech Lead@网易杭研

关注：Distributed database，大数据

目录

PingCAP

HTAP 历史

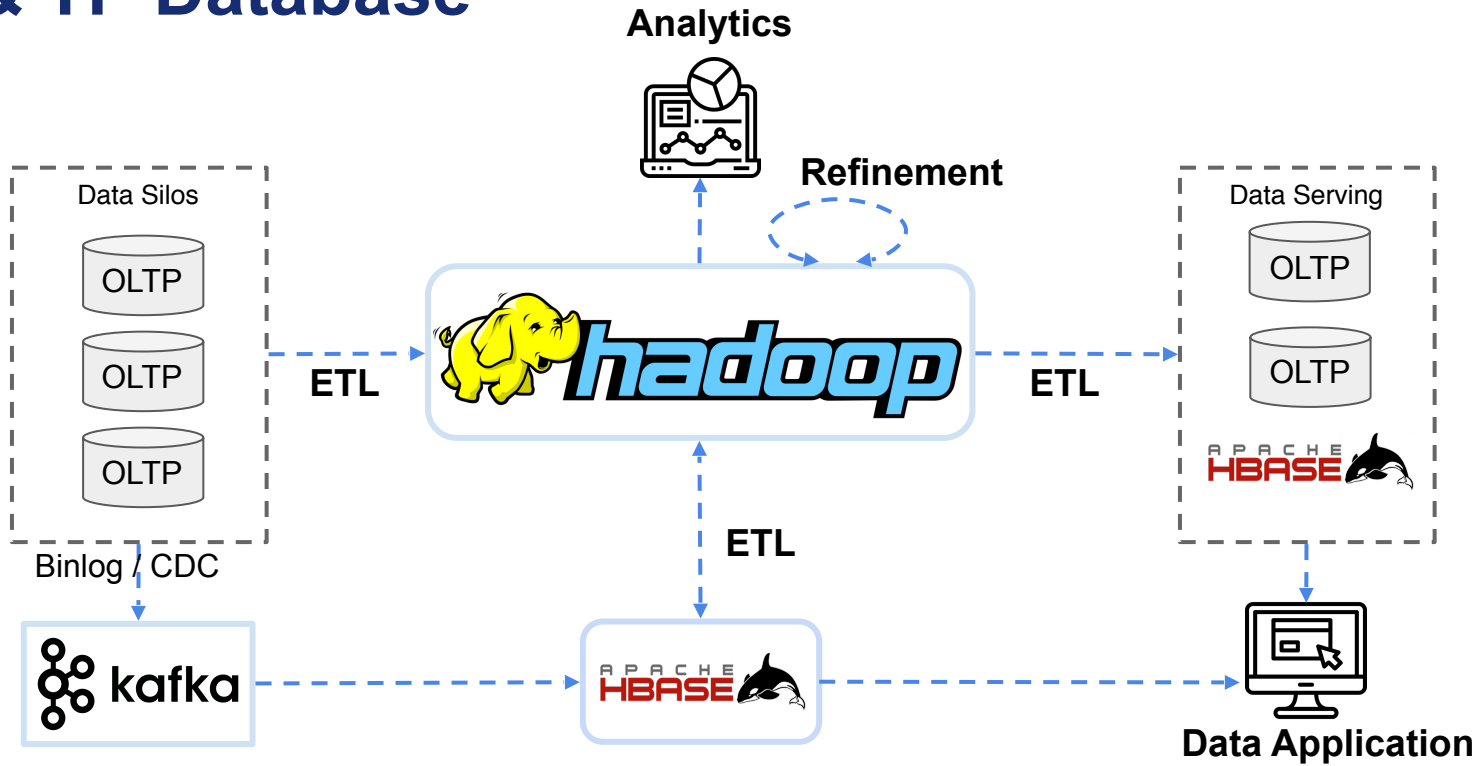
一点点历史

- **传统数据数据库都是 HTAP 的**
    - Oracle，DB2，SQL Server 等等
    - 一套引擎，一套代码
    - 对 TP 和 AP 业务都能处理（虽然往往 AP 并不优）
- **AP 特化数据库的探索**
    - 90 年代人们开始建设「数据仓库」进行数据分析
    - 分析型特化的数据库 / 技术涌现：Teradata，Vertica，Greenplum，Apache Hadoop
    - 随着分析用况流行，相关技术不断发展
        - MPP Architecture
        - Columnar Storage
        - Vectorized Engine

一点点历史

- **Michael Stonebraker 祖师爷总结说：「One size fit all」idea has gone**
  - 数据库厂商由于商业和研发代价等原因，让一个数据库用于多种场景
  - 但是越来越多新的场景出现
  - 「特别的爱给特别的你」每个特别的领域都值得拥有一个特别的数据库
  - 作为先驱之一，他自己也主持研发了分析特化的列存数据库 C-Store
    - C-Store 之后变成了 Vertica

AP & TP Database



Icons made by Eucalyp, prettycons from Flaticon

The Rise Of HTAP

- **近些年，随着实时分析需求的增加，TP AP 合流再次被提起**
  - 例如使用内存作为存储规避数据引擎的差异达到一套搞定效果
  - Gartner 给了它一个名字 HTAP（Hybrid Transanctional / Analytical Processing）
- **如果需求可以满足，用户还是更希望用简单的方案**
- **业务上而言，人们希望更快地分析到更新的数据**
  - 风控得以提早预防损失
  - 物流得以更实时进行资源调配
  - 电商得以更快调整促销或实时进行推荐
  - 公共服务得以提供更便捷的流程

PingCAP

TP 和 AP 的设计选择

HTAP 的「技术面分析」

- **TP 与 AP 是很难分清的两个词，业务侧和技术侧会有不尽相同的理解**
- **这里仅仅从技术侧分析 TP 与 AP，而非业务**
  - **Transactional Processing**
    - 高并发，强一致需求
    - 主 / 次索引支持，快速定位少量明细记录
    - 实时写入，删除，更改记录
  - **Analytical Processing**
    - 低并发，每个查询扫描 / 处理大量数据
    - 批次更新而非实时
    - 需要可扩展的能力存储和计算大量历史

传统存储层设计

- **存储设计需求**
  - TP 用法更偏向少量行的访问，且需要支持良好的实时更新
  - AP 用法更偏向少量列的大批量读取，传统上没有实时更新需求
- **存储设计取舍：尽可能让需要访问的数据呆在一起**
  - TP：点查点写优化，让点访问（点读，更新，插入）更快：行存
    - 单行数据连续排列，一次访问读取完整一行
    - 支持点查的索引技术（基于 Key 的排序整理）
  - AP：大批量部分列访问尽可能快：列存
    - 将行数据拆列纵向存储
    - 牺牲了点访问性能（需要更多次 IO）
    - 无需支持点更新，因此有可能无视 Key 排序

行存 vs 列存

Rowstore

id	name	age
0962	Jane	30
7658	John	45
3589	Jim	20
5523	Susan	52

SELECT AVG(age) FROM emp;

Columnstore

id
0962
7658
3589
5523

name
Jane
John
Jim
Susan

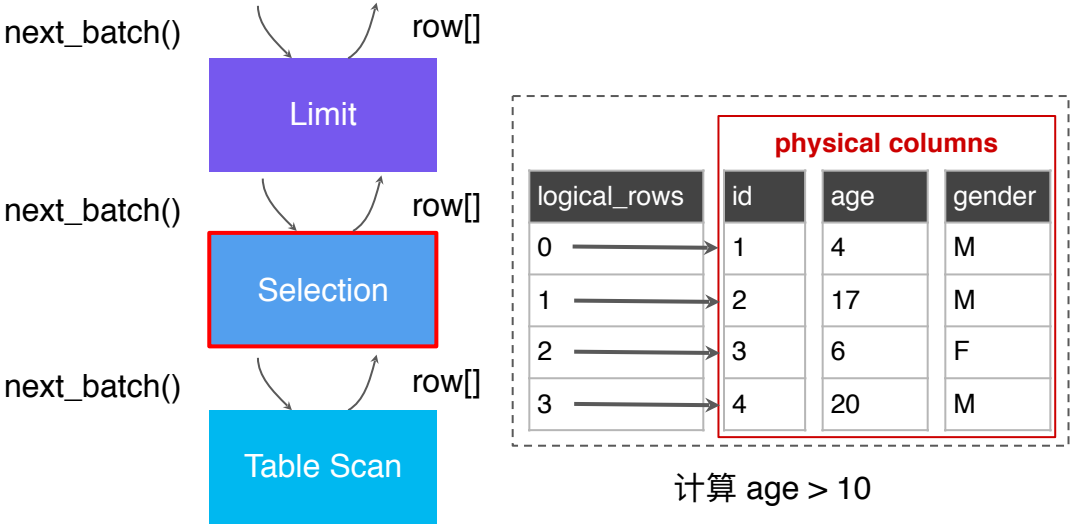
age
30
45
20
52

传统计算层设计

- **计算设计需求**
  - TP
    - 面向少量行的计算优化
      - 传统火山模型，更好的代码可维护性
    - 单机计算引擎，尽可能减小节点交互和调度开销
  - AP
    - 面向大批量数据处理
      - 更长的响应时间，足够插入更重的优化例如 JIT
      - 向量化技术只有这样在大量行均摊分支开销的前提下才成立
      - 单节点无法胜任大量数据计算，标准设计是分布式计算架构例如 MPP 架构

向量化

SELECT * FROM t WHERE age > 10 LIMIT 3



id	age	gender
1	4	M
2	17	M
3	6	F
4	20	M
5	10	F
6	30	F
7	70	M

PingCAP

HTAP 的技术挑战和 TiDB 的应对

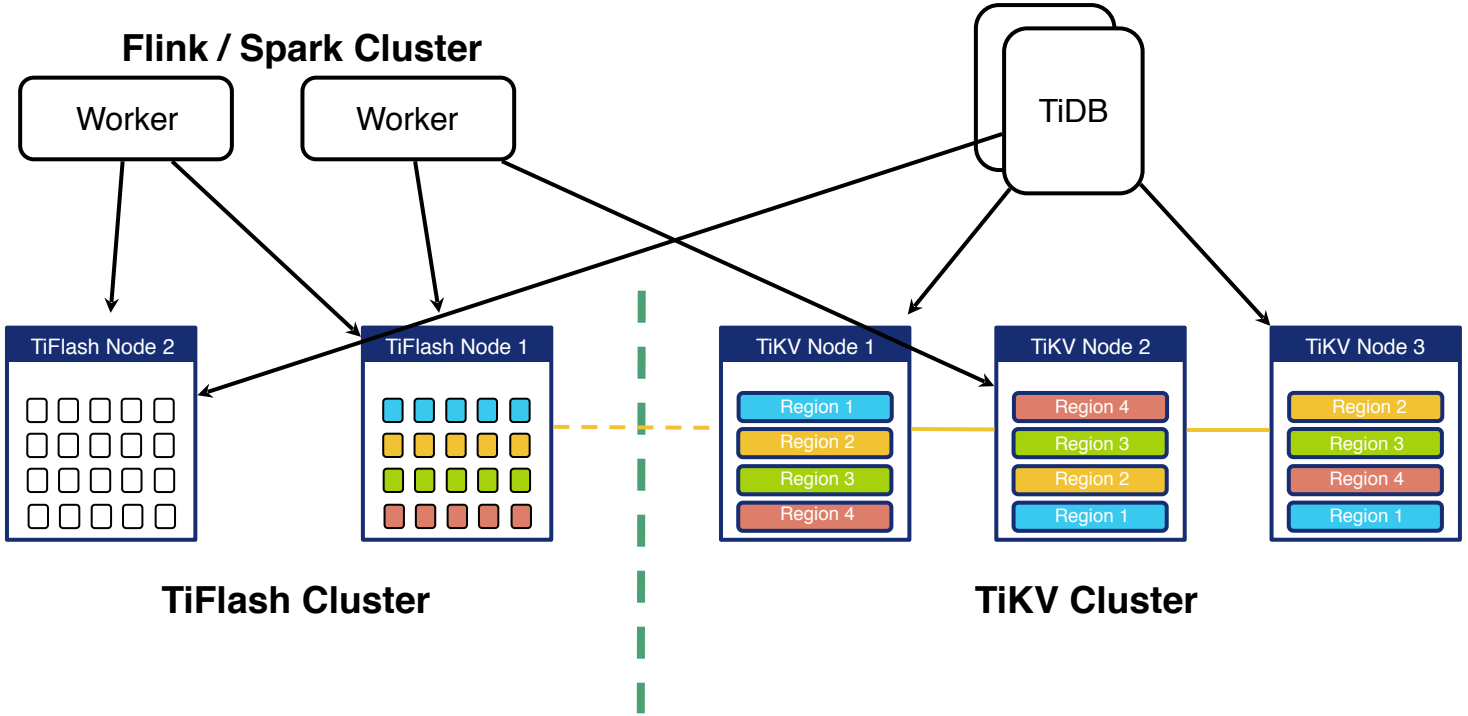
HTAP 的架构挑战

- **架构更复杂**
  - 单引擎适配多场景方案将带来指数上升的技术复杂度
    - 本质上存储的设计设计目标南辕北辙
    - 计算层面也倾向于选取不同的策略
  - 资源隔离和数据一致
    - 资源隔离是必须面对的话题
    - 资源隔离会引入数据同步一致性问题
      - 强一致同步降低 TP 侧稳定性
      - 弱一致同步减少可用场景（例如与钱相关）
    - 对用户入口如何尽量透明

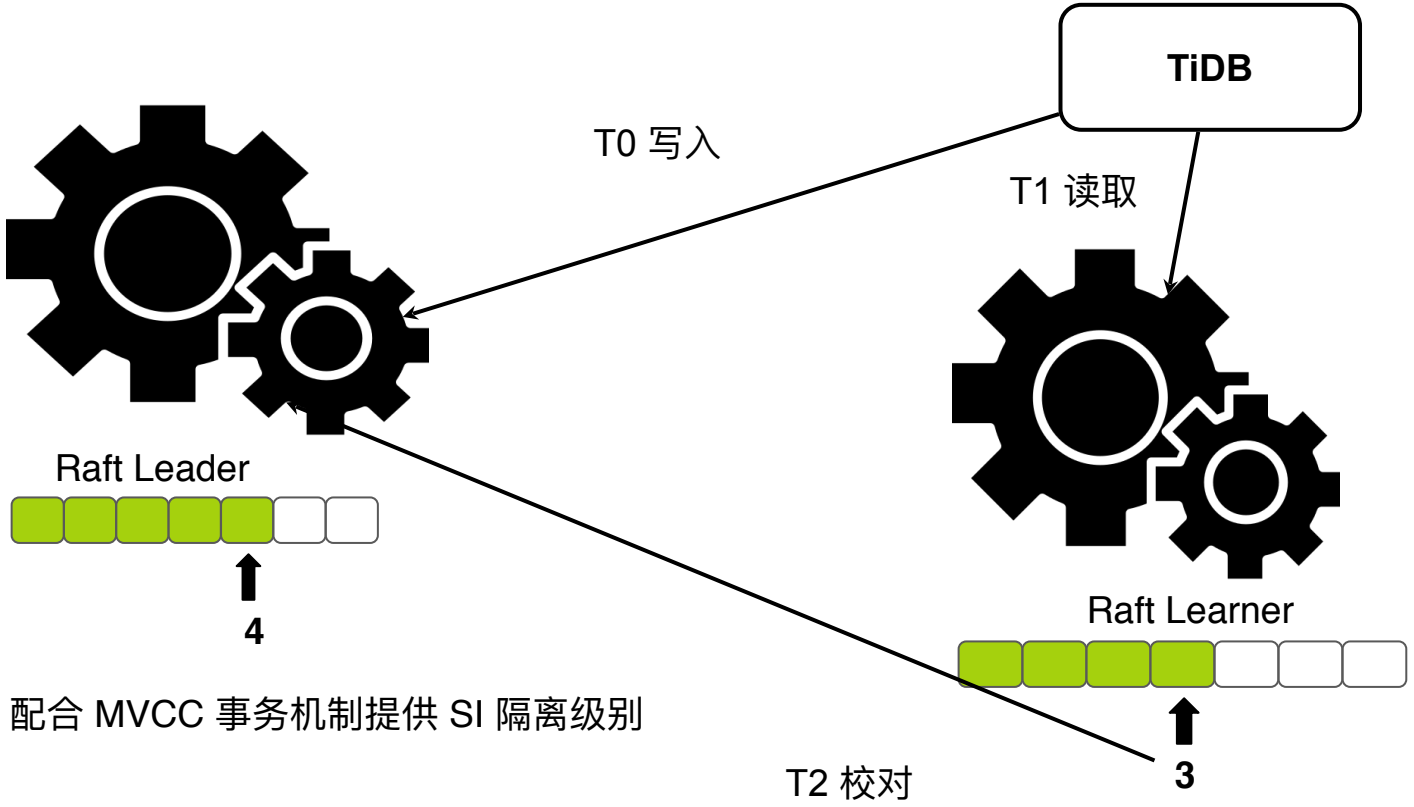
TiDB HTAP 的架构选择

- **AP 和 TP 引擎分离架构**
  - 完备的资源隔离方案
  - AP 查询对 TP 无干扰
- **列存由 Raft 协议创建镜像**
  - 非强一致写但确保强一致读
- **TiDB-Server 作为 HTAP 主入口**
  - 由 CBO 自动选择
  - 或者由人工指定选择存储引擎
  - 配合 Apache Flink / Apache Spark 的深度集成

TiDB HTAP 总体架构

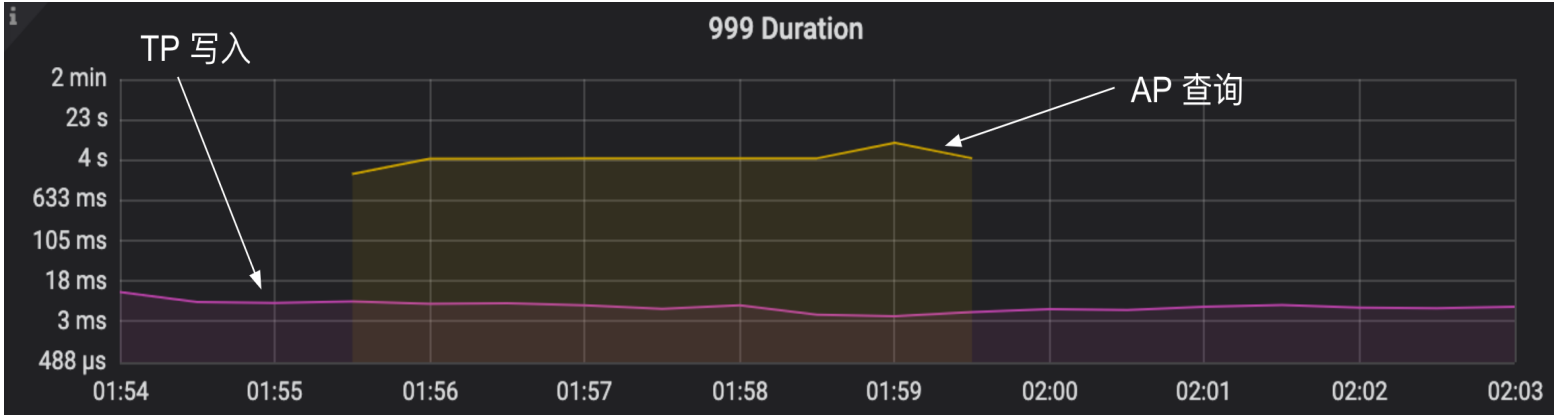


TiDB HTAP 强一致读



配合 MVCC 事务机制提供 SI 隔离级别

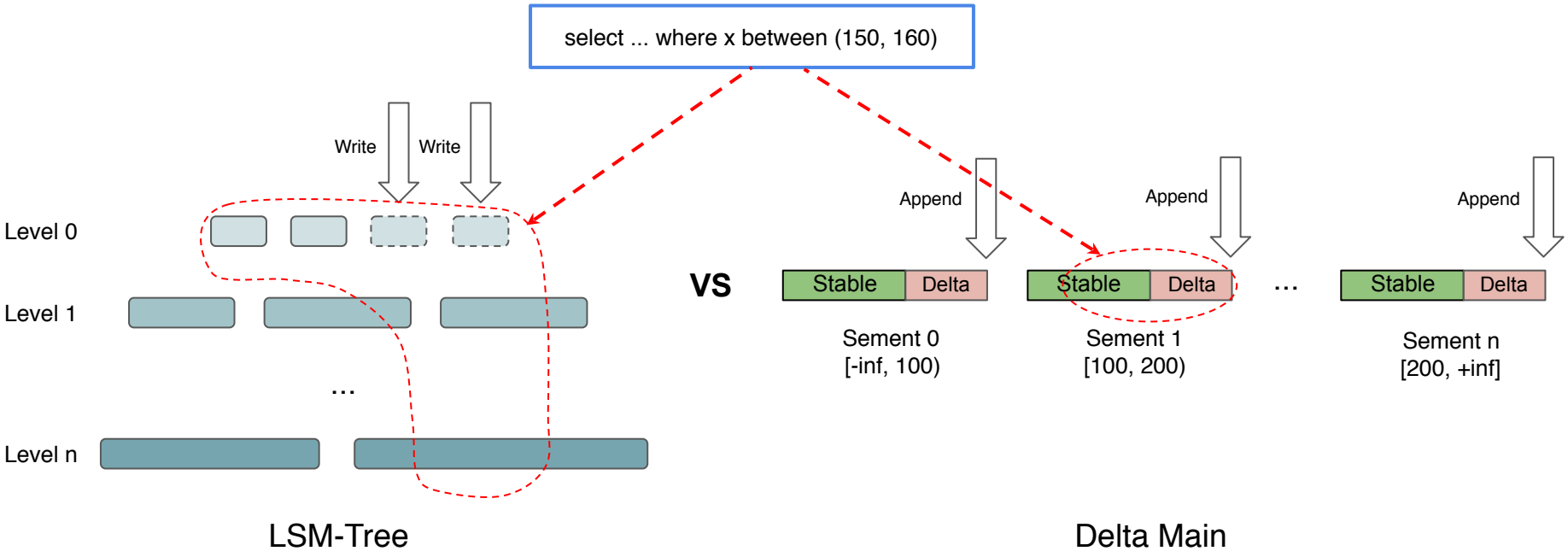
TiDB HTAP 隔离性



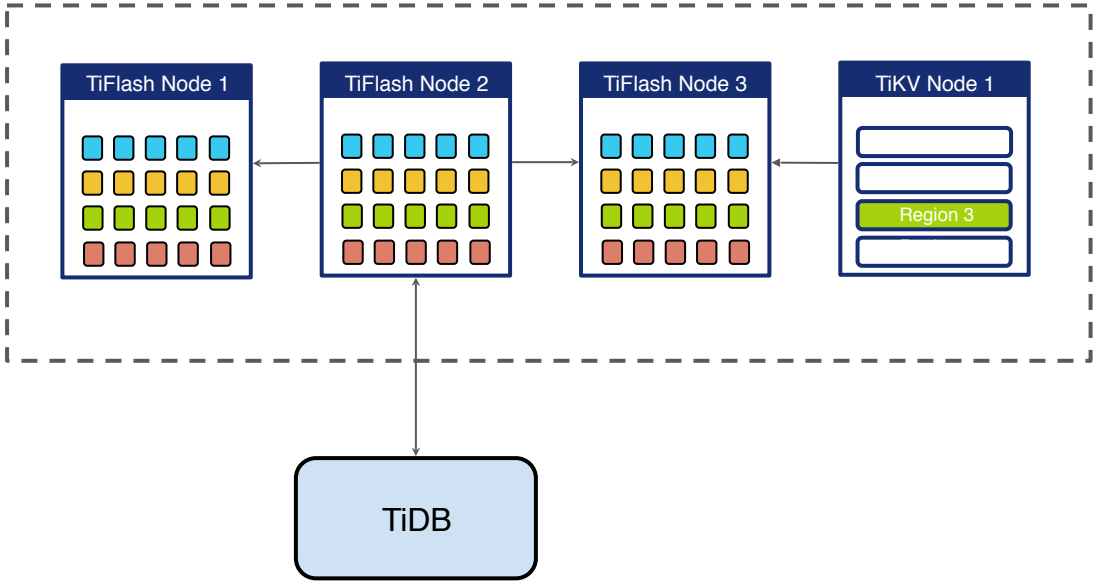
HTAP 存储层挑战

- **列存按主键实时更新的本质**
  - 按主键序进行整理以便更新能被快速定位（无论读时还是写时）
  - 传统手段的 B+Tree 用于完成主键序整理和更新
    - 对于列存而言，写入 IO 离散度巨大（拆列）
- **选取一个数据结构 + 列存**
  - 支持攒批（而非 In-Place Update）以抵抗列存带来的 IO 离散
    - 几乎所有可更新列存设计都基于攒批
  - 虽然攒批，仍能提供有效的整理以便对同一个主键新老数据版本选择
    - 攒批必然带来延迟整理
    - 所以你想到了什么：LSM。
      - 未尝不可，不过我们选了另一个设计

TiDB 的可更新列存设计



列存直接写入和容错（2021）



- TiDB 新版将允许列存引擎独立接受写入
- 适用于无需明细查询的场景
- 提供 Upsert 语义，允许按照主键进行更新
- 和 TiKV 一样的 Region 分区，通过 PD 进行调度
- 无需经过 TiKV，更少的资源用量，更高的吞吐

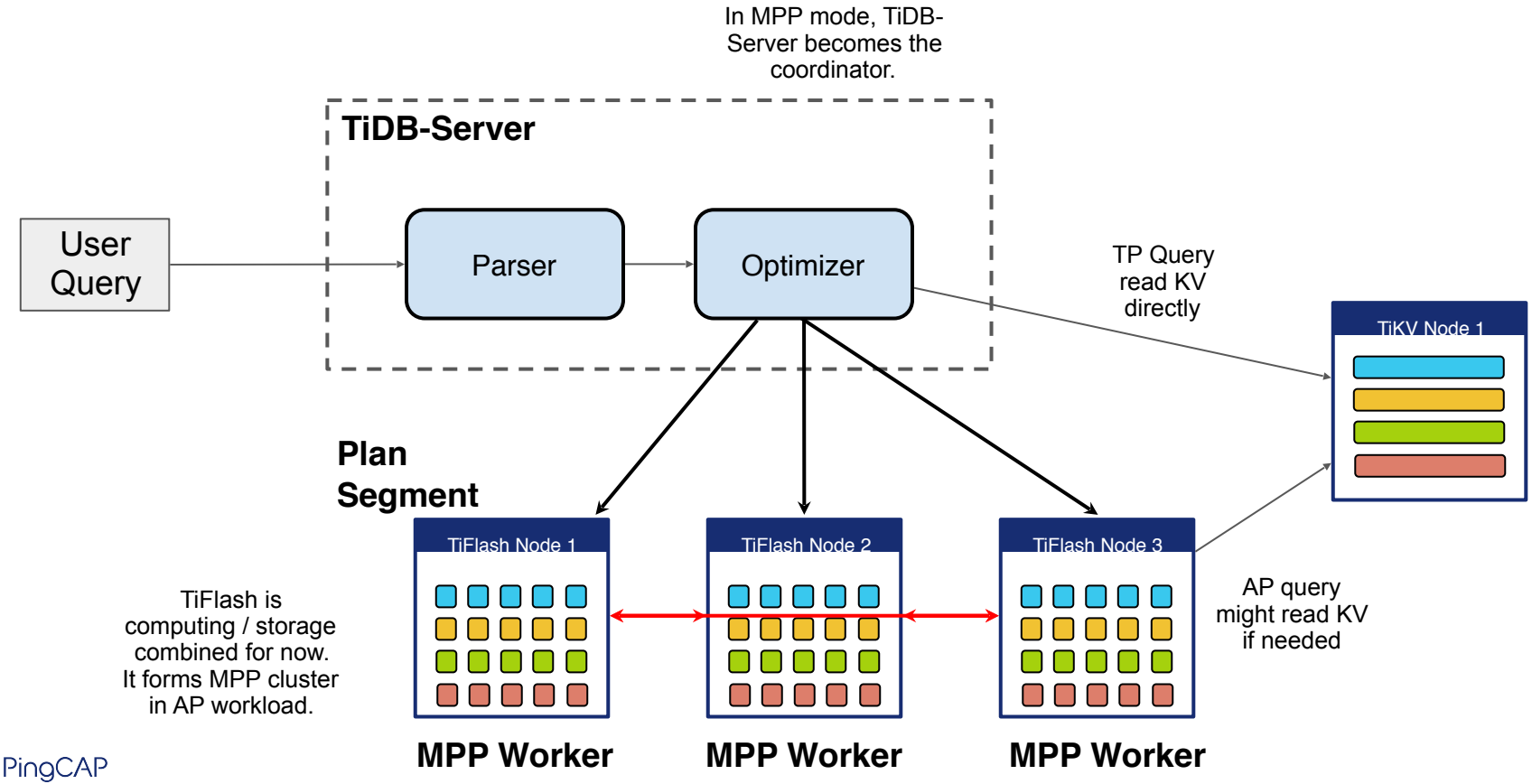
HTAP 计算层挑战

- **TP 类单笔事务数据量小**
  - 单机完全胜任，多机分布式计算反而引入大量协同和调度开销
- **AP 类单笔事务数据量大**
  - 需要完整的分布式计算框架
  - 很多 HTAP 产品借助外部大数据引擎完成 AP 查询（包括 TiDB 在 4.0 之前）
    - 用户是 DBA 有时候较为抗拒大数据组件

TiDB HTAP 计算层

- 双模单入口
  - 处理 TP 作业时，TiDB 使用单机引擎
  - 处理 AP 作业时，TiDB 使用相同入口但由 TiFlash 组成 MPP 引擎执行
  - 共享优化器，让用户允许时，行存列存配合暴露更多优化选择
  - 配合列存体系，使用向量化引擎
    - TiDB-Server 引擎，TiKV 协处理器，以及 TiFlash 的 MPP 引擎（部分借助 Clickhouse）

MPP 架构（2021 Q1）

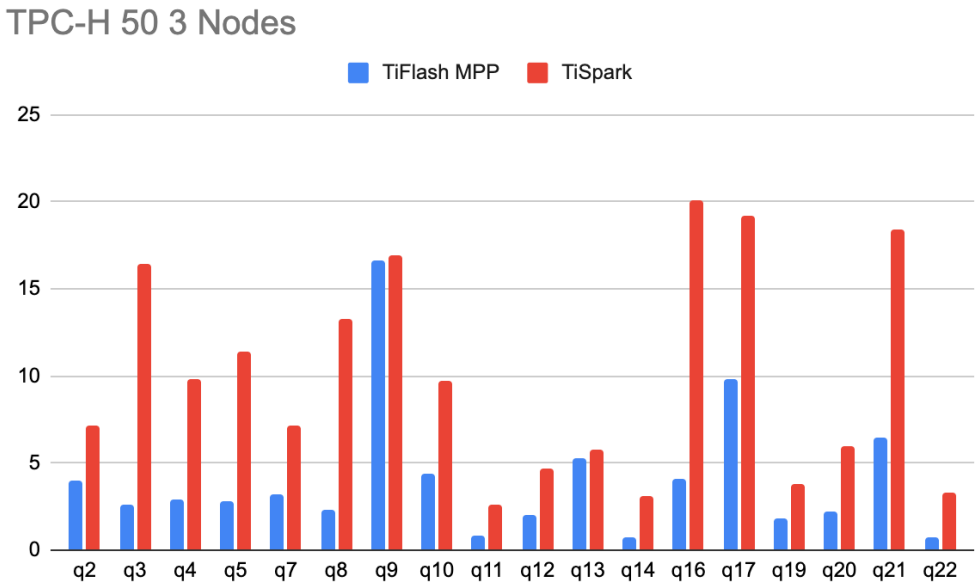


PingCAP

MPP 架构

开发中的部分 Benchmark，仍在优化中，MPP 功能将随 5.0 GA 正式发布（2021 年 3 月）

	TiFlash MPP	TiSpark
q2	3.97	7.151
q3	2.59	16.463
q4	2.86	9.8
q5	2.79	11.395
q7	3.25	7.128
q8	2.32	13.243
q9	16.62	16.949
q10	4.38	9.728
q11	0.83	2.655
q12	2.05	4.706
q13	5.23	5.76
q14	0.73	3.077
q16	4.05	20.08
q17	9.82	19.215
q19	1.81	3.814
q20	2.17	5.958
q21	6.5	18.424
q22	0.72	3.28



PingCAP

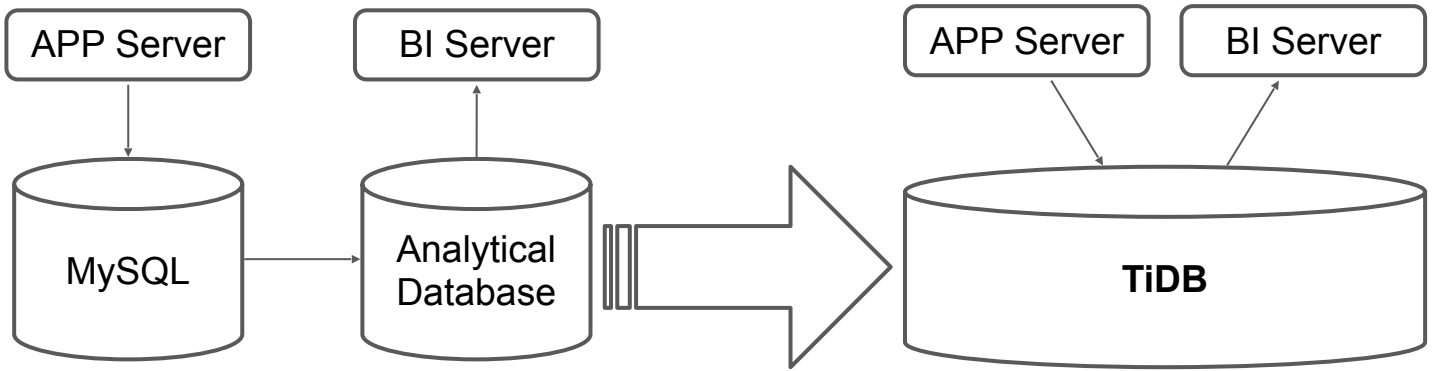
TiDB HTAP 用况

TiDB HTAP 用况

- **主流用法**
  - 使用 TiDB TP 部分作为交易库，并使用 AP 部分进行实时交易数据的查询
  - 利用 TiDB HTAP + 可扩展能力进行实时分析
    - Single View：汇聚不同业务线数据进行实时分析和服务
  - 利用 TiDB 的和大数据层整合
    - 作为离线数仓之前的实时数仓
  - 直接针对流计算数据的全能 Sink 方案
    - 可对接 CDC 类数据库变更流（Update 支持）
    - 提供行存列存双引擎（明细查询或者报表 / Ad-Hoc 查询）
    - 次级索引支持，SQL 接口，可扩展

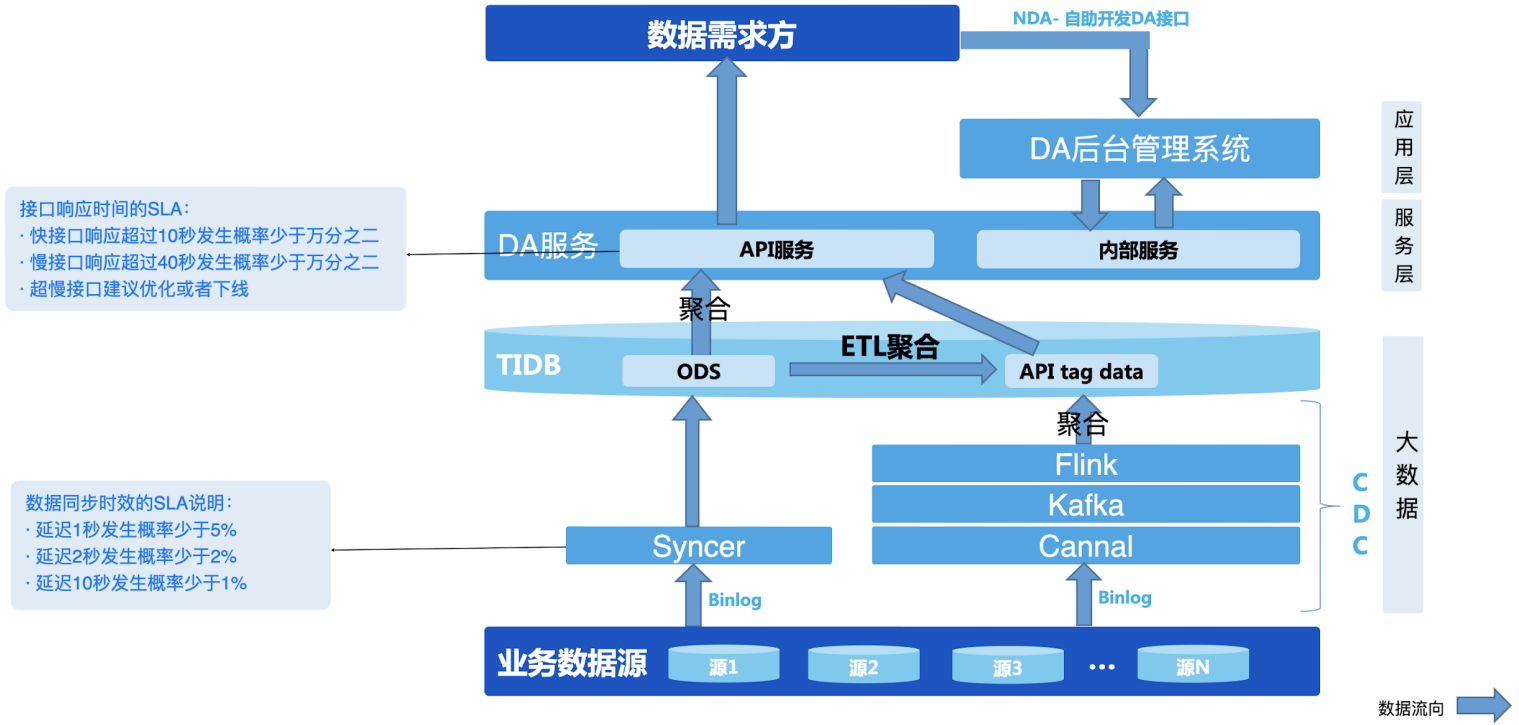
TP + AP 一体化

交易 + 分析一体架构简化

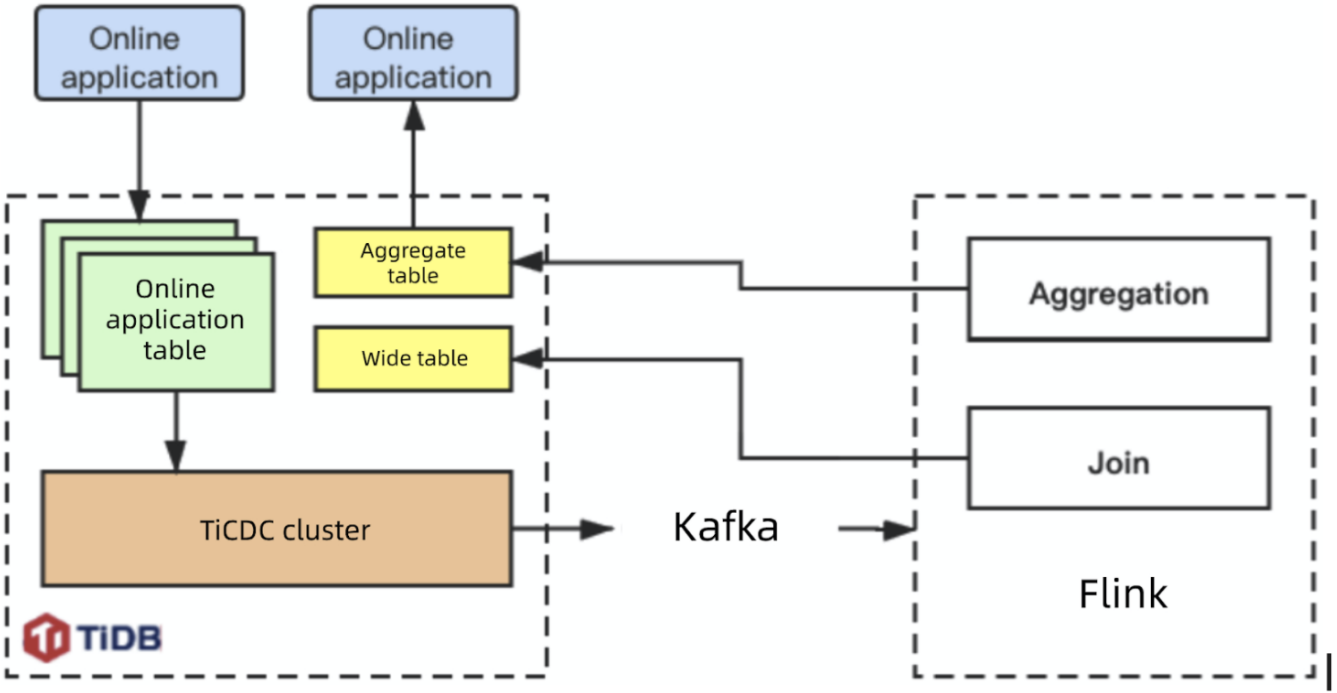


实时数仓

DA平台架构


数据需求方
NDA- 自助开发DA接口
DA后台管理系统
应用层
服务层
接口响应时间的SLA：
· 快接口响应超过10秒发生概率少于万分之二
· 慢接口响应超过40秒发生概率少于万分之二
· 超慢接口建议优化或者下线
DA服务
API服务
内部服务
聚合
TIDB
ODS
ETL聚合
API tag data
聚合
大数据
Flink
Kafka
Cannal
CDC
数据同步时效的SLA说明：
· 延迟1秒发生概率少于5%
· 延迟2秒发生概率少于2%
· 延迟10秒发生概率少于1%
Syncer
Binlog
Binlog
业务数据源
源1
源2
源3
...
源N
数据流向

实时分析


Online application
Online application
Online application table
Aggregate table
Wide table
TiCDC cluster
TiDB
Kafka
Aggregation
Join
Flink

PingCAP

总结 / 展望

总结 / 展望

- TiDB HTAP 并不是并不是简单的 1 + 1
  - 并非简单的 TP 或者 AP，而是同时有
  - 为业务带来简化架构以及实时性
- 分布式 + Raft 复制让我们有机会以更松的耦合，更低的技术难度设计 HTAP
  - 更好隔离性，同时兼顾一致性
- 结合 Apache Flink 等生态，从 TP 向实时 AP 更深入推进
- 云环境下能让这套组合架构更有生命力
- 数据库本意是为解放用户心智而诞生的软件
  - 合久必分，而分久也必合，这才符合用户期望

Thank you!