

Tencent 腾讯

WeOLAP: 微信基于ClickHouse内核的实时数仓实践

Lucas 微信-技术架构部

2022.07.02



目录

- 微信海量数据场景下的挑战
- 发展历程-典型应用场景举例
- 探索期-WeOLAP生态
- 增长期-核心技术解决
- 突破期-迈向云原生数仓
- 未来展望

1 背景-数据分析场景



交互分析

万亿级，秒级响应

Ad-hoc 数据查询，场景维度多、追溯层级深、数据随机性强，维度与条件可灵活选择，即席看到结果，效果分析对比，根因定位



A/B实验

实时关联查询、高纬度

实验命中数据与业务指标做关联，海量数据下关联查询，数据准确性要求高，分析数据大，涉及回溯历史数据



实时计算

低延迟，高并发

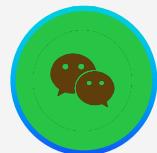
实时特征计算，异常监控，数据链路时延性要求高，查询低时耗，请求QPS大



明细查询

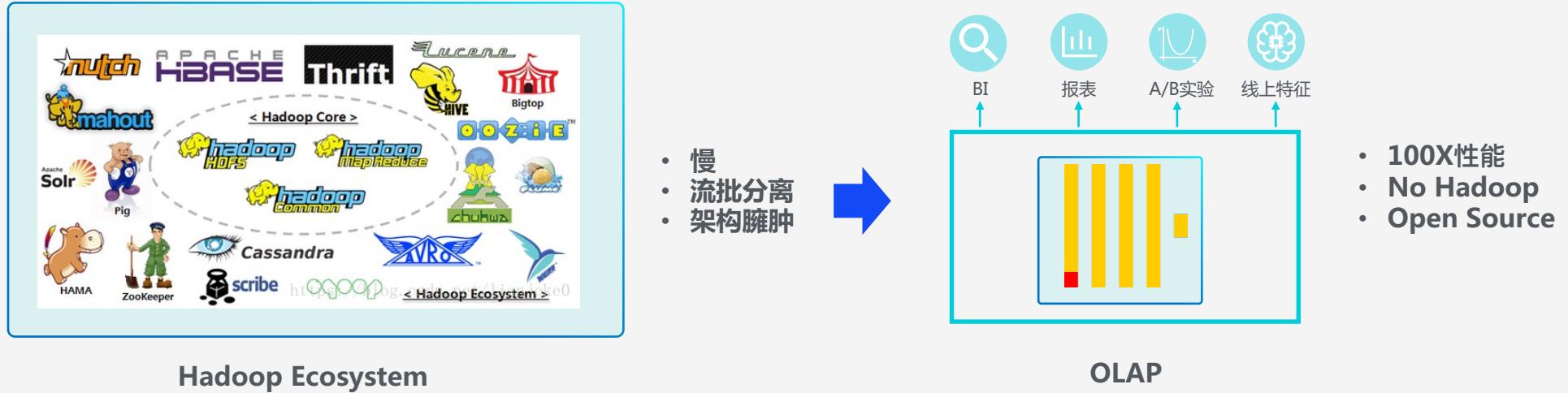
查明细，准确性高

推荐链路Debug与还原工具，快速查看，效果归因，问题定位



1 Hadoop数仓下的困境

视频号等推荐系统的对个性化体验强烈诉求，催生了“亚秒级”分析系统的诞生



设计目标：

- **亚秒级响应**：亿行数据亚秒响应、万亿行数据秒级返回，支持A/B实验平台，BI分析等复杂指标计算场景，可追踪明细；
- **海量数据低时延**：百亿级吞吐下的秒级接入时延，满足实时指标统计、异常检测需求；
- **极简架构**：批流一体，T+1、Realtime分析模型统一，简单易运维；
- **高可用**：数据接入有精确一次保障，单集群99.99% SLA。

在**OLAP**场景构建**Clickhouse**计算存储为核心**批流一体**数仓

1 集群现状

规模 **15W** 核+

响应 **0.34 s** (TP50)

数据百 **PB** 级

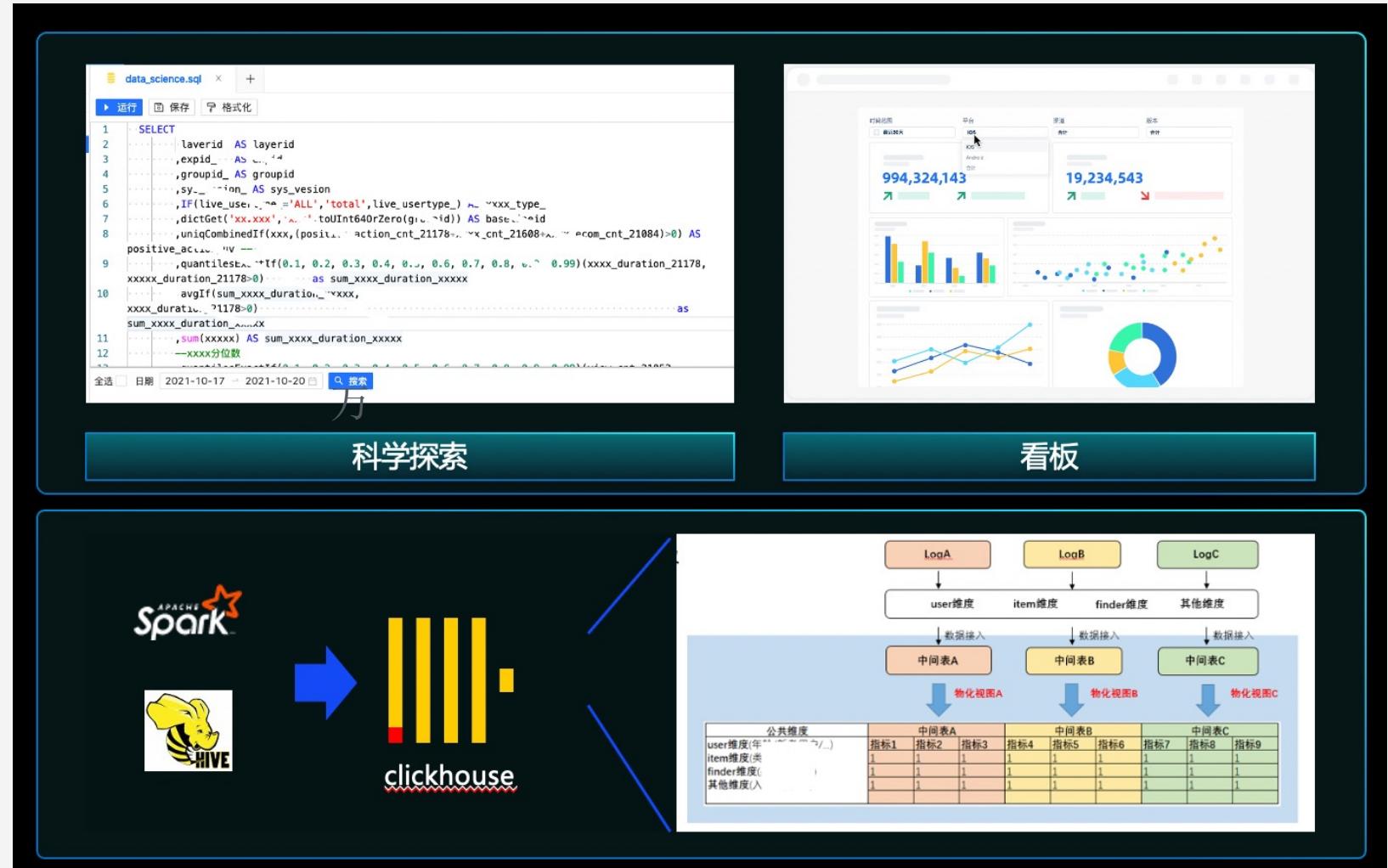
查询 **1M + /天**

TPS **亿** 级(单集群)

日单次分布式查询平均耗时(秒)	日单次分布式查询耗时TP50	日单次分布式查询耗时TP90
0.4723	0.062	0.3789
0.4038	0.084	0.649
N/A	N/A	N/A
3.19	0.486	6.81
0.1611	0.034	0.168
0.0452	0.028	0.0811
3.7	1.24	10.19
3.63	0.273	4.82
5.09	0.022	4.63
3.1	0.253	4.25
0.2215	0.079	0.3604
7.64	0.143	7.53
10.92	9.02	20.05
0.0441	0.034	0.0752

2 典型应用场景-BI用户分析

单表 万亿
小时 → 5 < s
流批 一体



2 典型应用场景-A/B实验平台

场景

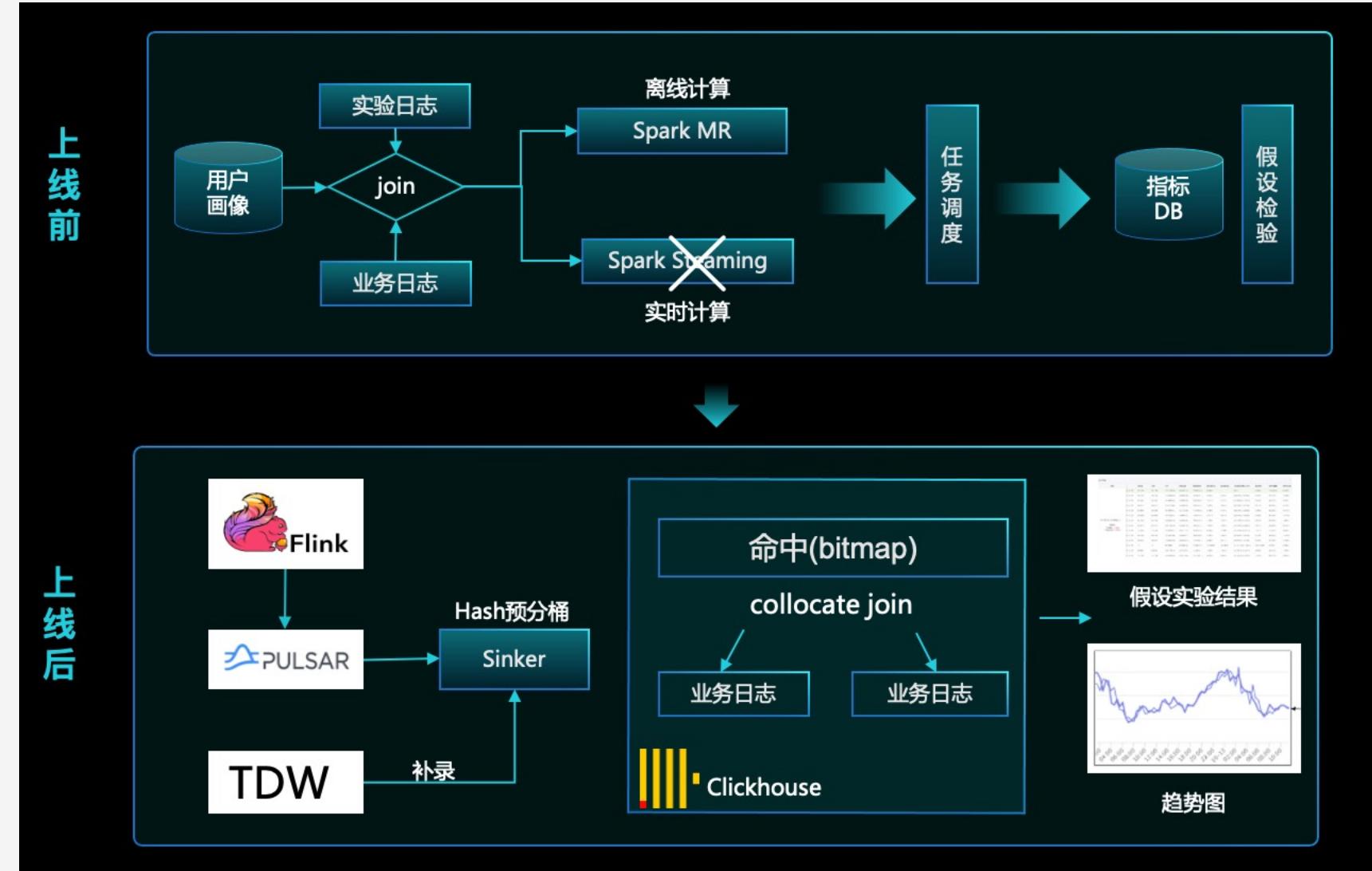
- 大表实时Join场景，单表数据量级 **千亿 / 天**

历程

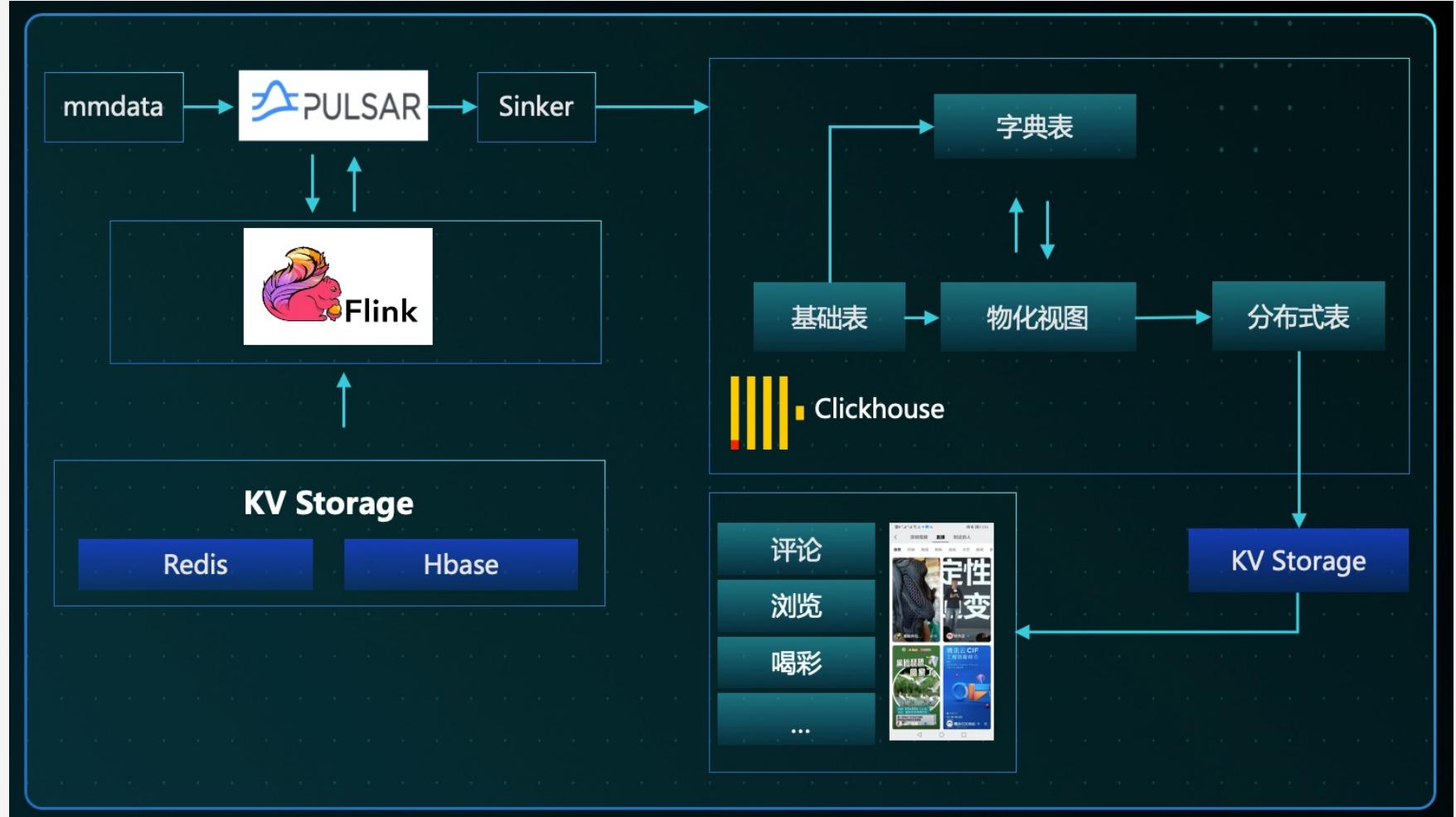
- CK本身方案逐步演进，近 **50X** 性能提升，
- 离线到实时

收益

- 实验结论**更准确**，周期**更短**，模型验证快
- P95响应：**<3S**



典型应用场景-实时特征计算



2 海量数据下的挑战

稳定性差

- ZK 瓶颈 (PR)
- 频繁OOM
- Too Many Part (PR)
- 慢查询拖死
- DDL/TTL卡死 (PR)
- 节点重启慢
- 数据不一致
- 坏盘频繁
- 物化视图误用

海量接入难

- 百亿吞吐
- 流量洪峰
- 精确一次
- 弹性伸缩

大规模集群运维

- 亚健康状态感知
- 扩容问题
- 运营体系建设
- 数据搬迁
- 监控报警
- SLA指标

用不好的CK没有那么快 ?

2 发展历程

验证期



小规模试用

- ZK高负载
- 频繁OOM
- DDL卡死
-

2020年底

增长期



建设周边生态， 业务发展迅速

- 生态建设(sinker,op-manager,qs,monitor)
- 内核优化，发布微信版本
- 查询优化，经验沉淀到平台
-

2021.03~2021.10

突破期



存算分离 云原生数仓

- 秒级弹性扩容
- 远程存储读取加速
- Bitbooster bitmap加速
- 精确一次
- 剔除ZK依赖

2021.8 ~ now

未来



数据湖生态兼容

- MPP Join优化
- 湖仓一体生态兼容

~

3 OLAP 生态介绍

QueryServer

- 数据网关，智能缓存，
大查询拦截，限流

Sinker

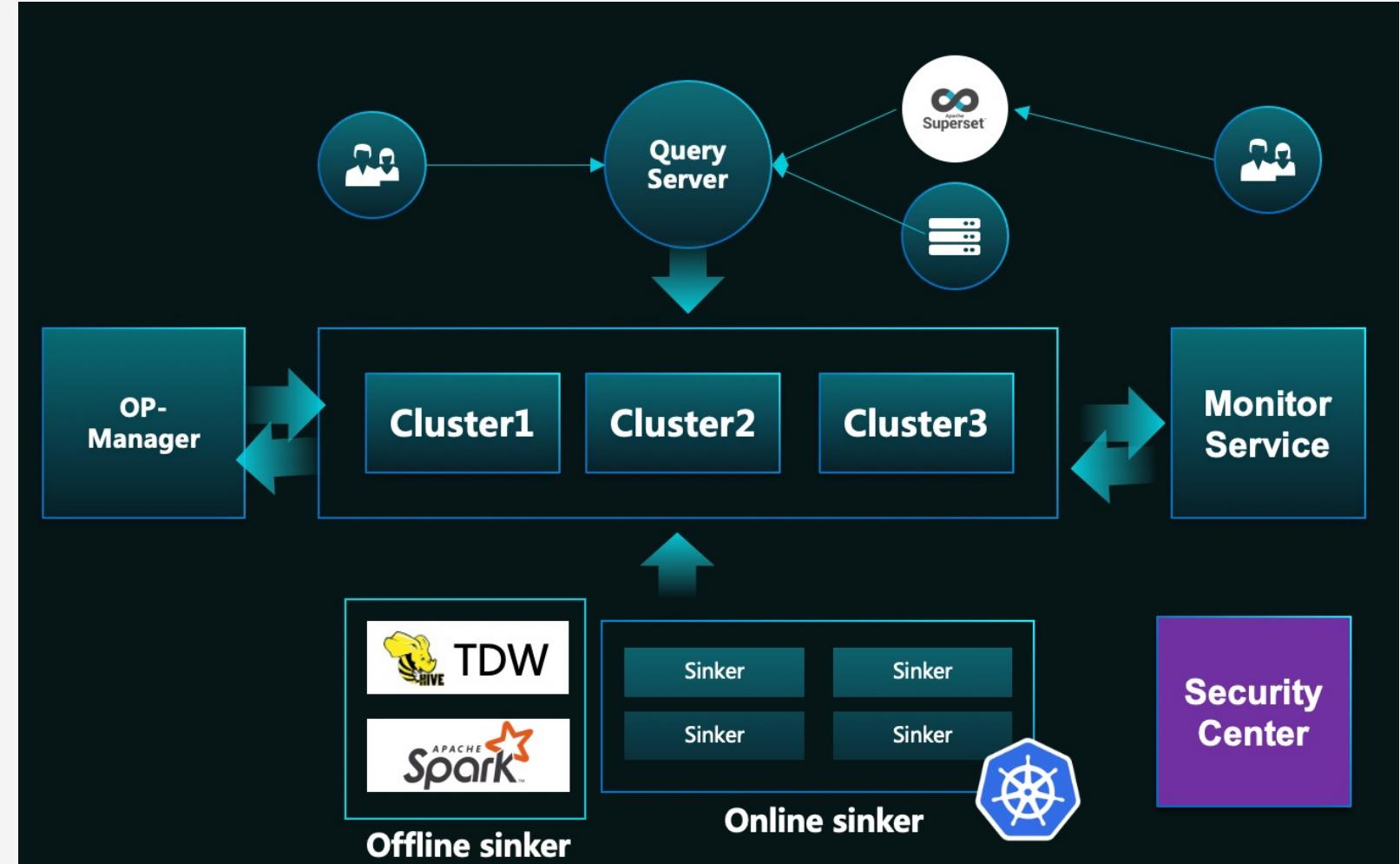
- 离线/在线高性能接入层
(消峰、hash路由，限流)

OP-Manager

- 集群管理，数据均衡
- 容灾切换，数据迁移

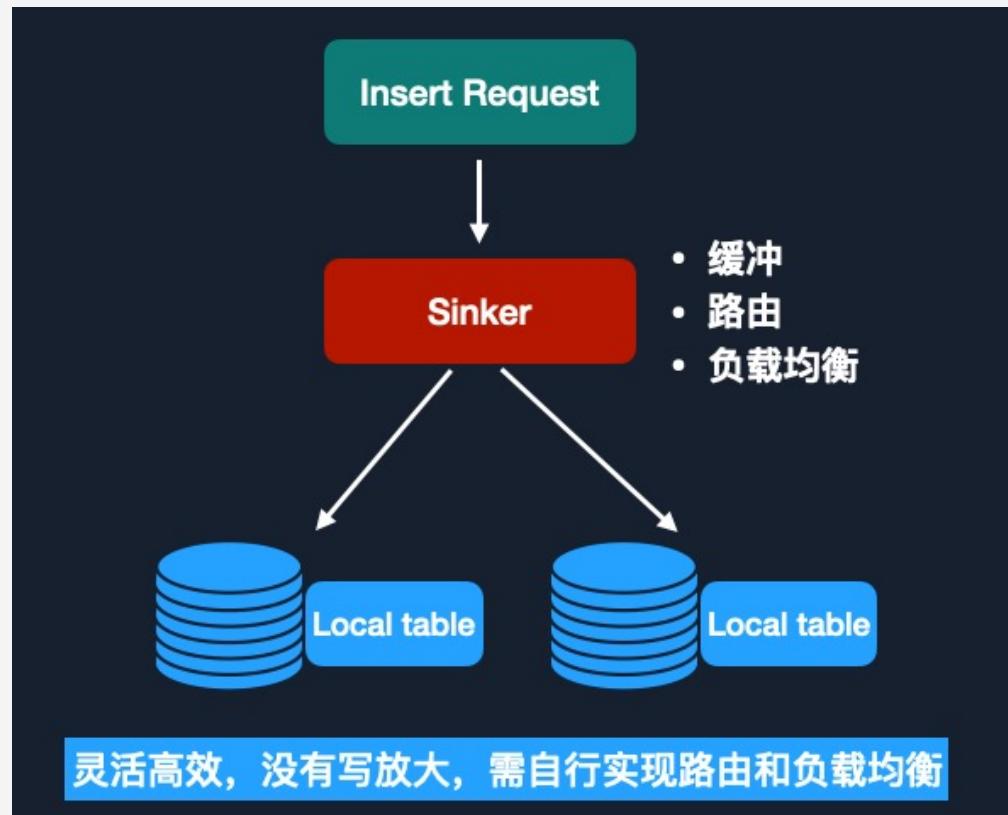
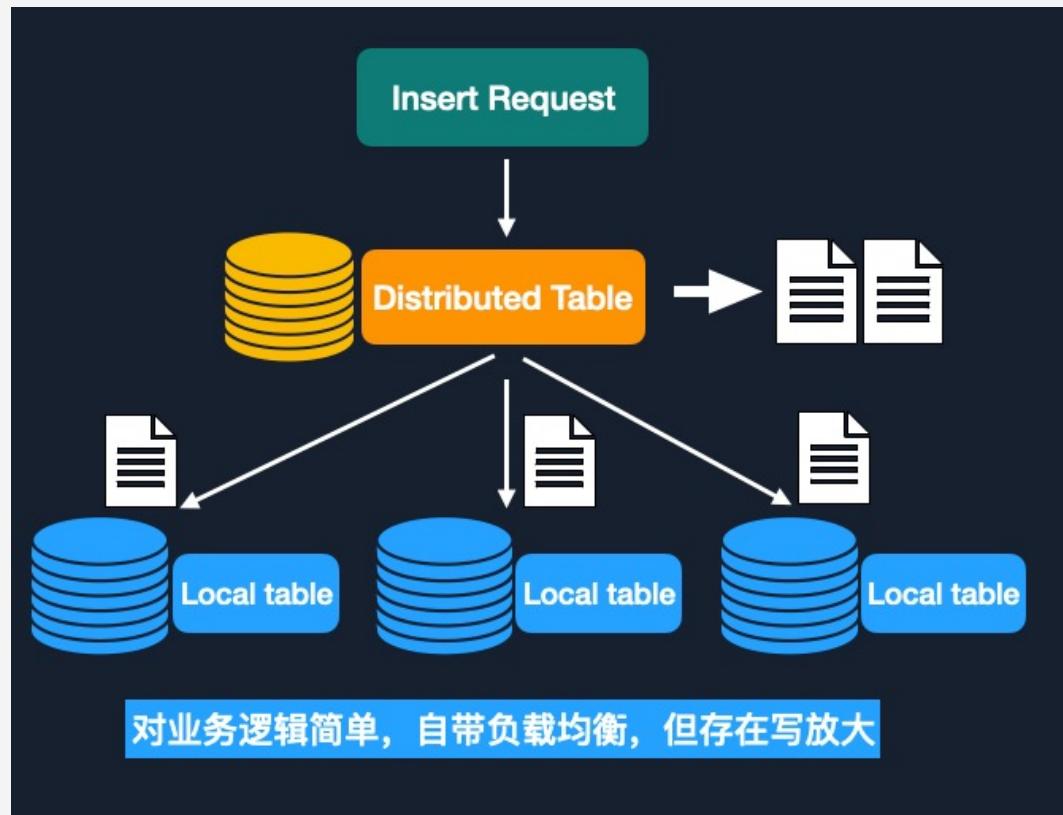
Monitor

- 策略监控报警，亚健康
检测，查询健康度分析，
可与Manager联动

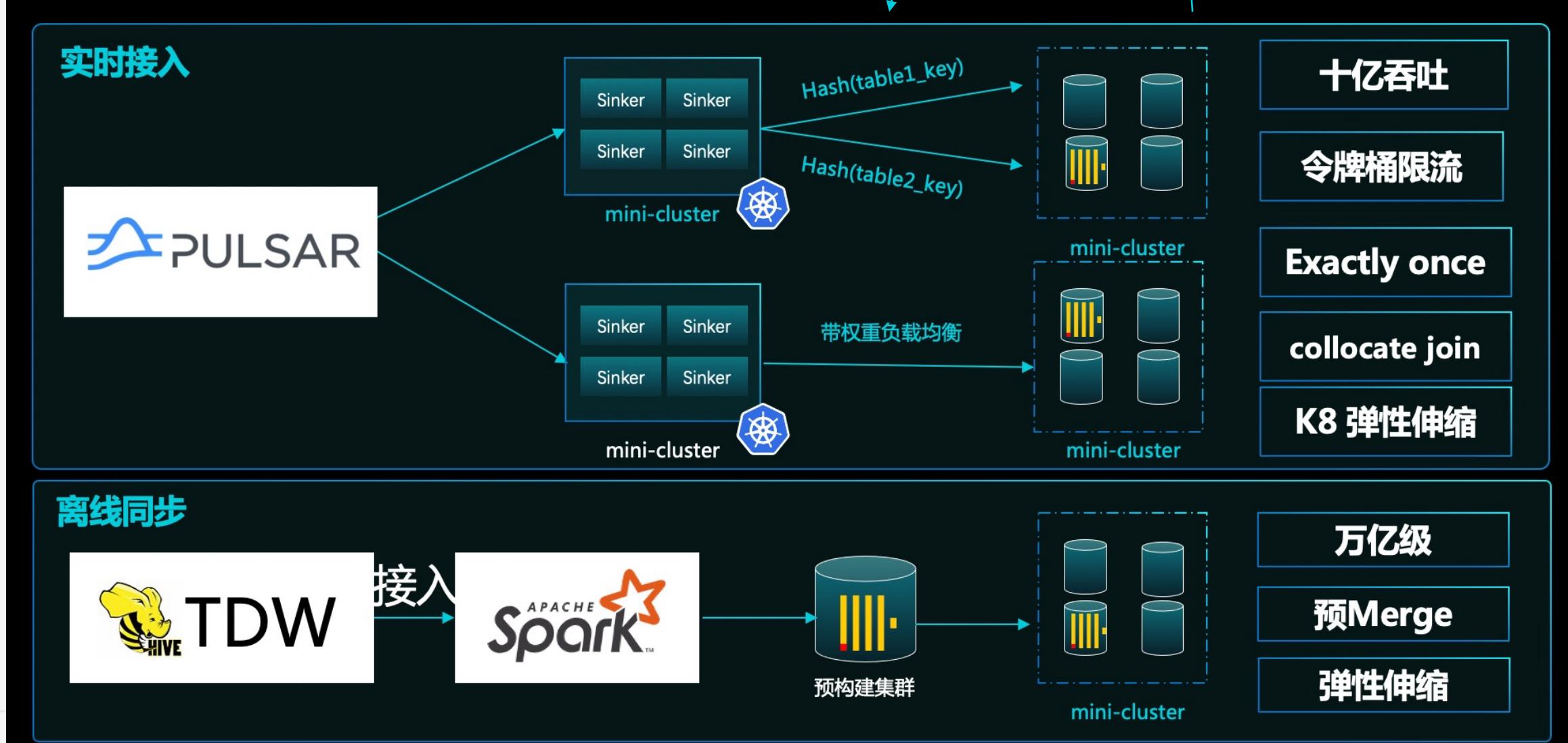
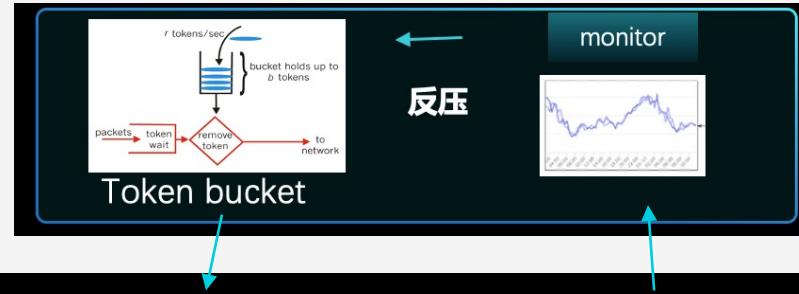


海量接入能力

Kafka接入/写分布式表/写本地表



3 海量接入能力



3 内核稳定性优化

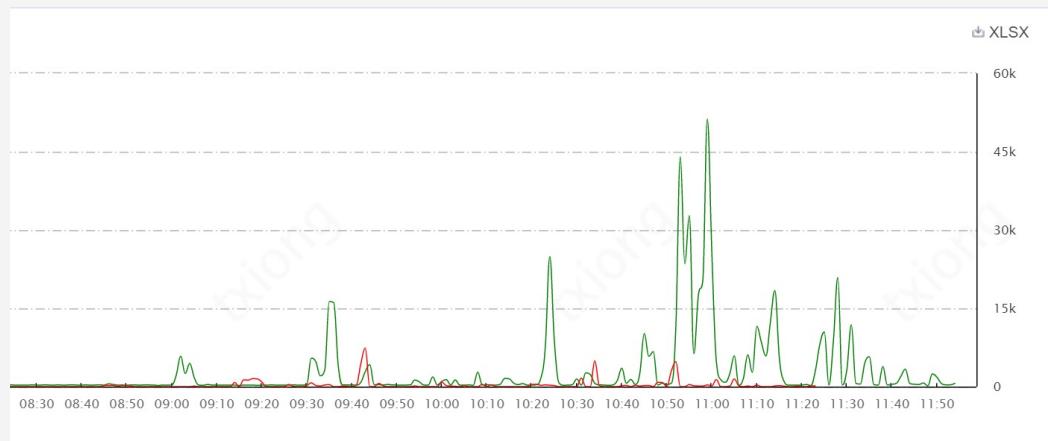


查询优化

直播、视频号等多个Case近10X性能提升

Fisher-BI: 查询优化经验沉淀平台,降低用户门槛

The screenshot shows a '基础信息' (Basic Information) section with dropdowns for '项目' (Project), '数据源' (Data Source), '库名' (Database Name), '表名' (Table Name), '表中文名' (Table Chinese Name), '公开' (Public), '敏感等级' (Sensitivity Level), and a '备注' (Remarks) text area. Below this is a '表字段' (Table Fields) section where users can define field names, Chinese names, types, and other properties for multiple columns.



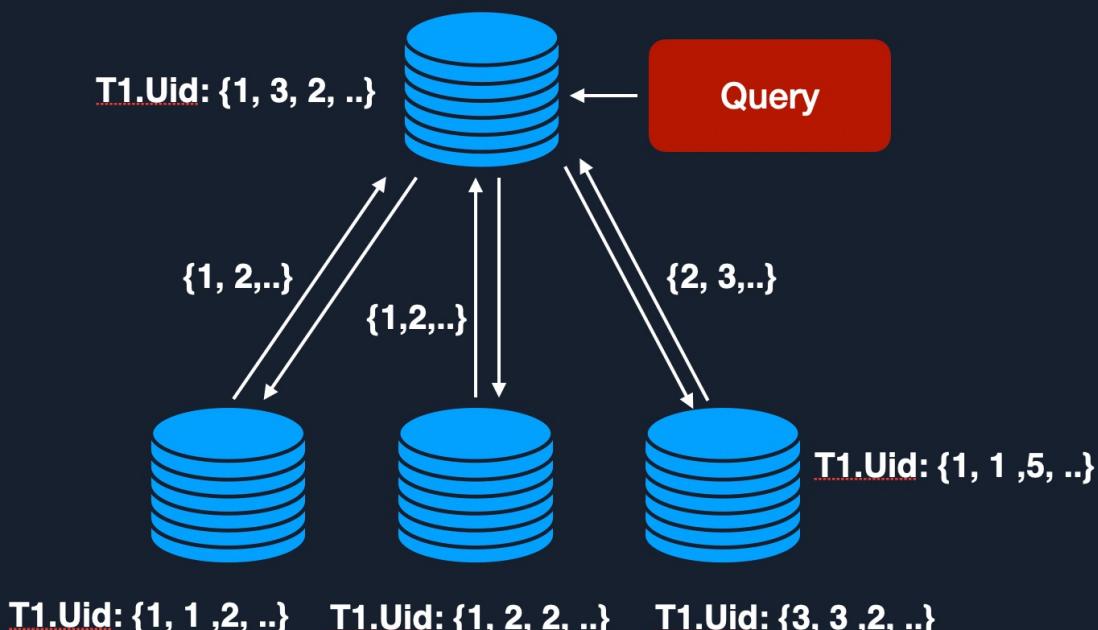
绿色为优化策略上线前，红色为上线后，响应时延控制在4s内

接入	写本地表，路由 hash接入
代理层	缓存、拦截
查询优化	函数优化(array join,近似计算)、Map、JsonObject、Bitmap
逻辑层	物化视图、字典编码、Collocate join LowCardinality、Projection、explain cost
引擎层	合理表引擎 合理索引(基数小) 主键/排序键拆分 二级索引
存储层	字段类型精简 合理分区(尽量少) 合理压缩(lz4,zxtd1)
硬件	冷热分离 Raid0 多盘并行(5GB/s) SSD

查询优化举例 (colocation join)

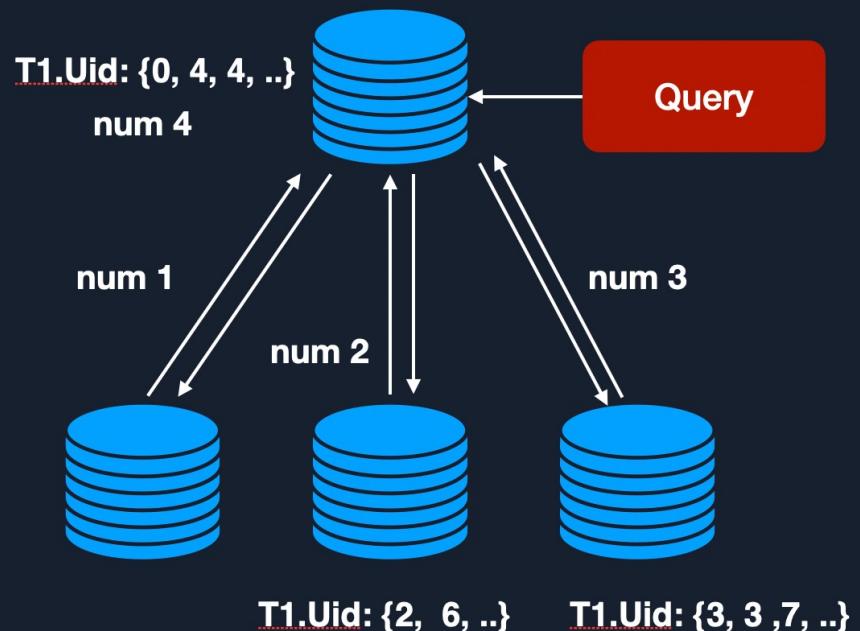
案例：对表T1，计算PV数据，当不采用 Hash 路由时，
Uid 在 shard 间随机分布

`SELECT count(distinct uid) FROM T1`



Uid 以亿计，需要传输数G数据！

案例：对表T1，计算PV数据，写入时按照 $Uid \% 4$ 进行路由，
相同的 Uid 将保存到相同的 shard



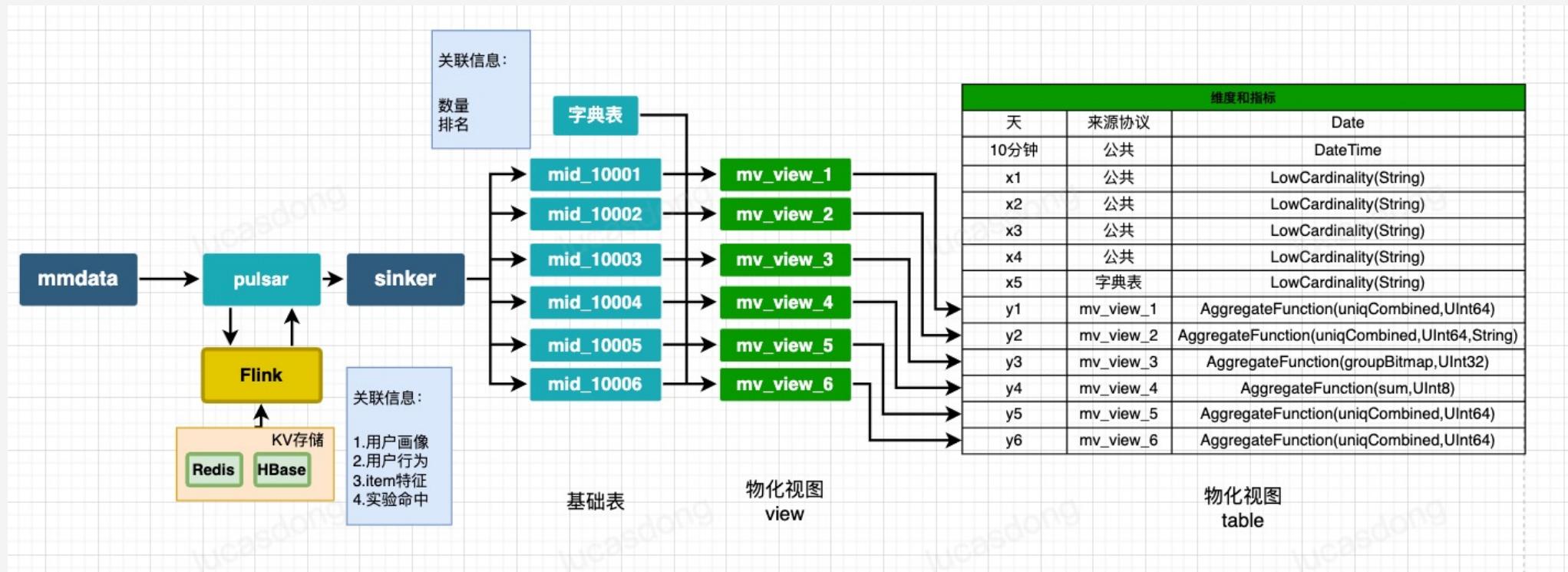
可避免传输中间结果，性能提升5倍以上

通过colocation join加速

3 查询优化举例(物化视图模型改写)

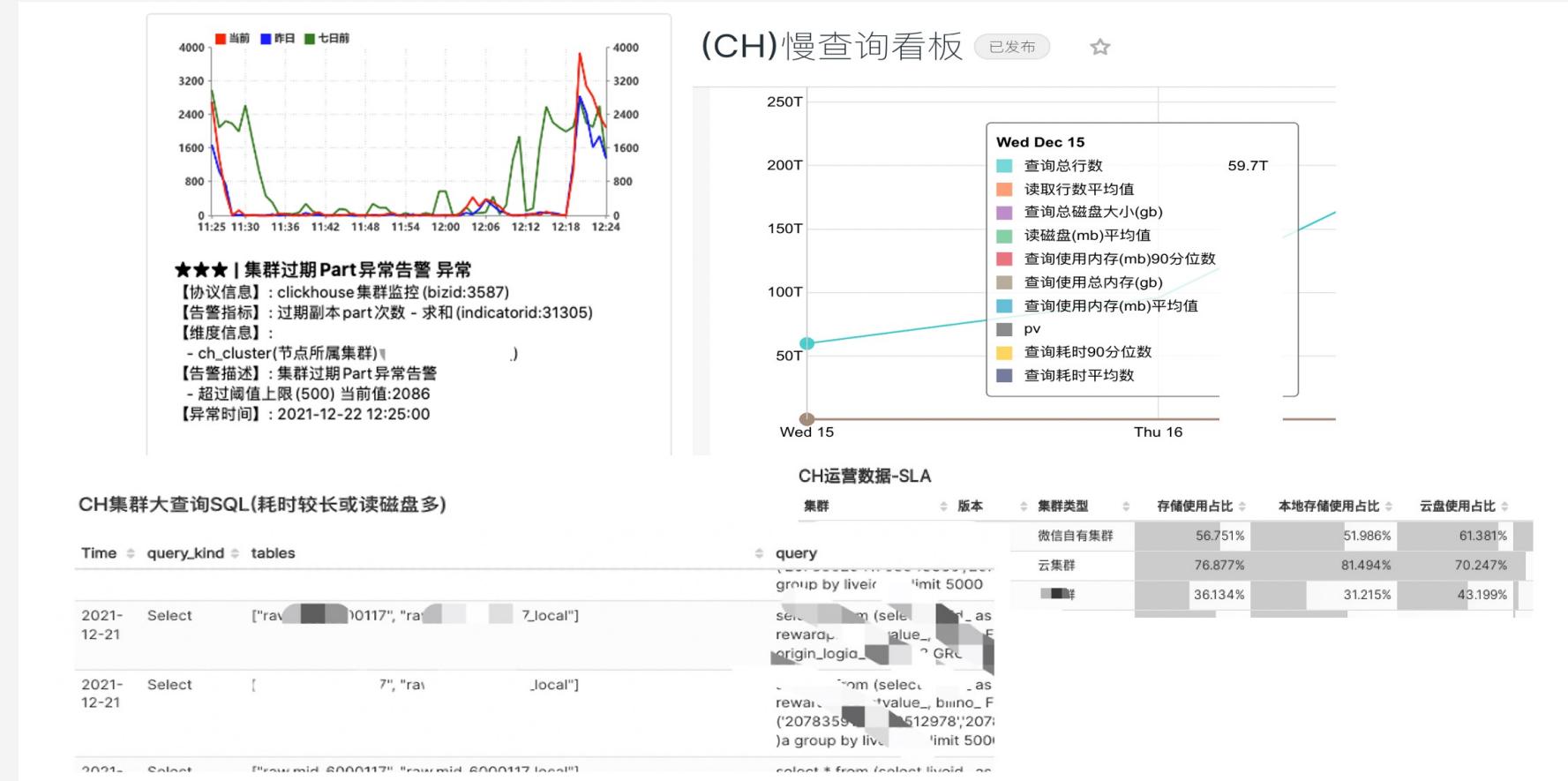
实时物化视图组合宽表：解决该场景下的join问题，避免了业务同学写上千行代码去处理多流指标聚合的问题，开发效率高，可维护性强。

避开join，单表查询，查询效率更高（union ALL 替代Join模型可用）



3 大规模集群-可运营性提升

亚健康监控
慢查询看板
数据均衡
全链路监控



4 发展历程

验证期



小规模试用

- ZK高负载
- 频繁OOM
- DDL卡死
-

2020年底

增长期



建设周边生态 业务发展迅速

- 生态建设(sinker,op-manager,qs,monitor)
- 内核优化，发布微信版本
- 查询优化，经验沉淀到平台
-

2021.03~2021.10

突破期



存算分离 云原生数仓

- 秒级弹性扩容，剔除ZK
- 远程存储读取提速
- Bitbooster bitmap加速
- 精确一次
- ...

2021.8 ~ now

未来



数据湖生态兼容

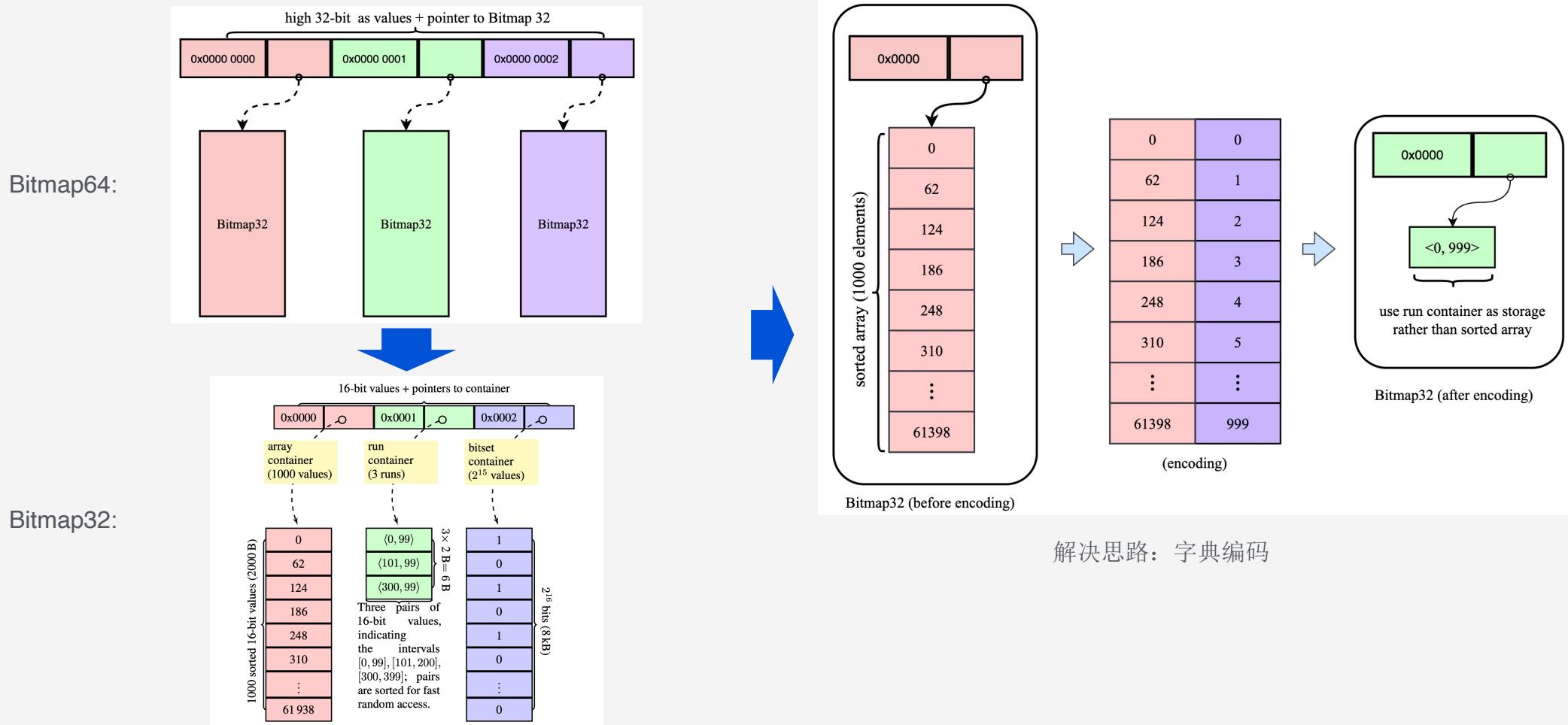
- MPP Join优化
- 湖仓一体生态兼容

~

内核优化举例(Bitmap 计算加速)

问题: 微信文章分析, 文章 id 均为 UInt64, 常见分析场景涉及 2k+ 个 bitmap 运算, 单机计算查询耗时 100s+

核心原因分析: 对于随机分布的数据, 构成的 bitmap64 内部布局稀



内核优化举例(Bitmap 加速引擎Bitbooster)

弹性扩容

计算图改写，加入repartition等计算

读取优化

读取并行化，in-place 构造

布局优化

优化bitmap64编码布局

指令集

开启RBM bitset等指令集优化

多机并行

优化数据分布，支持弹性伸缩

远程字典

保证全局一致性，提升易用性

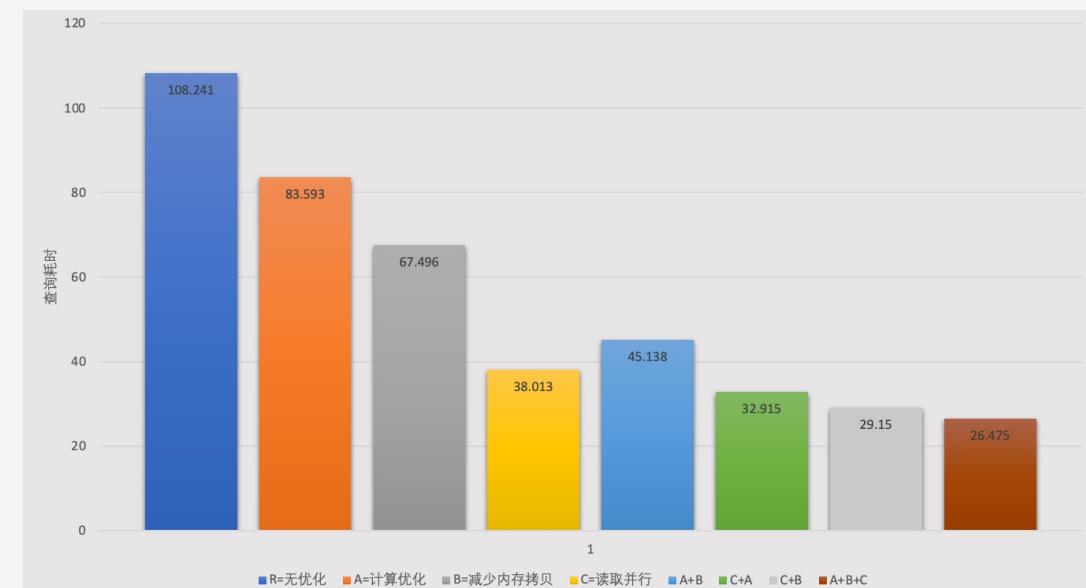
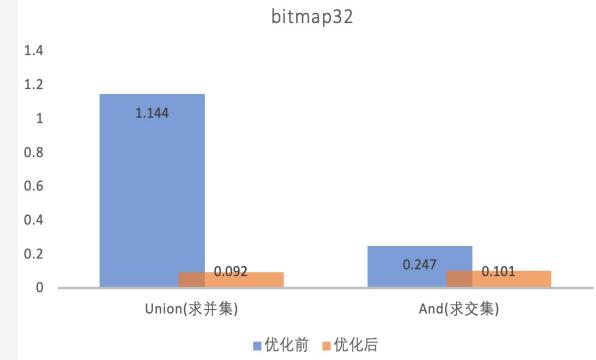
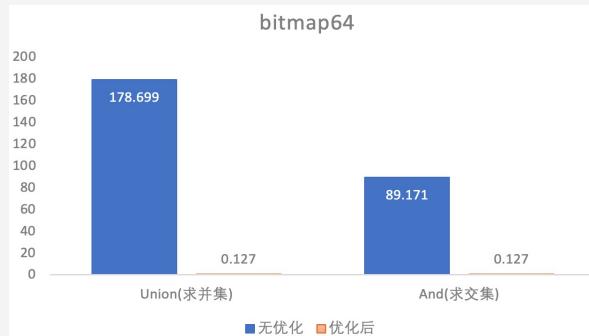
线上效果：

查询耗:100s -> 0.3s bitmap64

全表 bitmap 合并：超时-> 0.875s

测试数据集：

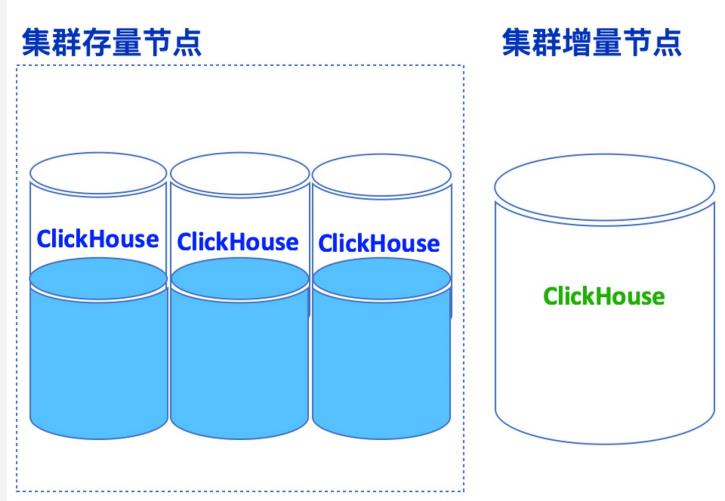
- 30个 bitmap64, 基数分布: $1000w * 5 + 100w * 7 + 10w * 9 + 1w * 9$;
- 30个 bitmap32, 基数分布: $1000w * 5 + 100w * 7 + 10w * 9 + 1w * 9$



云原生数仓-存算分离，按需查询更快

Shared-nothing 架构：数据本地化，但难以迁移，无法适应云上弹性能力

- 弹性扩容**：高峰查询更快，低峰成本更省
- 稳定性**：无ZK瓶颈，易读写分离，异地容灾
- 易运维**：数据容易均衡，存储无状态
- 功能全**：OLAP专注与查询优化 (cache、join)

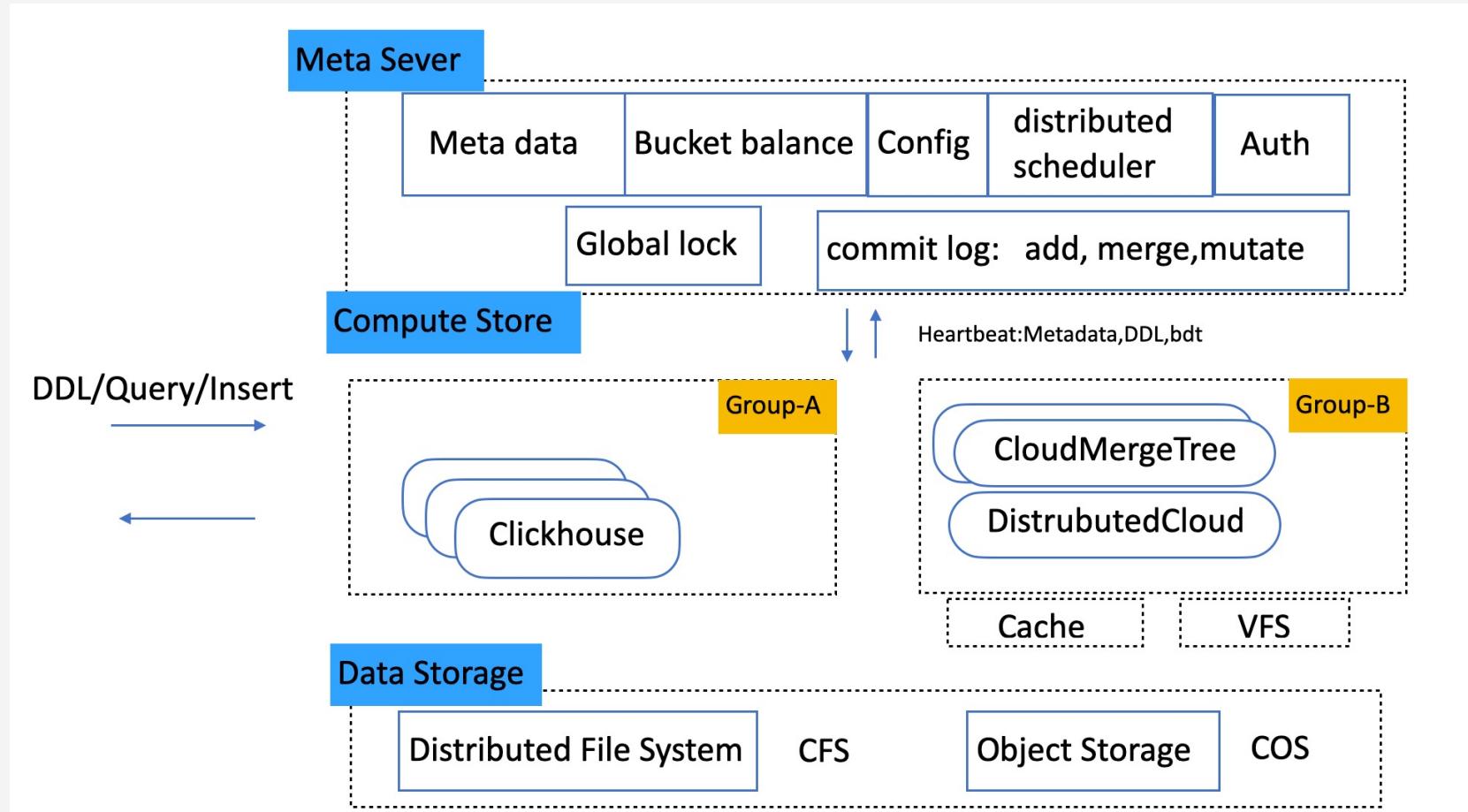


5 存算分离-远程读取优化

Meta Server: 基于分布式一致性协议实现的元数据管理服务、多集群数据均衡服务，分布式任务(DDL)调度服务，分布式集群的决策大脑

Compute Warehouse: 无状态，按需初始化；实现分组资源隔离，跨地域容灾；最小改动，Multi Master写入模型提供高吞吐，读取不经过Meta Sever

Data Storage: 对存储资源做统一抽象，无需关注底层存储，支持低成本对象存储，低延迟高吞吐存储，存储本身为强一致模型



5 存算分离-远程读取优化

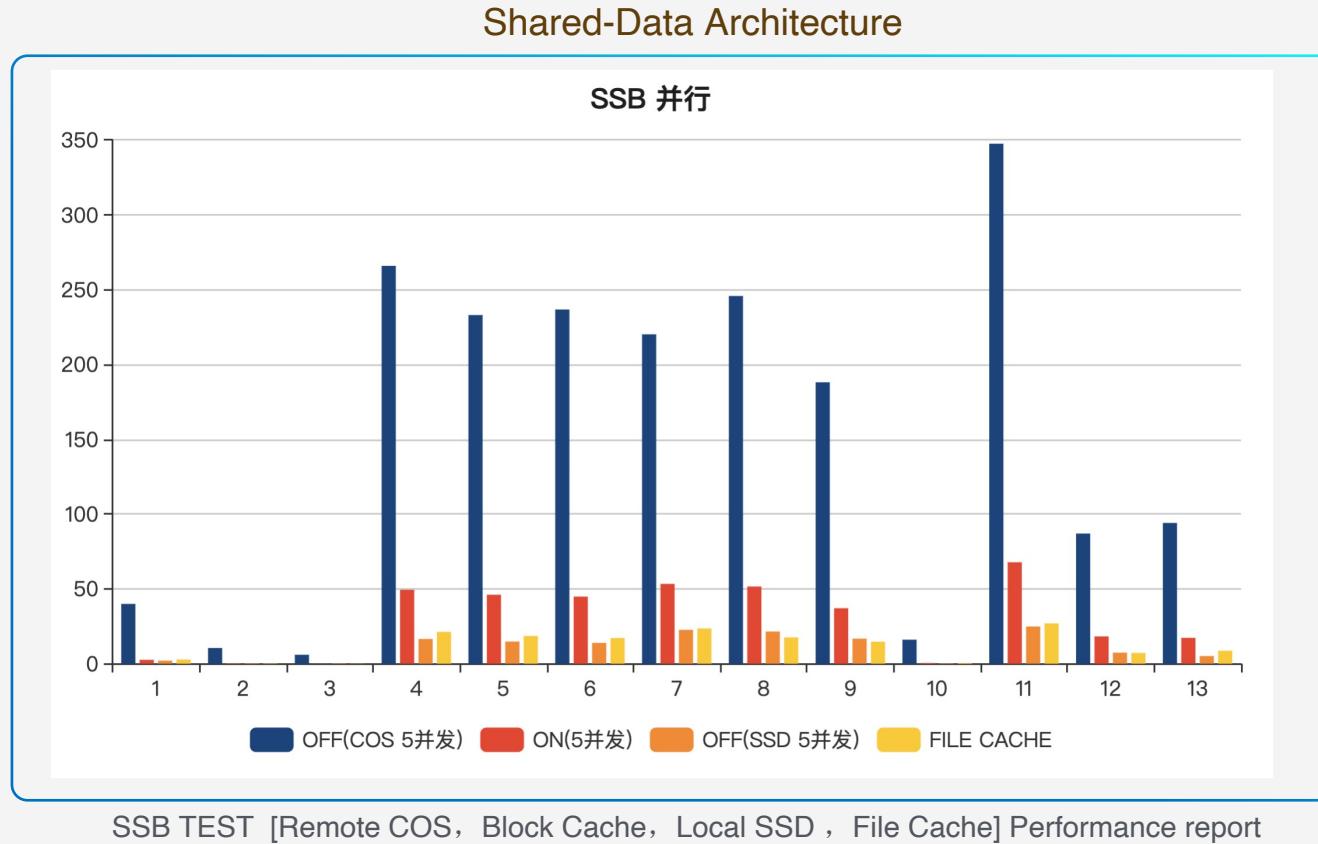
存算分离架构查询“按需”申请资源，节点更多，查询更快。但节点无状态方便迁移的同时，也带来远地访问额外的网络开销，因此打磨出真正适应生产的缓存策略和适合远程的并行方式，成为提升性能必须迈过的一道坎。

S3优化工作：

- Multi bucket
- compact
- COS proxy balance
- COS parallel read
- COS offset read
- COS concurrent write
- Metadata cache

DFS优化工作：

- Multi Cluster
- 表级bucket
- 目录布局优化
- 文件操作优化



研究表明，近一天的热数据承担了95%的查询，因此通过“有限缓存”解决“远地数据访问性能”具备统计理论支撑

发展历程

验证期



小规模试用

- ZK高负载
- 频繁OOM
- DDL卡死
-

2020年底

增长期



建设周边生态 业务发展迅速

- 生态建设(sinker,op-manager,qs,monitor)
- 内核优化，发布微信版本
- 查询优化，经验沉淀到平台
-

2021.03~2021.10

突破期



存算分离 云原生数仓

- 秒级弹性扩容，剔除ZK
- 远程存储读取加速
- Bitbooster bitmap加速
- 精确一次
- ...

2021.8 ~ now

未来



数据湖生态兼容

- MPP Join优化
- 湖仓一体生态兼容

~

Thanks

