ClickHouse在vivo大数据平台的落地实践

vivo互联网 | 大数据高级研发工程师 | 郭小龙

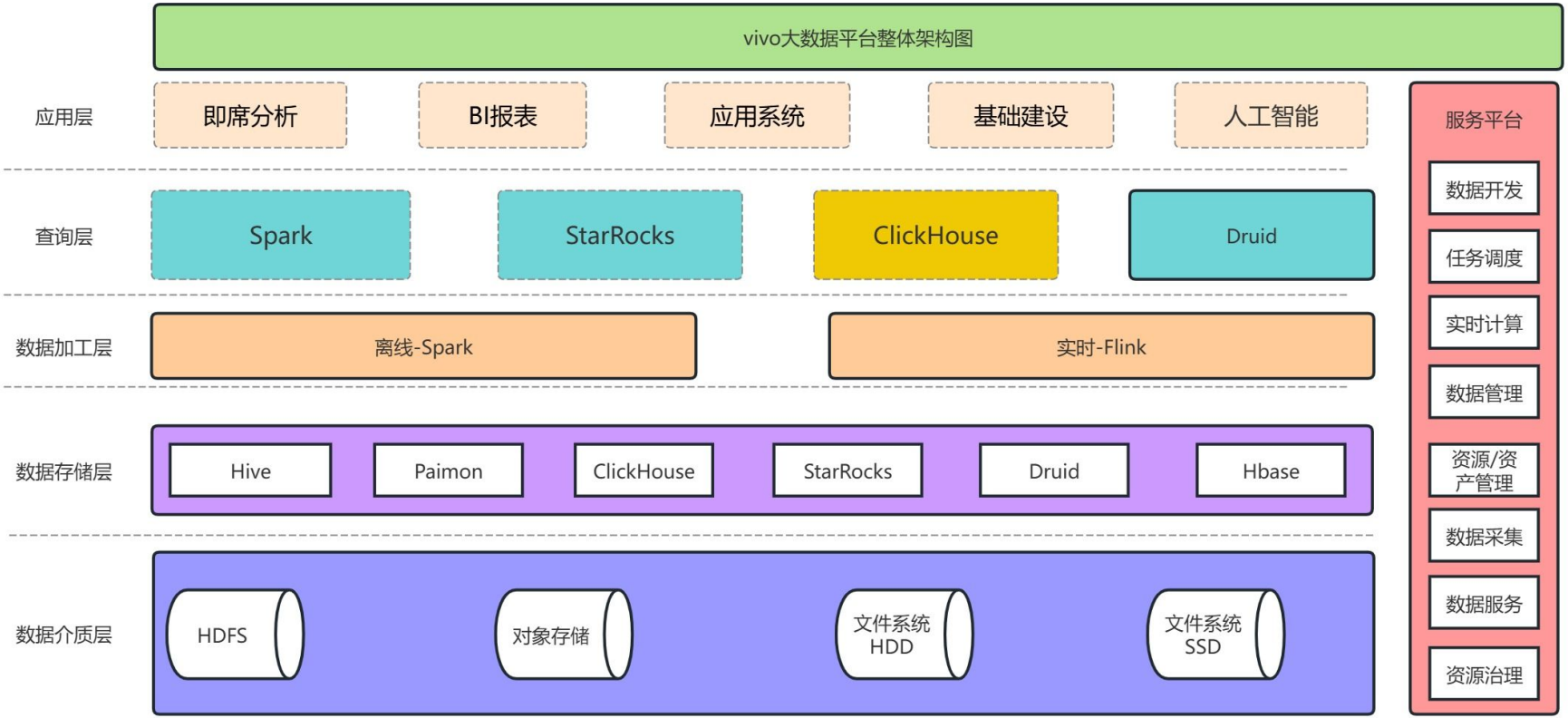
content

目录

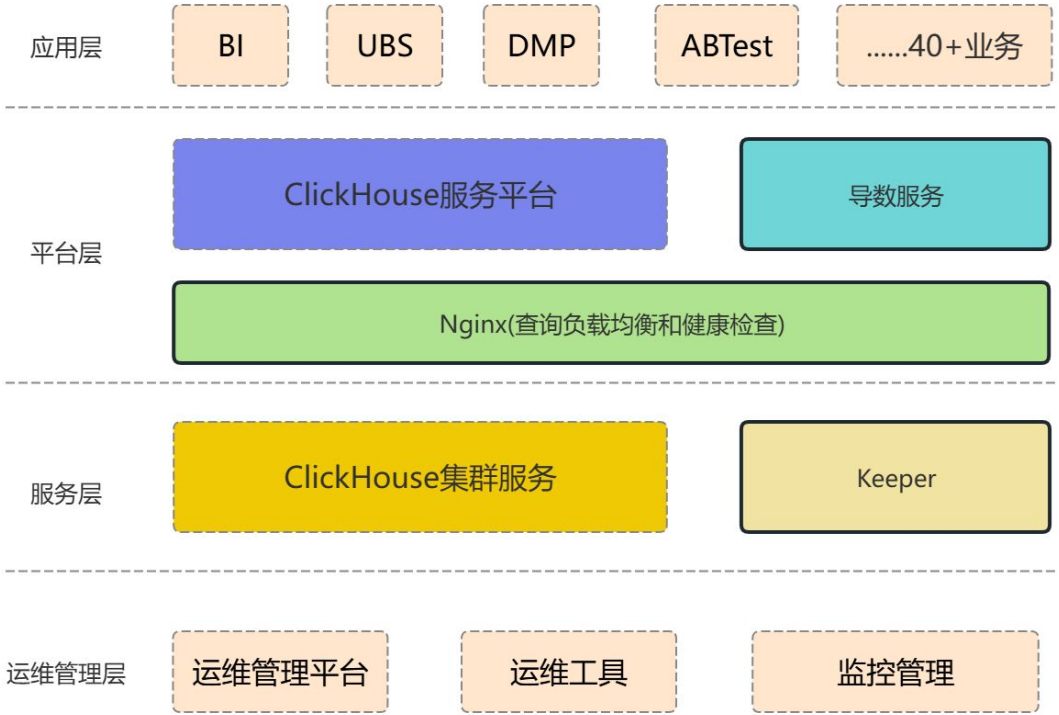
vivo大数据平台简介

01

vivo大数据平台概述



ClickHouse系统架构



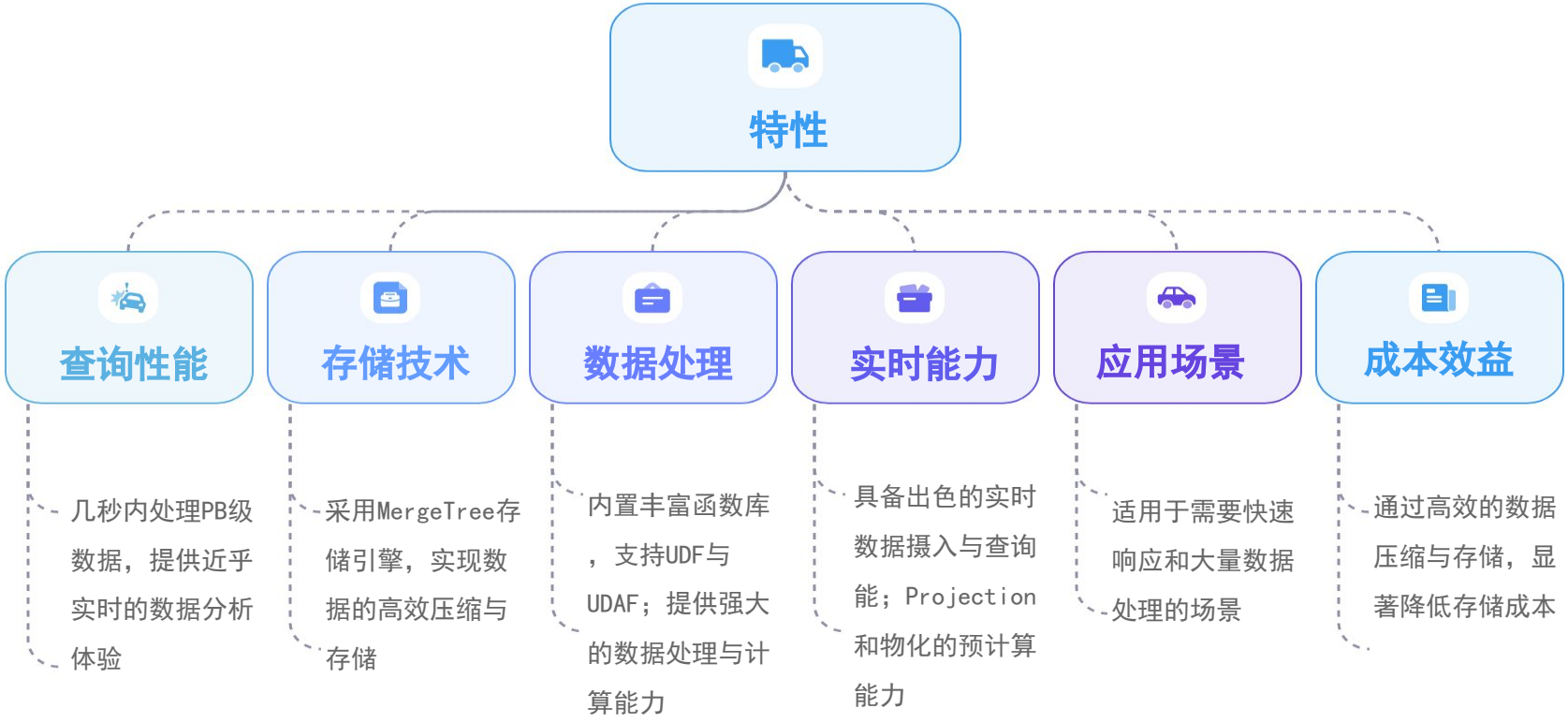
服务概述

- 2021年中引擎选型确定
- 2022年产品规划、能力建设，业务接入
- 当前版本为22.8-lts
- Clickhouse Keeper覆盖率接近100%
- 3个Kylin集群迁移到ClickHouse
- 接入不同类型业务40+
- 集群10+，数据行数百万亿，数据表数千张，磁盘PB级
- 服务整体可用性： 99.99%

引入平台能力

- 方便用户接入
- 规范用户合理使用
- 提供各种优质能力，提高服务水平

ClickHouse核心特性



ClickHouse服务管理与技术解决方案

02

资源申请与配置流程

资源组申请

用户提交资源组申请，按需选类型、定磁盘等参数，保障资源适配

项目属性绑定

提供项目详情，关联账户与项目，方便管理、核算成本，保障资源精准分配

成本系统绑定

关联预算成本管理系统控开支，强化成本管理

平台审核与配置

管理员按需选集群、分权限、建账号，保障用户安全高效使用所分配资源

库表管理服务

优化查询性能

通过优化查询算法，提高数据检索速度，减少响应时间，提升用户体验

实时数据统计

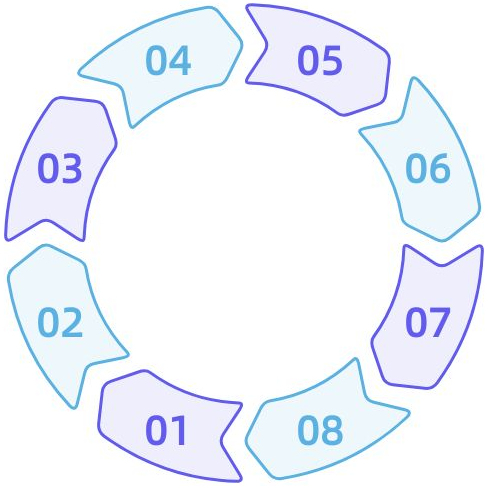
提供实时数据统计功能，帮助管理员及时了解数据库状态，优化资源分配

控制访问权限

细致调整表结构，严格控制访问权限，防止未授权访问，保障数据安全

提供库表管理界面

系统提供直观的库表管理界面，支持创建、修改、删除库表



简化日常维护

简化日常维护流程，降低维护成本，使数据库管理更加便捷高效

提升数据处理效率

通过技术手段提升数据处理效率，加快数据流转速度，增强系统整体性能

精细用户权限

精细控制用户权限，确保每位用户只能访问其被授权的数据

理解使用情况

利用数据统计功能，深入理解数据库使用情况，为后续优化提供依据

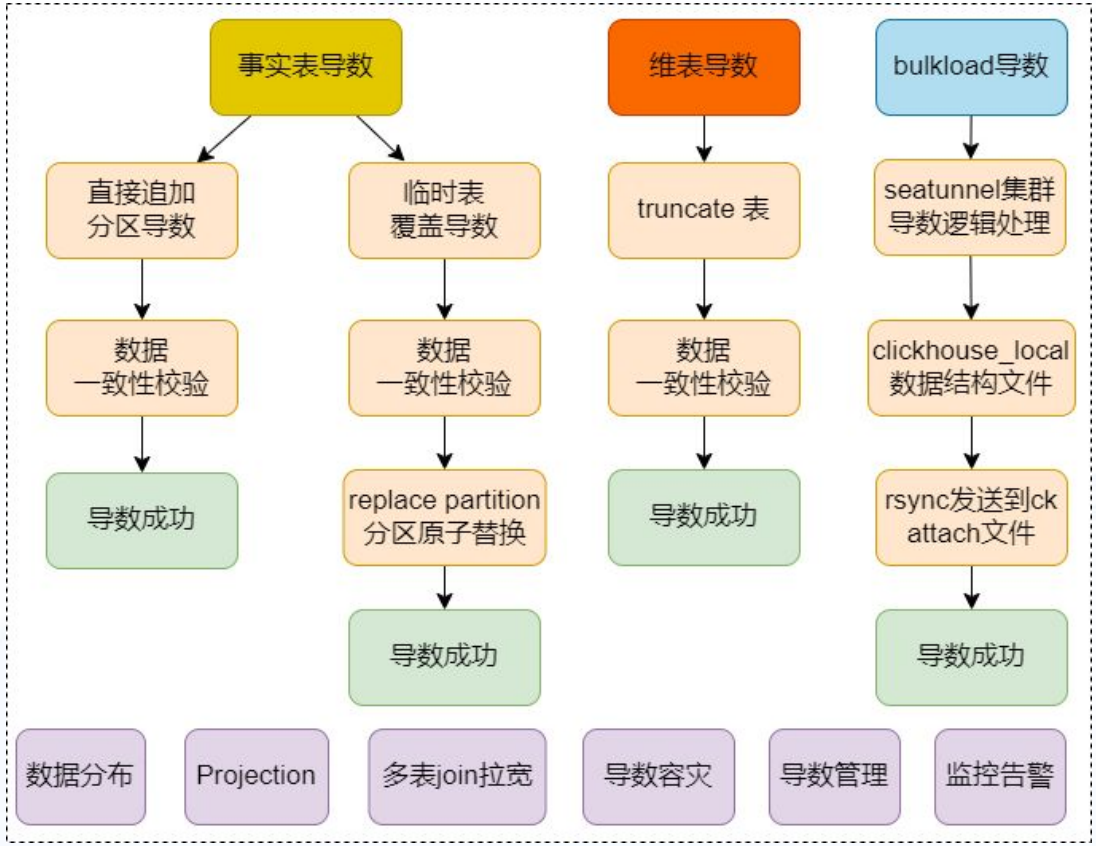
离线与实时导数管理

离线导数优化

通过事实表分区数据导入与维表数据导入，结合存算分离的bulkload技术，大幅提升大规模数据处理效率与灵活性

实时导数稳定性

采用高稳定性数据导入、智能分布策略和强大容灾管理，确保数据环境稳定可靠，满足实时数据流需求



本地表导入

20W一批次

异常告警

反压机制
选择空闲节点

存算分离

实时频率
30s-1min

借助buffer等
引擎

异步插入

定制需求

一级索引与去重优化

一级索引策略

•索引位置根据查询频率和基数决定

•查询频率越高越靠前

•查询频率接近则低基数靠前

•索引个数控制4~5个，个数越多则插入性能会下降

•Projection 索引调整

去重优化技巧

- 去重字段数据分布且使用distributed_group_by_no_merge=1分布式聚合
- 采用Projection，提前预聚合去重
- 去重字段用hash函数sipHash64，降低查询去重数据传输量大小
- groupBitmap精确去重
- 模糊去重方法代替精确去重方法，误差千分之二，推荐使用uniqCombined方法 。

Join操作优化

01

构造大宽表

通过预先执行表连接而非查询时动态Join，提前降低查询复杂度，节省资源消耗

02

改写SQL

手动改写SQL，大表在前，小表在后

03

均匀分布数据

确保Join操作中的关联字段数据分布均匀，避免数据倾斜

04

提升查询效率

优化性能，引入高版本并行join，reorder和runtime filter等特性

ClickHouse运维管理

运维自动化

标准化库表创建流程，自动化权限管理，使用Ambari平台简化集群管理，实现进程自动拉起与负载均衡，提升响应速度

常见运维场景

扩缩容、副本替换等操作标准化，故障机器快速恢复，ZooKeeper平滑迁移到Keeper，确保系统稳定运行

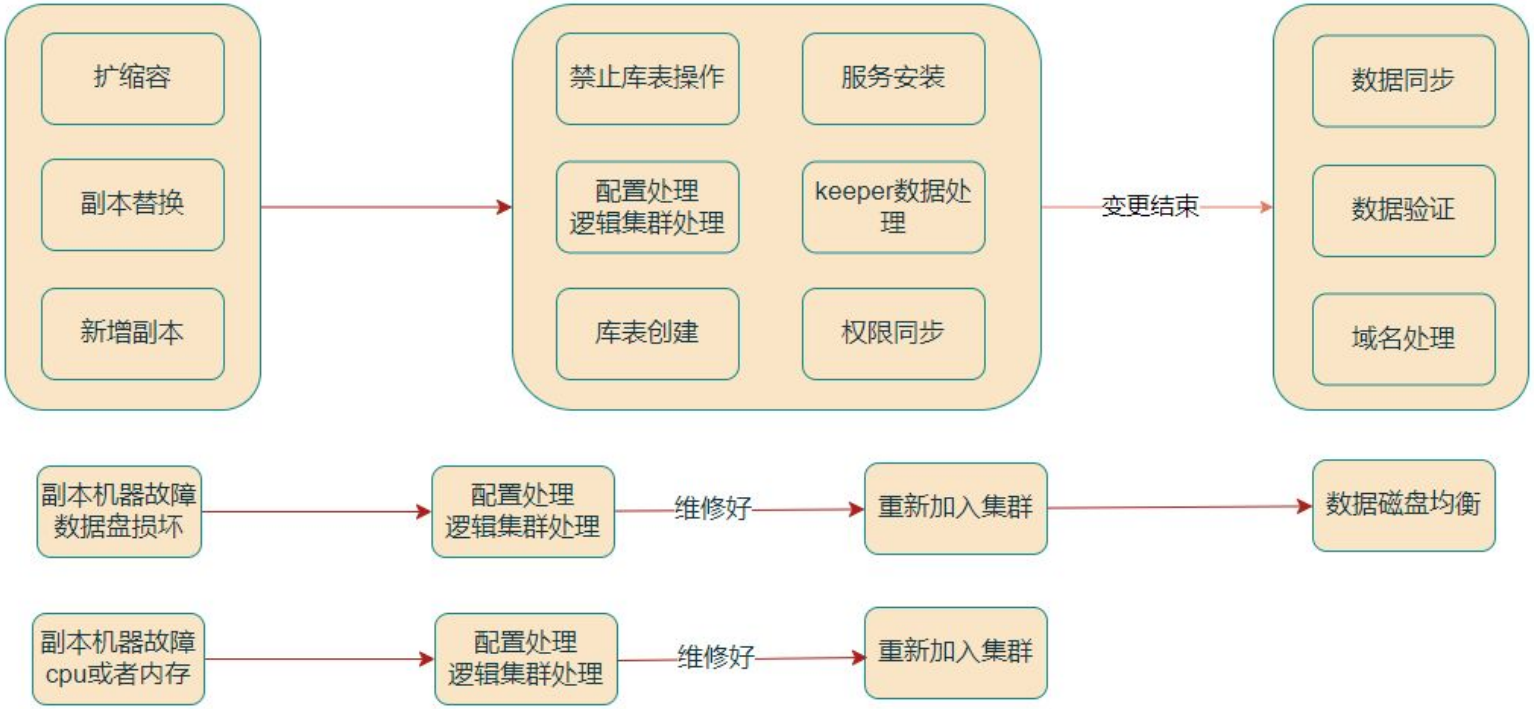
故障应对策略

数据保护机制，硬件故障快速响应，优化数据分布，确保数据一致性与服务连续性

持续优化实践

定期评估与调整，引入新技术，优化资源配置，提升ClickHouse运维效率与系统性能

ClickHouse常见运维操作



ClickHouse在某些运维场景需人工介入，虽看似不够智能，但此设计促使运维人员积累经验，深化系统理解，从而提升快速处理问题的能力。

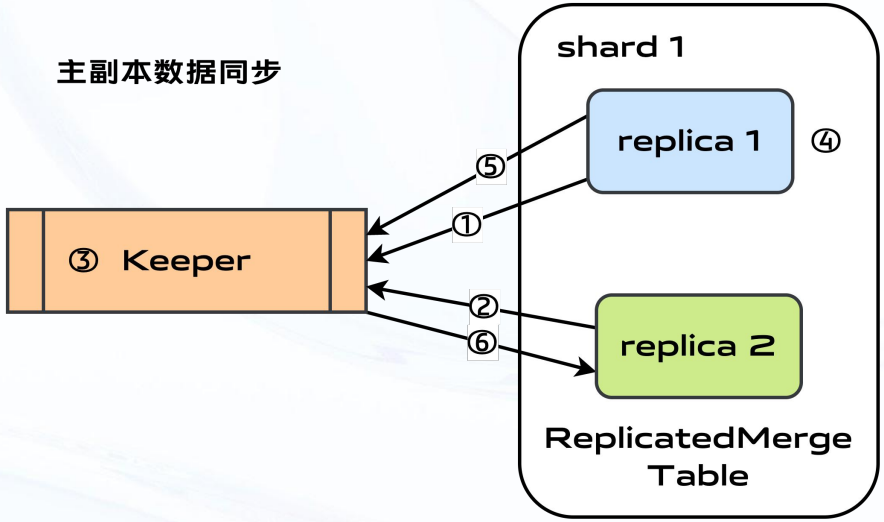
ZooKeeper和ClickHouse Keeper对比

ZooKeeper性能问题

- 因存储表元数据等信息，随znode增多性能下降，建议znode数量控制在1000万内
- 过多znode会导致GC卡顿，增加操作延时
- 只读问题突出，客户端经常超时
- 内存要求高，经常需要调整JVM堆内存
- 快照和日志未压缩，会导致磁盘打满和磁盘IO
- 存在zxid溢出问题

ClickHouse Keeper优势

- 使用C++开发，专属ClickHouse的分布式协调服务
- 基于Raft协议，能更快速恢复
- 性能高，基本上没有只读问题
- 支持对快照和日志的压缩，磁盘存储和IO消耗低
- 无zxid溢出问题



ZooKeeper迁移到ClickHouse Keeper

迁移步骤

- ClickHouse Keeper独立部署
- 通知用户，闲时停ClickHouse集群服务
- zk的快照通过ClickHouse开源工具
  ClickHouse-Keeper-Converter一键迁移
- ClickHouse集群修改配置启动
- 整个流程对用户产生影响预计控制在10分钟内
- 基础服务验证

资源消耗下降

迁移后效果

- 磁盘IO：峰值35%降到10%以下
- 内存使用率：从25%左右降到6%
- cpu资源使用率：降低明显
- 快照大小：347M减少到45M左右，下降87%
- 系统稳定性：只读发生率接近0

资源低消耗
Keeper集群混部

广告DMP应用实践

03

DMP平台挑战与改造目标

老DMP平台问题

基于Spark/Hive，存在扩展性差、计算资源消耗大、数据处理瓶颈及业务扩展难题

难以满足需求

现有架构难以适应新业务需求，限制了业务发展

核心改造目标

通过分离数据计算与业务逻辑，实现成本降低和效率提升

统一状态管理

优化状态管理机制，提高数据处理的一致性和可靠性

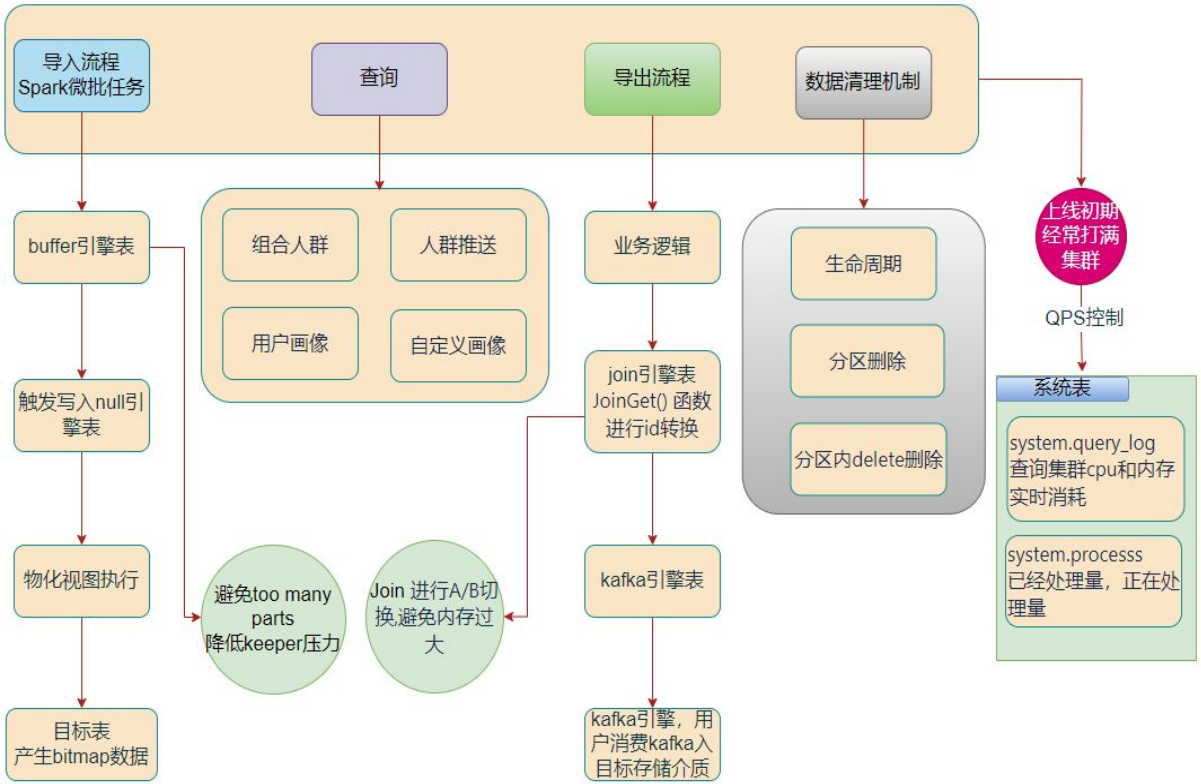
增强平台可靠性

提升系统的稳定性和可用性，确保业务连续运行

提高业务灵活性

使平台能够快速响应市场变化，支持更多元化的业务场景

DMP平台实现架构图



模块名	模块功能
CK-SQLGEN	基于ANTLR4的DSL到CK-SQL转换模块，高效生成计算基数、提取列表和执行插入选择的SQL语句
CK-LOADER	用于标签，人群数据从Hive借助Spark任务模板导入到CK表中，物化成bitmap数组
CK-PROXY	增强CK-SQL的使用体验，具备3大功能： • 自动处理Cluster主从副本故障切换； • 支持直接从本地表查询，无需依赖分布式表； • 封装HTTPClient和JDBC访问方式，简化数据交互
Dispatcher-Process	基于CQRS和Event Flow状态机，异步调度处理包括画像、组合，推送及数据导出等业务逻辑，并同步更新CK和MySQL中的数据
领域业务逻辑	基于Event，通过Dispatcher-Process模块进行链接与流程管理，实现具体业务逻辑的转换和处理

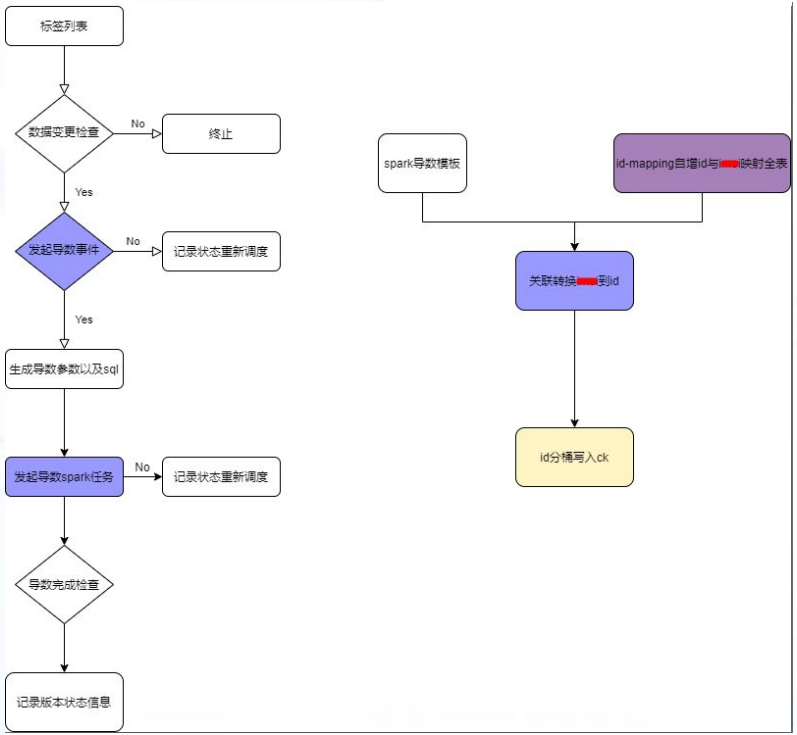
库表和导入设计

```
--定义buffer表用于ck进行写入
create table null引擎表buffer表 on cluster '逻辑集群' (
    aa_id UInt64 comment 'aa_id',
    分类_id  UInt32 comment '分类_id',
    标签_id  UInt32 comment '标签_id',
    version LowCardinality(String) comment '版本号'
)
engine = Buffer('库', 'null引擎表', 16, 120, 240, 1000000, 1800000, 58000000, 108000000);

---当buffer达到一定时间或者批次后进行flush 触发null表的insert
create table null引擎表 on cluster '逻辑集群' (
    aa_id UInt64 comment 'aa_id',
    分类_id  UInt32 comment '分类_id',
    标签_id  UInt32 comment '标签_id',
    version LowCardinality(String) comment '版本号'
) engine = Null;

--最终我们需要生成的bitmap数据表
create table bitmap表 on cluster '逻辑集群'(
    分类_id UInt32 comment '分类_id',
    标签_id UInt32 comment '标签_id',
    bitmap AggregateFunction(groupBitmap, UInt64) comment '自增id的roaringbitmap',
    version LowCardinality(String) comment '版本号'
) engine = ReplicatedAggregatingMergeTree
    PARTITION BY (version, 分类_id)
    ORDER BY (version, 分类_id, 标签_id)
    SETTINGS index_granularity = 1, storage_policy = '存储策略';

---当buffer flush 时候触发insert null表进行物化生成bitmap数据
CREATE MATERIALIZED VIEW 库.物化视图 on cluster '逻辑集群' TO 库.bitmap表 (
            `分类_id` UInt32,
            `标签_id` UInt32,
            `bitmap` AggregateFunction(groupBitmap, UInt64),
            `version` LowCardinality(String)
        )
AS SELECT 分类_id, 标签_id, groupBitmapState(aa_id) AS bitmap, version
FROM 库.null引擎表
GROUP BY version, 分类_id, 标签_id;
```



- buffer表，null引擎表和物化视图来实现最终目标表的bitmap类型的生成

- id进行%集群总shard数，得到每一个ck分片节点

ClickHouse方案经验总结

01

znode数暴增&Too Many Parts

问题：高频小批次插入产生大量part文件，导致Merge线程过载和znode数量激增，影响集群稳定性

方案：使用buffer表引擎或async insert机制减少小文件生成

02

控制内存增长

挑战：每日导入数据更新频繁，历史记录在内存中累积，导致内存使用量持续增加

方法：使用ifnull函数进行A/B表切换，并在每次数据导入后清空join数据

03

QPS限制与SQL调用频率控制

需求：针对QPS限制，在计算量突然增大时控制SQL执行频次

策略：通过ck-proxy层监听机制，定时统计SQL执行情况。当系统负载高时，限制特定SQL的调用频次以维持系统稳定

04

同分区下的交集运算优化

同一目录_id下的多个标签_id进行交集运算结果为0时，需要拆分为多个子查询使用groupBitmapAndState来计算，以提高准确性和效率

05

组合人群删除方式

设置 TTL 后，部分标签一天内仍有多个版本，需手动删去不必要版本，直接删除整个分区或者用分区查询后删指定数据

06

其他优化建议

明细表通过Projection及物化字段代替物化视图，简化数据处理流程

对于复杂逻辑，可利用自定义UDF/UDAF实现，支持Python、C++或SQL创建，满足多样化业务需求

应用ClickHouse方案后收益

01

资源节省显著

- 新架构只需40T的ClickHouse存储
- Yarn队列：节省了2500个CPU核心资源和20000GB内存资源

02

计算耗时锐减

- 组合人群计算：时间从30分钟降至1分钟内
- 画像洞察计算：从30-60分钟缩短至1-2分钟
- 自定义画像面板：生成时间控制在30秒内
- 人群推送实时性：触达时间缩短至7-15分钟

03

业务灵活性增强

- 高效性能和资源节省，为业务扩展和技术演进提供了更广阔的空间，增强了业务灵活性和响应速度
- 后续ClickHouse方案会应用分析，检测校验，因果推断等商业化场景

风控业务应用实践

04

ES存在的问题

高昂的存储成本

随着业务流量激增，ES存储成本飙升

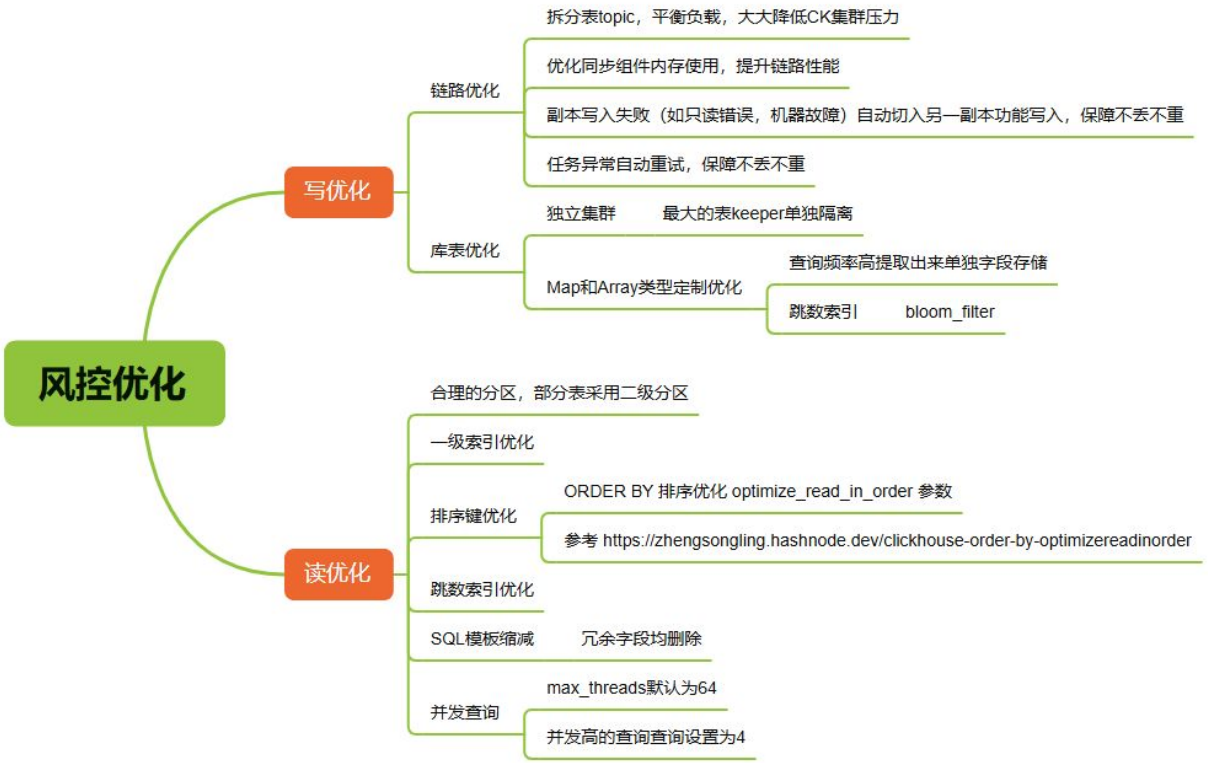
稳定性隐患

频繁遭遇写入延迟，每月至少1至2次，恢复时间长，严重影响业务连续性和用户体验；节假日存储天数需要缩短

高维护成本

为保持节点负载均衡，需频繁调整场景分片数量，字段变更依赖人工干预，增加运维负担

业务改造和经验总结



经验总结

- 整体顺利，无需额外架构设计，业务改造较小
- 选择合适的建表设计和索引优化，查询性能满足用户需求
- 导数做了稳定性保障，保障不丢不重无延迟
- 查询结果一致性没问题
- 价值收益明显，用户痛点解除
- CK代替ES是一个迁移成本低收益大的成熟方案

切换到ClickHouse组件的收益

系统稳定性增强

提高业务连续性和用户体验

写入零延迟

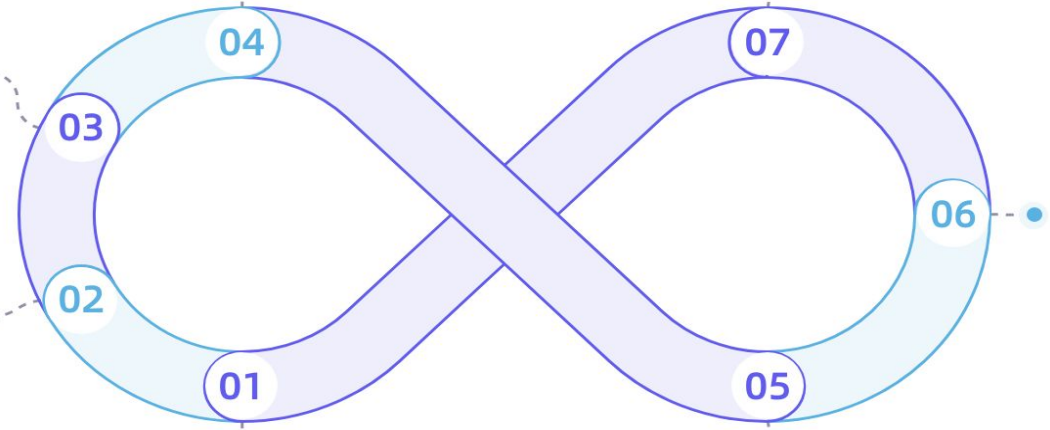
实现了写入操作的零延迟，即使在进行大规模查询时也不会受到影响

查询速度提升

明细查询速度与ES相当，聚合查询速度提升30%，提高了数据处理效率

存储成本降低

存储成本大幅降低71.3%



人力资源释放

简化操作流程和自动化功能减少了对人力资源的需求，提升了团队的工作效率

维护成本下降

通过减少手动干预和自动化管理，有效降低了系统的维护成本，节假日0运维

操作更加简便

无需频繁调整分片数量，字段变更过程自动化，降低了维护难度。

未来展望

05

25.3-lts新版本预研

01

Join性能优化

- 并行hash join优化
- reorder 自动优化join执行顺序
- runtime filter 谓词下推到join两侧

02

新特性

- 时序表引擎，支持存储Prometheus数据，满足监控业务需求
- 全新架构的Json引擎，满足半结构化数据高效存储和查询
- 自适应异步插入

03

性能优化

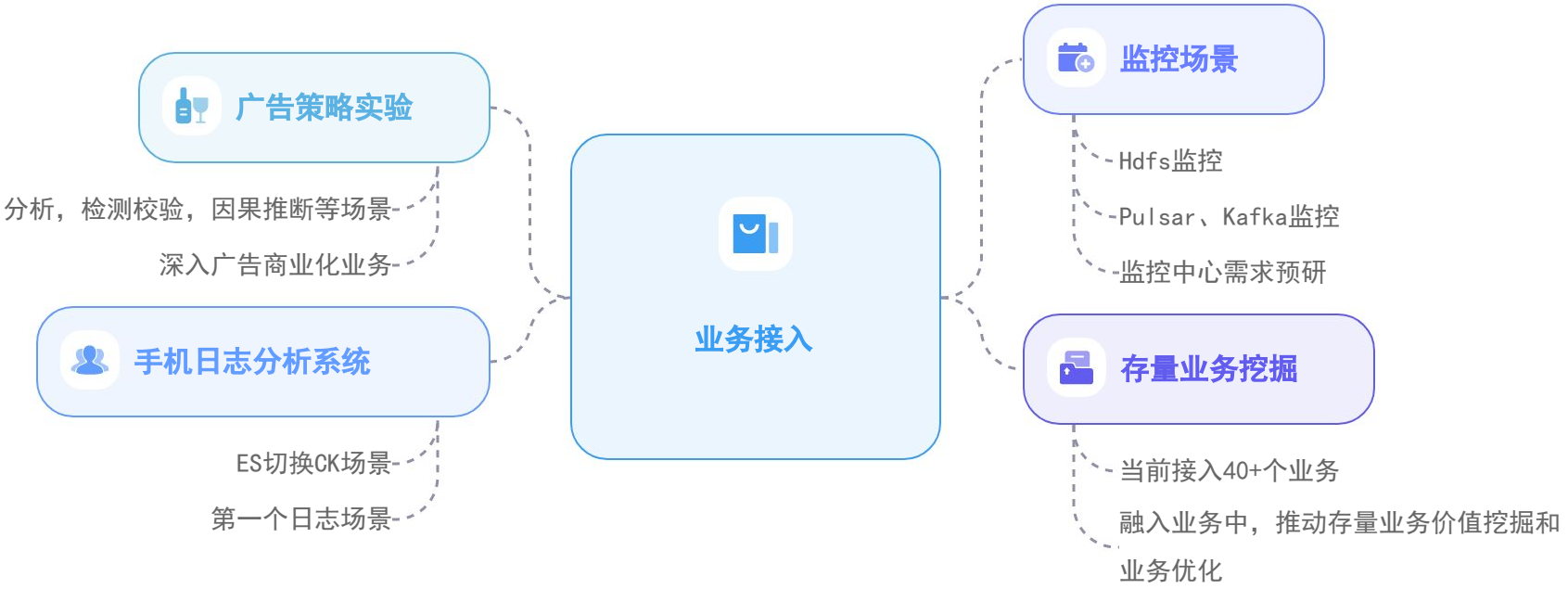
- 查询和导入的性能提升
- order by查询性能优化
- S3相关优化
- keeper性能优化

04

稳定性

- 表只读问题根本性解决
- S3稳定性问题解决
- ……

业务场景探索


广告策略实验
分析，检测校验，因果推断等场景
深入广告商业化业务
手机日志分析系统
ES切换CK场景
第一个日志场景
业务接入
监控场景
Hdfs监控
Pulsar、Kafka监控
监控中心需求预研
存量业务挖掘
当前接入40+个业务
融入业务中，推动存量业务价值挖掘和业务优化

THANKS

关注我们，了解更多技术干货