

# *When ClickHouse Meets Iceberg*

*How Data Lake Analytics Implemented in WeChat*

冯吕 腾讯微信

# About Me

- *ClickHouse Collaborator & Active Contributor, 180+ merged PRs*

➤ <https://github.com/ucasfl>



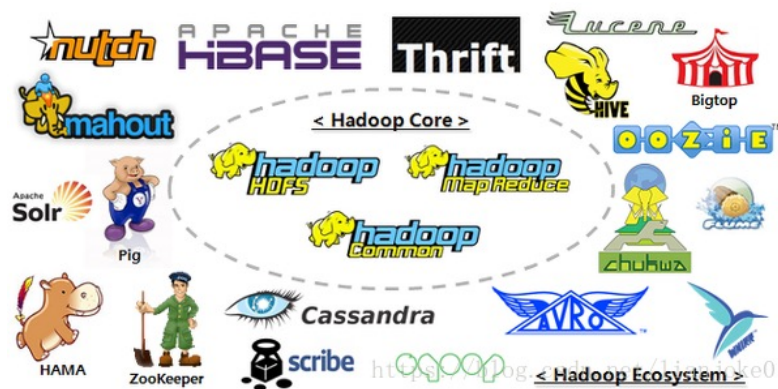
- *Software engineer at Tencent WeChat*
  - *Working on ClickHouse development*

# 目录

- 背景
- 湖读取设计与优化
- 业务实践
- 未来规划

背景

# 从 Hadoop 到亚秒级实时数仓

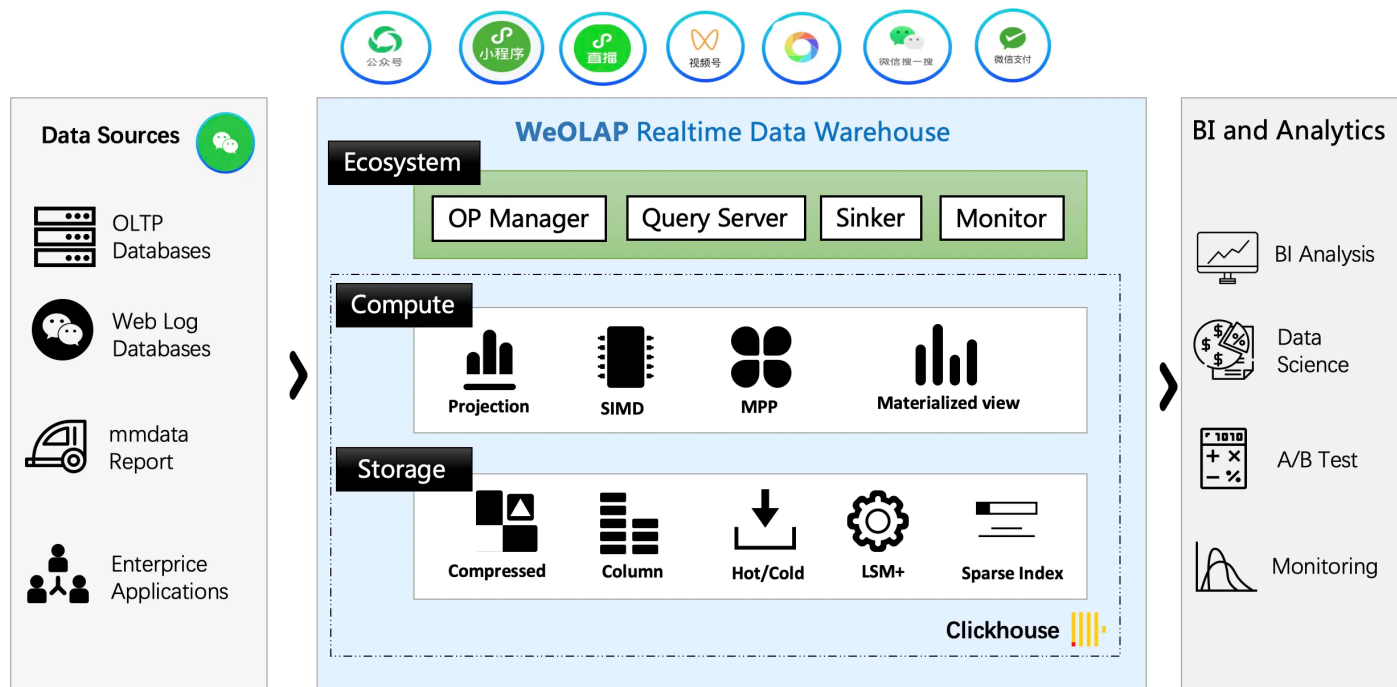


Hadoop 生态:

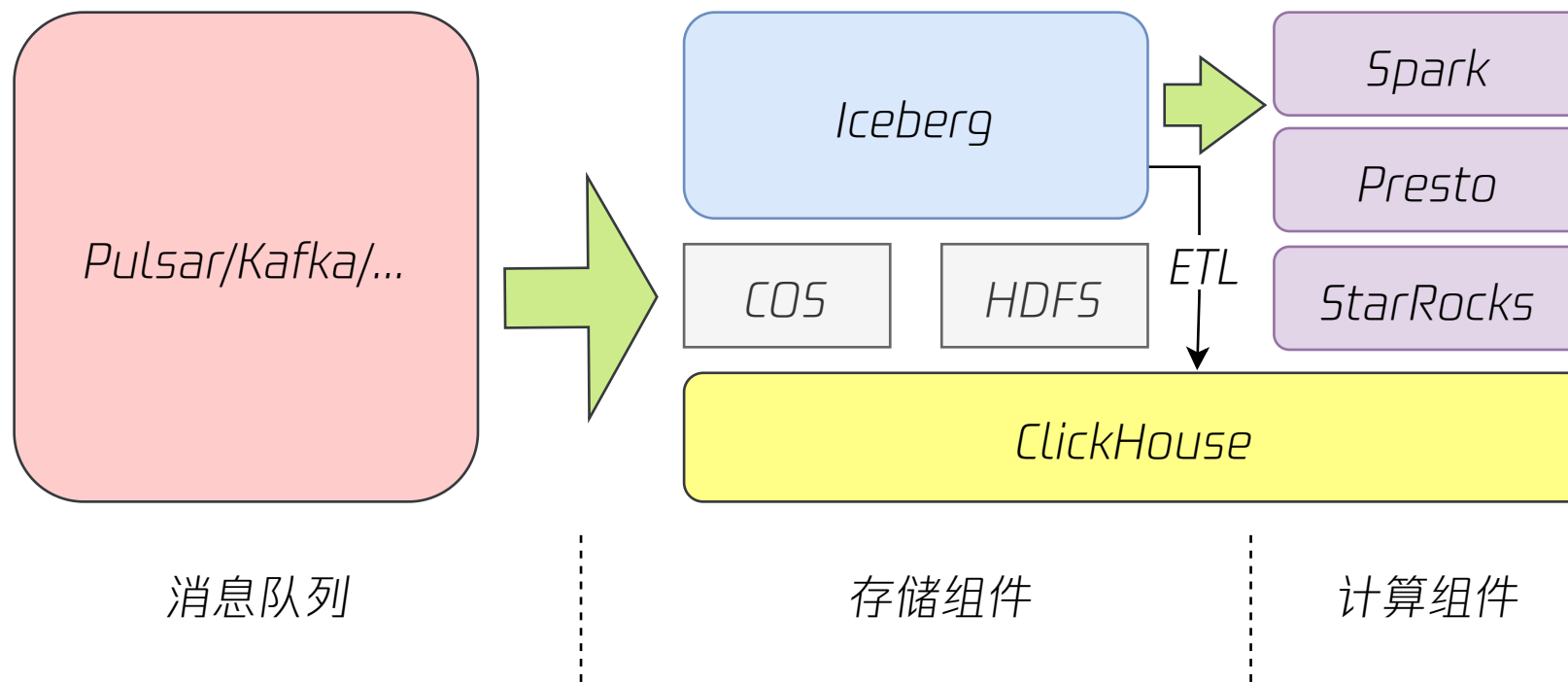
- 慢: 查询慢, 延迟高, 开发慢
- 流批分离
- 架构臃肿

视频号等推荐系统对个性化体验的强烈诉求, 催生了“亚秒级”分析系统的诞生:

- 亚秒级响应
- 万亿级数仓
- 流批一体
- 精确一次、低时延接入

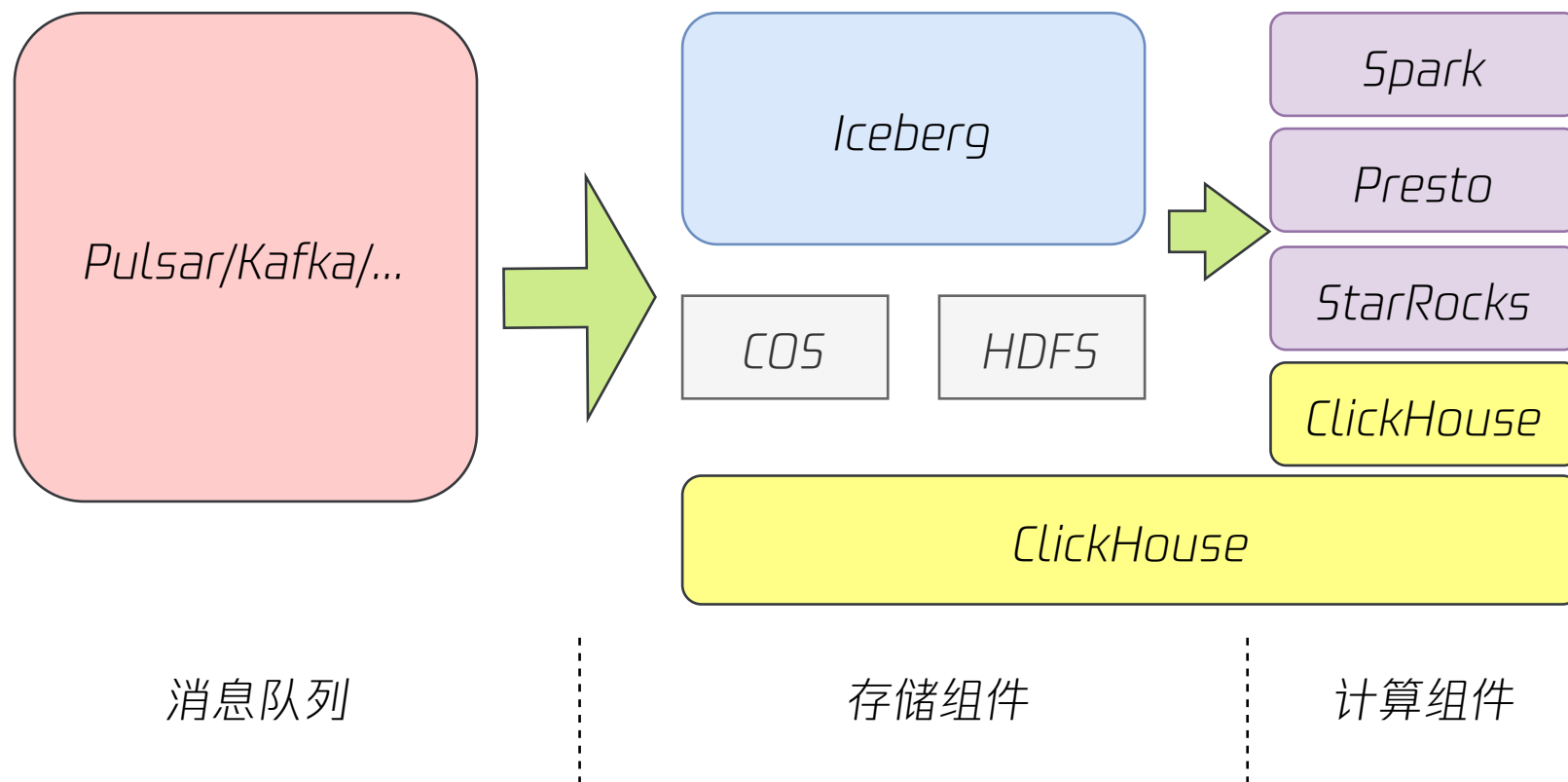


# Why ClickHouse + Iceberg



- ClickHouse 中的数据存在数据孤岛倾向
- 湖上数据分析依赖于繁重的 ETL，流程繁琐
- 存储冗余，资源浪费

# Why ClickHouse + Iceberg



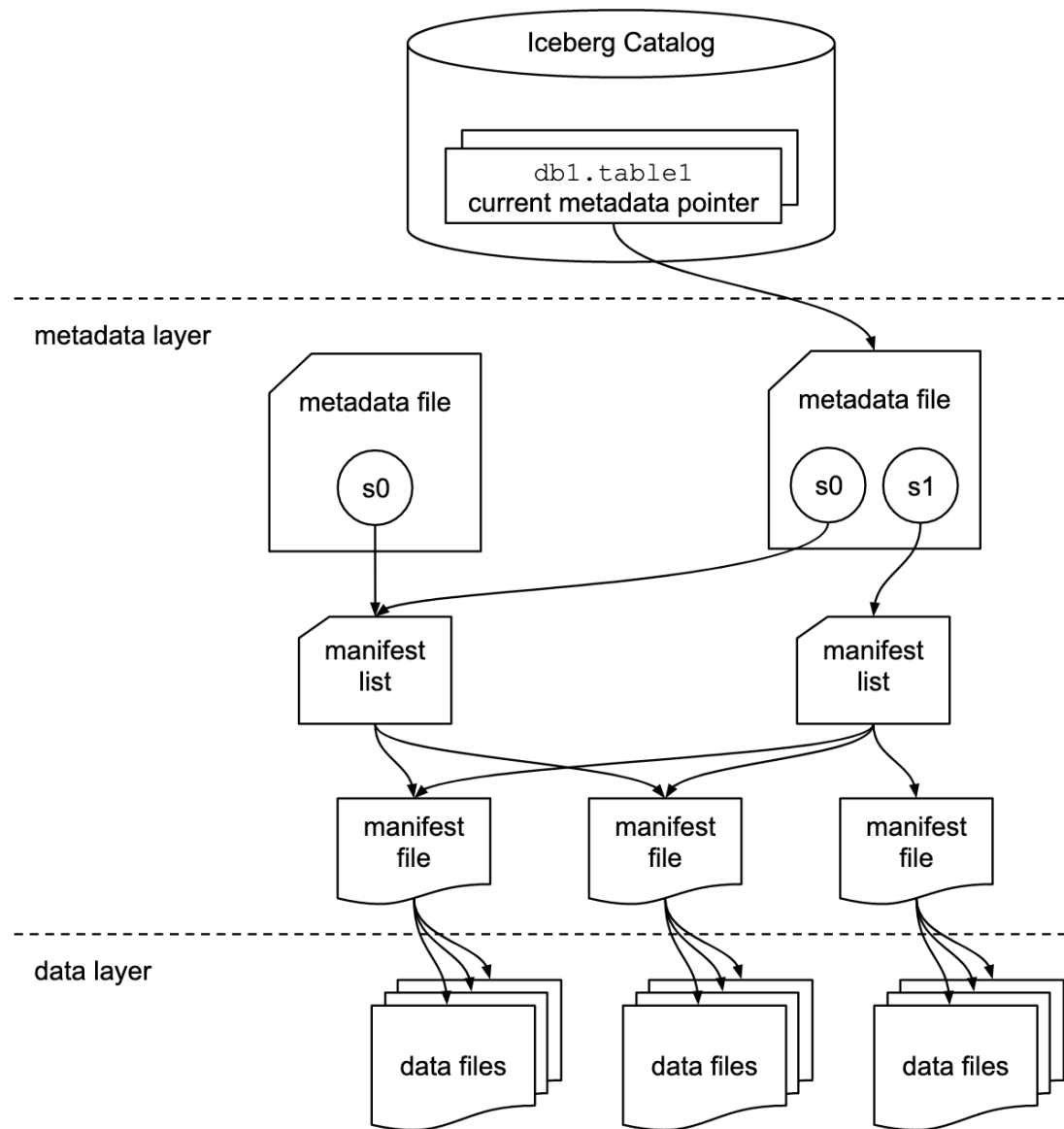
内核增强: ClickHouse 作为计算组件, 支持高性能 Iceberg 数据湖分析

# 湖读取设计与优化



# Iceberg 简介

- 通用开源表格式，架构简洁、轻量  
具有比 *Hive* 更好的性能
- 三层元数据结构：
  - *metadata* -> *manifest list* -> *manifest file*
- 在进行 *Iceberg* 表查询时，需要  
利用元数据进行分区剪裁、*min-max* [文件粒度] 索引剪裁等工作
- 痛点: *Iceberg* 无 *C++ API*, 而  
*ClickHouse* 是纯 *C++* 开发的



# 社区 Iceberg 读取实现

## Add StorageIceberg and table function iceberg #45384

Merged kssenii merged 44 commits into ClickHouse:master from ucasfl:iceberg on Feb 18

Conversation 69 Commits 44 Checks 131 Files changed 134



ucasfl commented on Jan 18

Collaborator ...

### Changelog category (leave one):

- New Feature

### Changelog entry (a user-readable short description of the changes that goes to CHANGELOG.md):

Add StorageIceberg and table function iceberg to access iceberg table store on S3.



12

不足:

- 单机读取，不支持分布式
- 仅支持 S3 上的 Iceberg 表
- 由于 Iceberg 无 C++ API，所有查询均为全表扫描，真实业务场景不可用

<https://github.com/ClickHouse/ClickHouse/pull/45384>

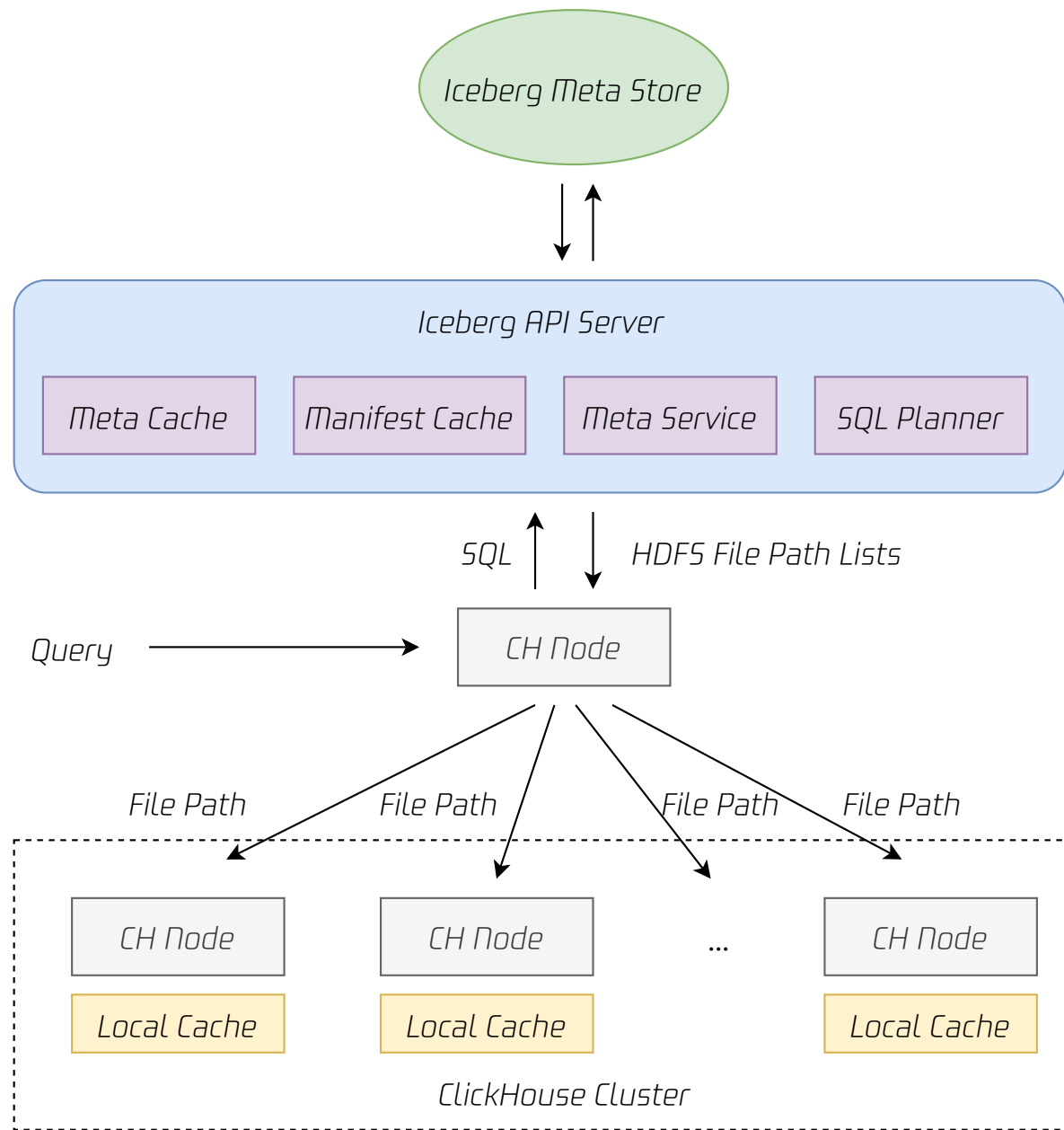
# 如何实现高性能数据湖读取

- 方案一: *Pure C++*, 延续现有实现, 需要通过 *C++* 实现一整套 *Iceberg API*
  - ✓ 优点: 社区接受度高, 元数据处理性能更高[理论上]
  - ✓ 缺点: 工程量大, 后期维护困难[*Iceberg* 还在不断发展演化中]
- 方案二: 引入外置 *Iceberg API Server* 来提供元数据服务
  - ✓ 优点: 工程量较小, 后期维护简单
  - ✓ 缺点: 社区接受度低
- 业内方案:
  - *duckDB*: [Iceberg Extension](#), 纯 *C++* 实现, 但目前只有基础功能
  - *ByConity*: 通过 [JNI 调用实现了 Hudi 表的读取](#), 但还未实现 *Iceberg* 读取

我们的选择: 方案二

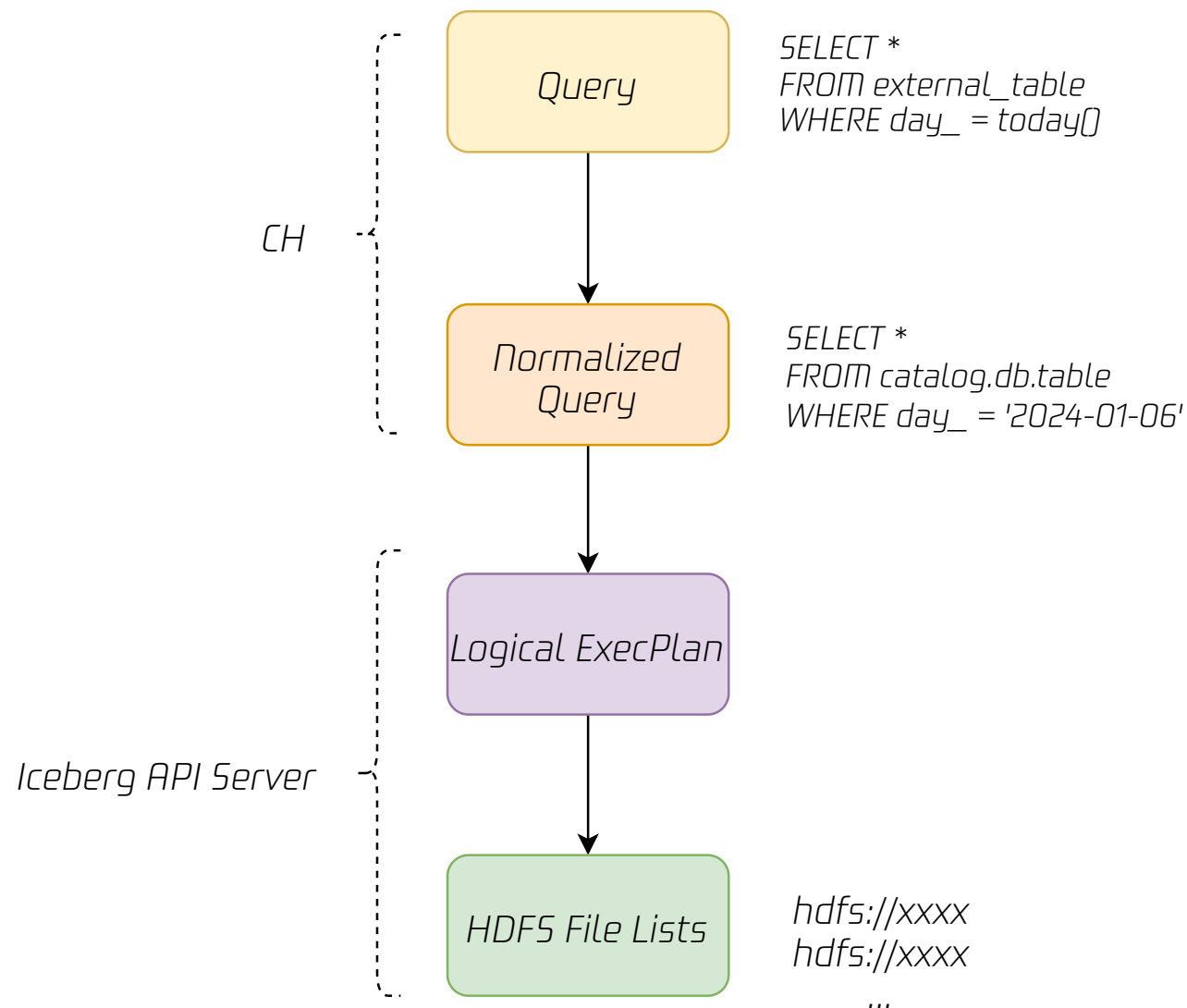
# 整体架构

- 引入 *Iceberg API Server*
  - ✓ 元数据服务
- 提交到 *CH* 的外表查询会与 *API Server* 进行交互，完成分区剪裁、*min-max* 剪裁等得到需要读取的文件列表
- 文件列表按照一致性 *hash* 分发到集群节点进行分布式读取



# 查询交互

- 提交到 *ClickHouse* 的外表查询SQL, 会先转换成 *MySQL* 兼容 SQL, 然后提交到 *Iceberg API Server*
  - ✓ BugFix: [PR#56456](#)
  - ✓ BugFix: [PR#57888](#)
- *API Server* 接收 SQL 请求, 对 SQL 进行 *Plan*, 得到 *Logic Plan*, 从 *Logic Plan* 中拿出全部文件路径, 并返回给 *ClickHouse*



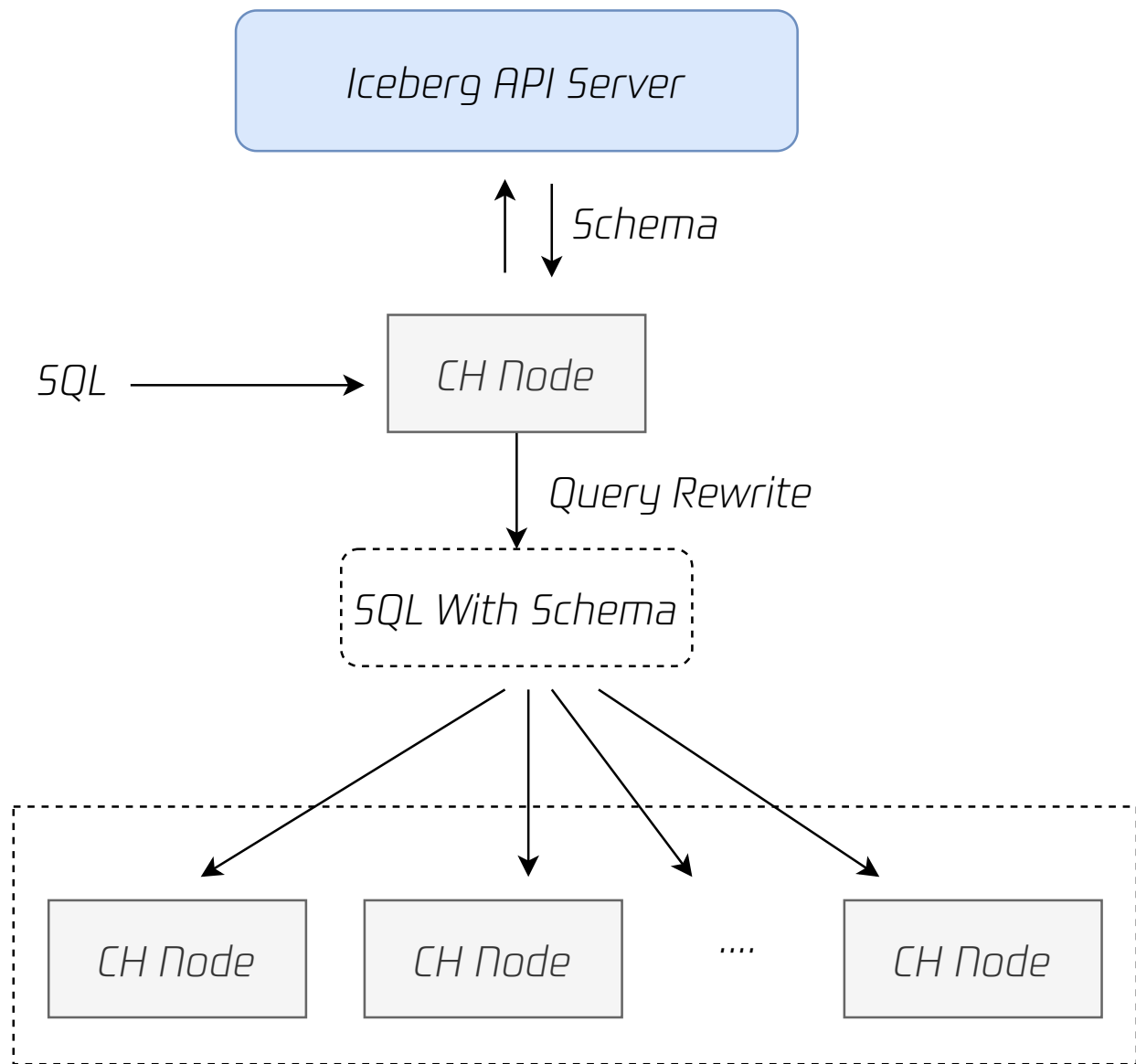


# *Optimizations*

- 元数据访问优化
- 本地缓存
- *Parquet Filter Push down*
- *Parquet* 读取内存优化
- 全表扫描限制
- 读取集群、聚合集群分离

# 元数据访问优化

- 自动 *schema* 推导:
  - ✓ 发起查询节点请求 *API Server* 获取 *schema*;
  - ✓ 通过查询改写下发 *schema* 到全部分布式查询节点;
  - ✓ 避免分布式查询节点多次元数据请求, 减少元数据请求开销;





# 本地缓存

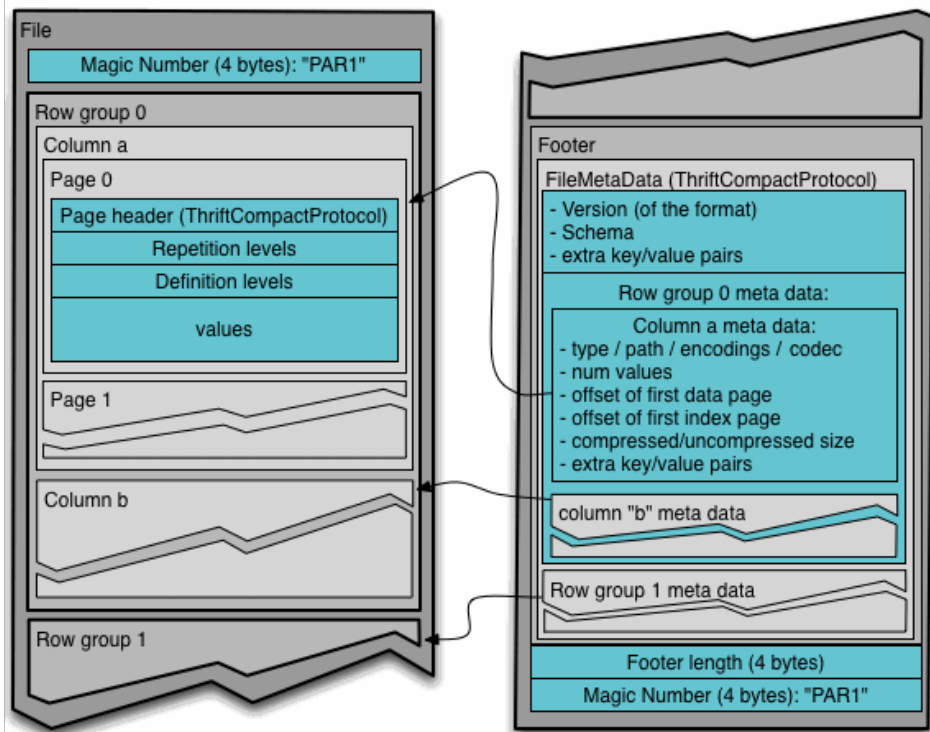
- 在读取节点本地按照 LRU 策略缓存远程文件，命中缓存时无需再从远程读取
- 文件分发策略：随机分发 -> 按照一致性 hash 分发

```
...) select count() from iceberg(iceberg, ...);  
  
SELECT count()  
FROM iceberg(iceberg, ...)  
  
Query id: ebb352f1-cf77-421b-8d83-c91455b3f4dc  
  
count()  
154730727  
  
1 row in set. Elapsed: 7.178 sec. Processed 154.73 million rows, 1.24 GB (21.56 million rows/s., 172.45 MB/s.)  
  
...) select count() from iceberg(iceberg, ...);  
  
SELECT count()  
FROM iceberg(iceberg, ...)  
  
Query id: fdea2197-3e1f-4224-a365-0d1734c78f89  
  
count()  
154730727  
  
1 row in set. Elapsed: 2.007 sec. Processed 154.73 million rows, 1.24 GB (77.10 million rows/s., 616.78 MB/s.)
```

缓存命中情况下, 2X 以上性能提升

# Parquet 谓词下推

- Iceberg 元数据的 min-max 剪裁, 仅到文件级别, 不能剪裁 Row Group
- Parquet 谓词下推: [PR#52951](#), 利用 min-max 剪裁不需要读取的 Row Group
- 特殊情况下 [数据排序后], 一个数量级以上的 IO 优化



```
mmexpt_lakehouse.metric_vector_8616 :) select sum(metric_id), sum(version), max(ds), sum(length(denominator_vec)), sum(length(numerator_vec)), sum(length(only_exist_vec)) from mmexpt_lakehouse.metric_vector_8616 where ds = '2023-12-08 00:00:00' and metric_id=1000 and version = 2023120700 and bucket_num < 32 ngs_input_format_parquet_filter_push_down=0;

SELECT
  sum(metric_id),
  sum(version),
  max(ds),
  sum(length(denominator_vec)),
  sum(length(numerator_vec)),
  sum(length(only_exist_vec))
FROM mmexpt_lakehouse.metric_vector_8616
WHERE (ds = '2023-12-08 00:00:00') AND (metric_id = 1000) AND (version = 2023120700) AND (bucket_num < 32)
SETTINGS input_format_parquet_filter_push_down = 0

Query id: 81bf0474-9c2f-4d81-a319-64d543c443bb

sum(metric_id) | sum(version) | max(ds) | sum(length(denominator_vec)) | sum(length(numerator_vec)) | sum(length(only_exist_vec))
-----|-----|-----|-----|-----|-----
1 | 1 | 2023-12-08 00:00:00 | 1 | 1 | 1

1 row in set. Elapsed: 10.618 sec. Processed 1.02 thousand rows, 143.28 MB (96.44 rows/s., 13.49 MB/s.)

mmdcch2test-shard1-mmdcch2test1 :) select sum(metric_id), sum(version), max(ds), sum(length(denominator_vec)), sum(length(numerator_vec)), sum(length(only_exist_vec)) from mmexpt_lakehouse.metric_vector_8616 where ds = '2023-12-08 00:00:00' and metric_id=1000 and version = 2023120700 and bucket_num < 32 ngs_input_format_parquet_filter_push_down=0;

SELECT
  sum(metric_id),
  sum(version),
  max(ds),
  sum(length(denominator_vec)),
  sum(length(numerator_vec)),
  sum(length(only_exist_vec))
FROM mmexpt_lakehouse.metric_vector_8616
WHERE (ds = '2023-12-08 00:00:00') AND (metric_id = 1000) AND (version = 2023120700) AND (bucket_num < 32)

Query id: ddd94496-1466-4774-b8d7-838fc0a94893

sum(metric_id) | sum(version) | max(ds) | sum(length(denominator_vec)) | sum(length(numerator_vec)) | sum(length(only_exist_vec))
-----|-----|-----|-----|-----|-----
1 | 1 | 2023-12-08 00:00:00 | 1 | 1 | 1

1 row in set. Elapsed: 1.132 sec.
```

# Parquet 读取内存优化

- 过去，一次 Parquet IO 最少读取一个 Row group 数据，一个 Row group 包含百万行数据，当其中某一行很大时（比如大 String），容易造成 OOM

```
mmdcchschedul-shard6-mmdcchschedul6 :) WITH 1000000 AS row_group_size
SELECT
    byteSize(extra_info_) AS record_size,
    formatReadableSize(record_size * row_group_size)
FROM [redacted]
WHERE hour_ = '2023-09-20 00:00:00'
LIMIT 1;

WITH 1000000 AS row_group_size
SELECT
    byteSize(extra_info_) AS record_size,
    formatReadableSize(record_size * row_group_size)
FROM [redacted]
WHERE hour_ = '2023-09-20 00:00:00'
LIMIT 1

Query id: 2eda17c0-7707-4283-814b-610476149eb9

+-----+-----+
| record_size | formatReadableSize(multiply(byteSize(extra_info_), row_group_size)) |
+-----+-----+
| 4260        | 3.97 GiB                                                             |
+-----+-----+

1 row in set. Elapsed: 3.440 sec. Processed 8.19 thousand rows, 34.46 MB (2.38 thousand rows/s.)
```

一个 IO 线程读取一个 Row Group 需要 2 GB 以上内存，24 线程 -> 48 GB 以上内存

解决方法: [Parquet Batch Reader](#), 从一次读取一个 Row group, 变成一次读取一个指定 batch 大小数据, 默认 batch 大小为 8192

# 全表扫描限制

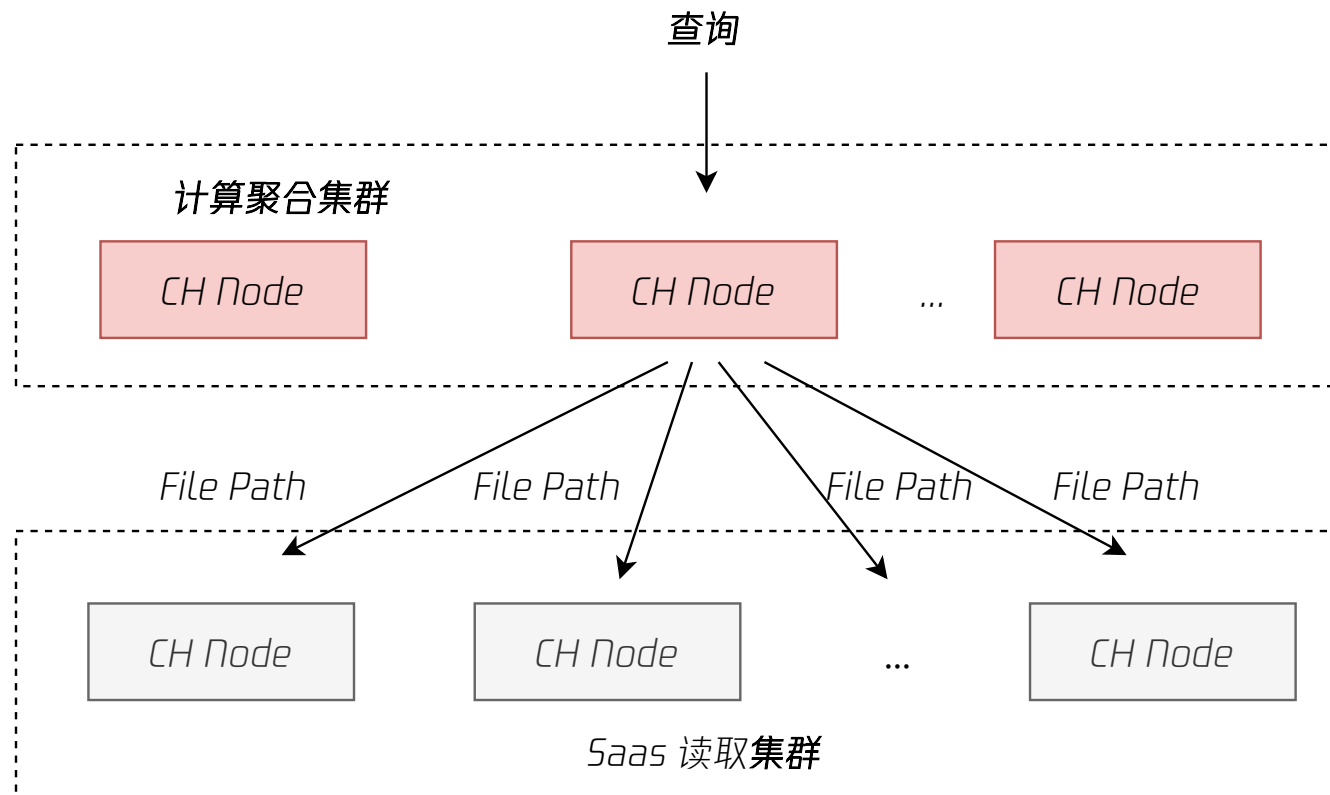
- 湖上表数据保留周期长 [半年到一年以上]，数据量大
- 全表扫描会造成 IO、网络、计算等资源浪费，影响系统稳定性
  - ✓ **解决方法**：在 *Plan* 阶段检查查询是否包含分区剪裁谓词，如果没有，则禁止查询，同时提示用户表中的分区字段是什么：

```
mmdcchschedule-shard6-mmdcchschedule6 :) select * from [redacted];  
  
SELECT *  
FROM [redacted]  
  
Query id: 53a8244b-4a75-4bb1-a8b8-a0672d032be7  
  
0 rows in set. Elapsed: 1.178 sec.  
  
Received exception from server (version 52.8.6):  
Code: 210. DB::Exception: Received from [redacted]:21671. DB::Exception: Iceberg Service http request returned not OK  
tion: Query on iceberg table must have at least one partition predicate. predicate columns: ; partition columns: hour_  
ddress: [redacted].. (NETWORK_ERROR)
```

# 读取集群、聚合集群分离

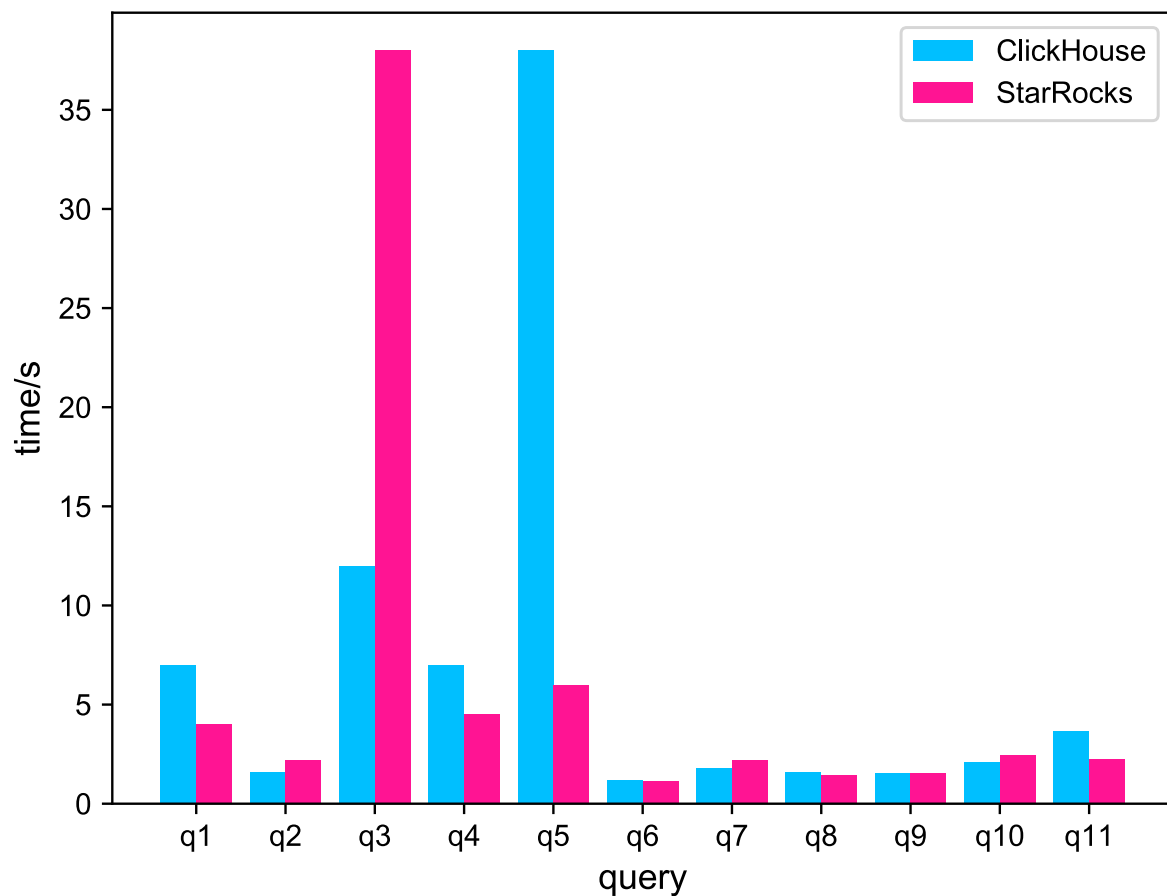
- 资源隔离, *Iceberg* 的读取会耗费大量的 IO、网络带宽, 直接在业务集群读取可能影响集群稳定性
- 按照 *ClickHouse* 两阶段执行将读取预聚合 [阶段一] 与聚合合并 [阶段二] 分开在不同集群执行

- ✓ 阶段一: SaaS 集群, 无状态, 可弹性扩展
- ✓ 阶段二: 业务集群, 完成最终结果聚合计算



# 性能评估

- 与 *StarRocks* , 在具体业务查询上对比湖读取计算的性能



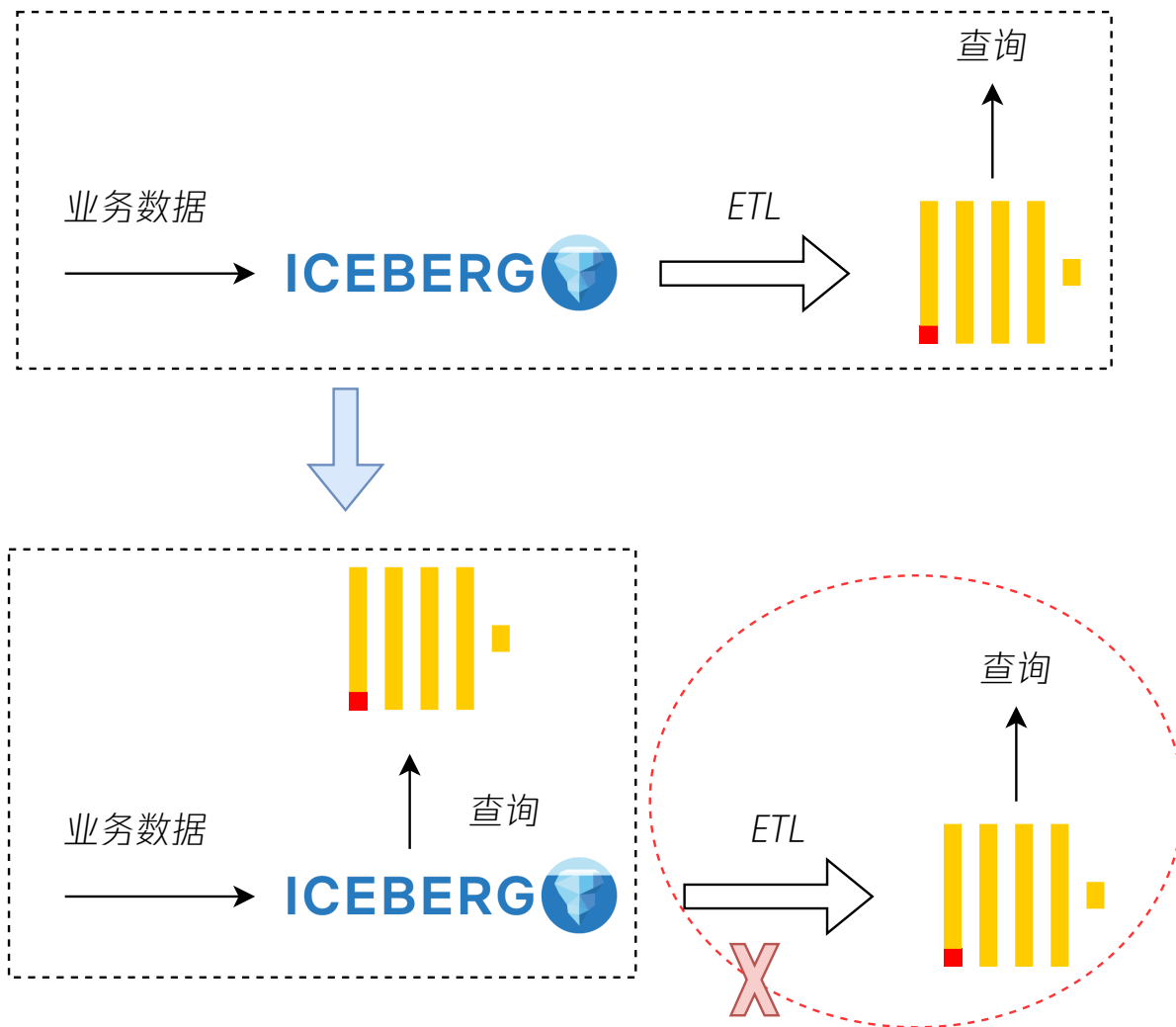
结论:

- ✓ 湖查询性能基本对齐 *StarRocks*
- ✓ 读湖稳定性, 资源占用优于 *StarRocks* [得益于上述优化]
- ✓ Q3: 大数据量 *count distinct* 显著优于 *StarRocks*
- ✓ Q5: 窗口函数计算, CH 无完整 MPP 能力, 单节点计算  
➤ [PR#56827](#)

业务实践

# 湖上数据即席查询

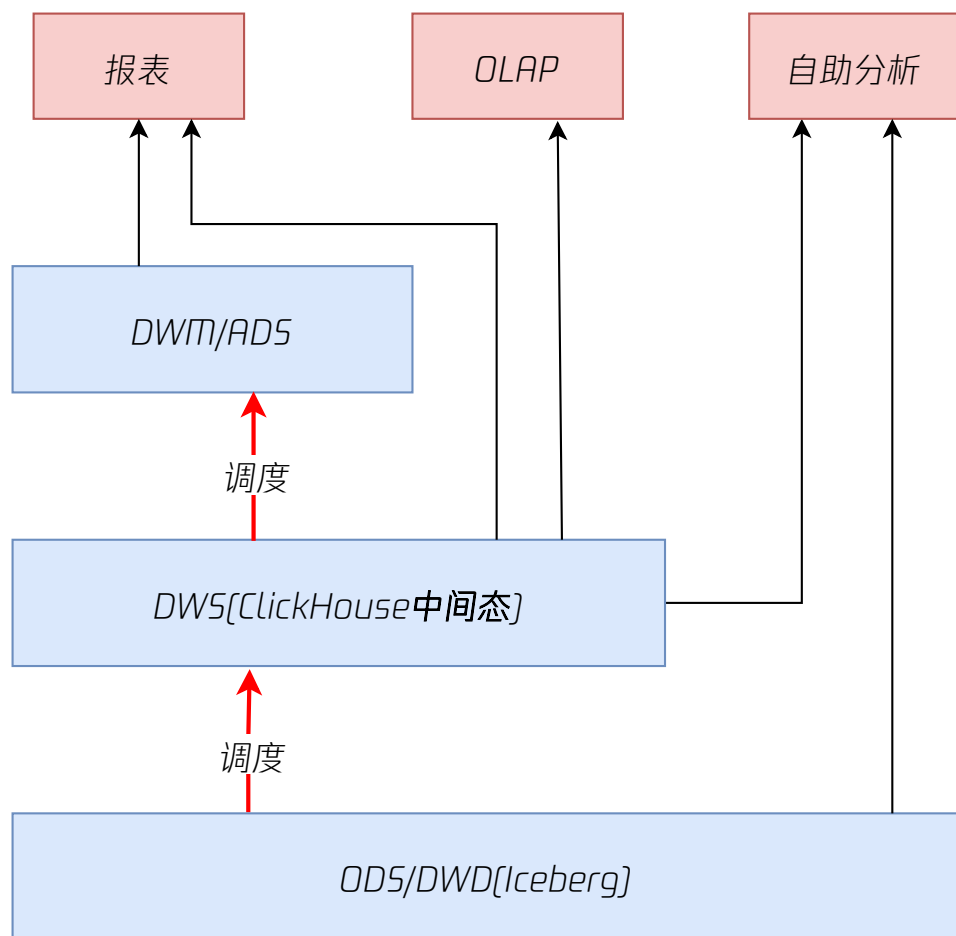
- 过去：先通过 ETL 导入数据  
ClickHouse 再进行查询  
➤ 流程繁琐，资源浪费
- 当前：ClickHouse 作为计算引擎，  
直接分析湖上数据，无需 ETL
  - ✓ 避免数据孤岛
  - ✓ 消除繁琐 ETL
  - ✓ 消除冗余存储
  - ✓ 性能有保障：Presto 3~6倍  
以上



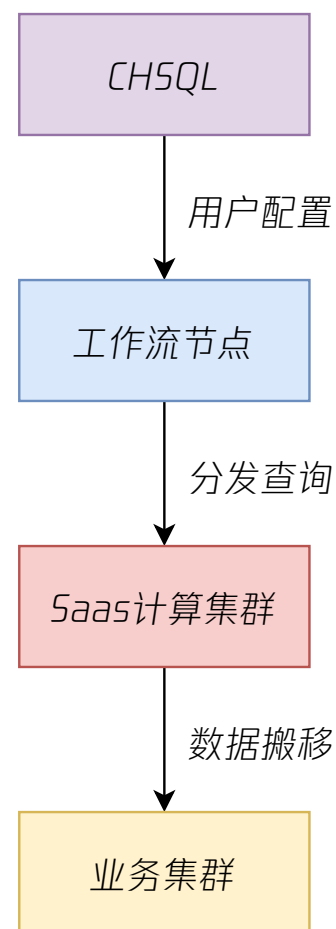


# 湖上数据加工

- 湖上建仓: DWS、ADS 层加速, 读湖 -> 加工 -> 写内表
  - 未来: 读湖 -> 加工 -> 写湖, 写湖能力支持



- 加工 workflow 平台化
  - ✓ 资源隔离
  - ✓ 写入幂等
  - ✓ SQL 并发执行
  - ✓ 失败自动重试/取消



# BI 分析



- BI 报表从查询内表切换到直接查询 Iceberg 外表
  - 过去：一份实时数据接入 ClickHouse 中，通过内表查询生成报表
    - ✓ 内表存储成本昂贵 [SSD]，存储周期短
    - ✓ 湖上已有一份用于离线分析的数据，导致两份存储，资源浪费
  - 当前：直接查询湖上的 Iceberg 表
    - ✓ 缓解本地存储压力
    - ✓ 消除冗余存储

# X 实验分析场景优化

- 基于 SQL 的实验因果推断
  - 特点：数据量较小 [亿级别到十亿级别]，“即用即走”
  - 过去：先通过 Spark 任务导入数据 [分钟级延迟]，再进行分析，流程繁琐
  - 当前：直接通过 ClickHouse 读湖进行分析，无需导入，简化流程
- 离线指标 - 画像按需分析
  - 过去：全部 Spark 预计算好，
    - 每天消耗几十万核资源
    - 90% 的预计算的结果最终是不需要的
  - 基于湖读取：
    - ✓ 用户按需计算，灵活，节省资源
    - ✓ 数据按照 bucket 排序好，支持 parquet 谓词下推，能够剪裁一个数量级的 IO，延迟有保障

未来规划

# 未来规划

- 湖上加工场景探索：当前，读湖 -> 加工 -> 写本地 , 读湖 -> 加工 -> 写湖 
  - ✓ 写湖能力支持
  - ✓ 读湖稳定性，计算资源弹性扩展
  - ✓ MPP 探索，应对复杂计算
- *Iceberg Catalog* 支持：无需创建外库

## Interfaces & External Data

Support for Iceberg Data Catalog

Support for Hive-style partitioning

Explicit queries in external tables

Even simpler data upload

HTTP API for simple query construction

Unification of data lake and file-like functions

[ClickHouse 2024 Roadmap](#)

*End*