

字节跳动数据湖索引 演进

耿筱喻 字节跳动数据平台大数据工程师



目录 CONTENT

01 HUDI 索引介绍

03 字节数据湖索引演进

02 问题与挑战

04 未来规划

01

HUDI 索引介绍



传统数仓数据更新

在传统 Hive 数仓的场景下，数据更新方式为：

增量 Join 全量 -> 覆盖历史分区

- 读全部文件
- 更新全部文件
- Join

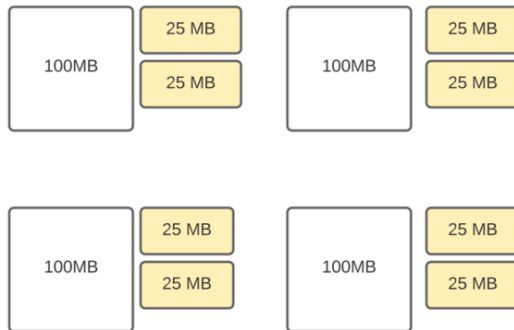
Hudi 索引作用

更新数据可以快速被定位
到对应的 File Group

- 避免读取不必要文件
- 避免更新不必要文件
- 全局 Join -> Local Join

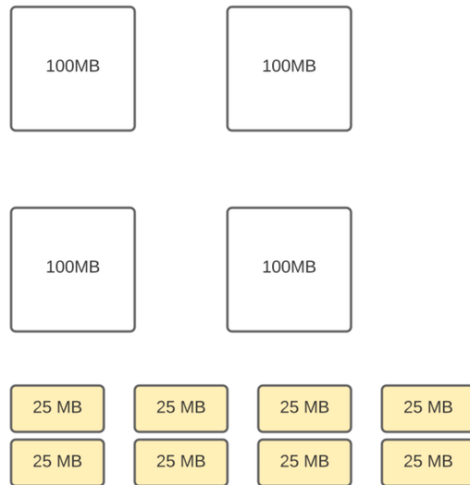
With Index

(Each file is merged against ONLY updates for that file)



Without Index

(Each file is merged against ALL updates)



Hudi 索引类型

| | 特点 | 支持的版本 |
|---------------------------|--|-------|
| Bloom Filter Index | <ul style="list-style-type: none">轻量级，默认的索引方式索引信息存储在数据文件的 Footer 中仅支持 Spark 写入 | 0.6+ |
| HBase Index | <ul style="list-style-type: none">重量级，依赖 HBase索引信息存储在 HBaseFlink / Spark 均支持 | 0.6+ |
| Bucket Index | <ul style="list-style-type: none">轻量级索引信息通过文件名感知Flink / Spark 均支持 | 0.11+ |
| State | <ul style="list-style-type: none">轻量级索引信息存储在 State 中仅支持 Flink 写入 | 0.7+ |

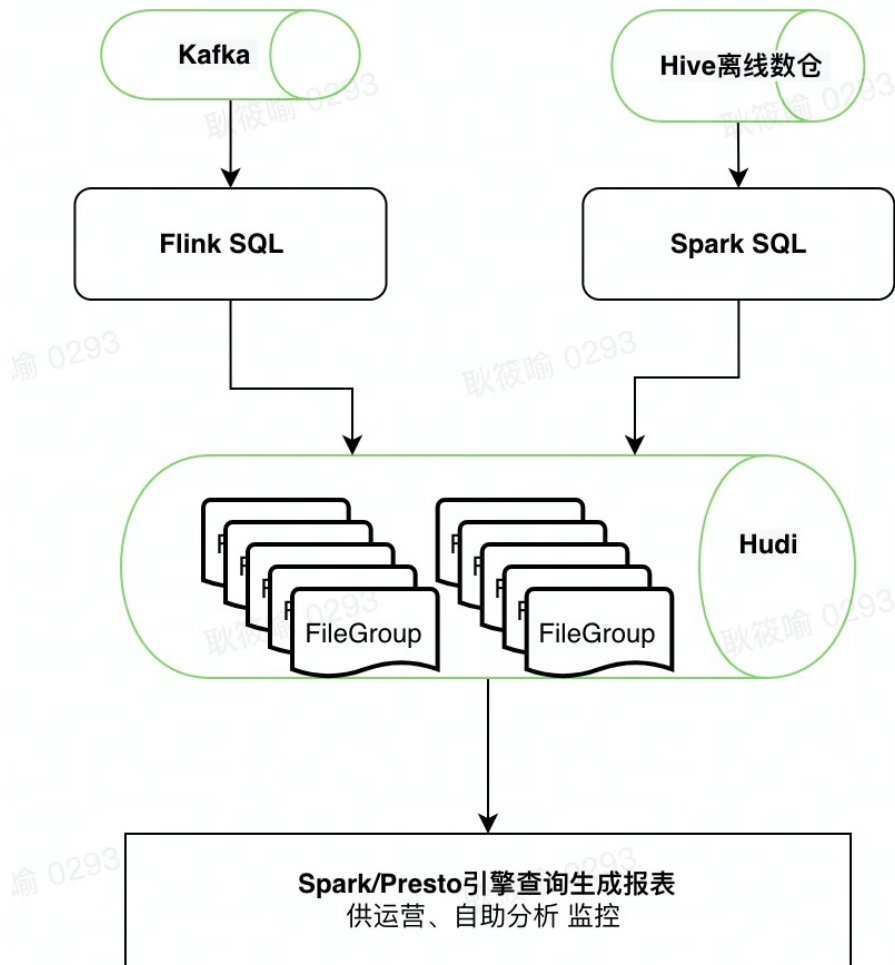
02

问题与挑战



数据入湖的业务场景

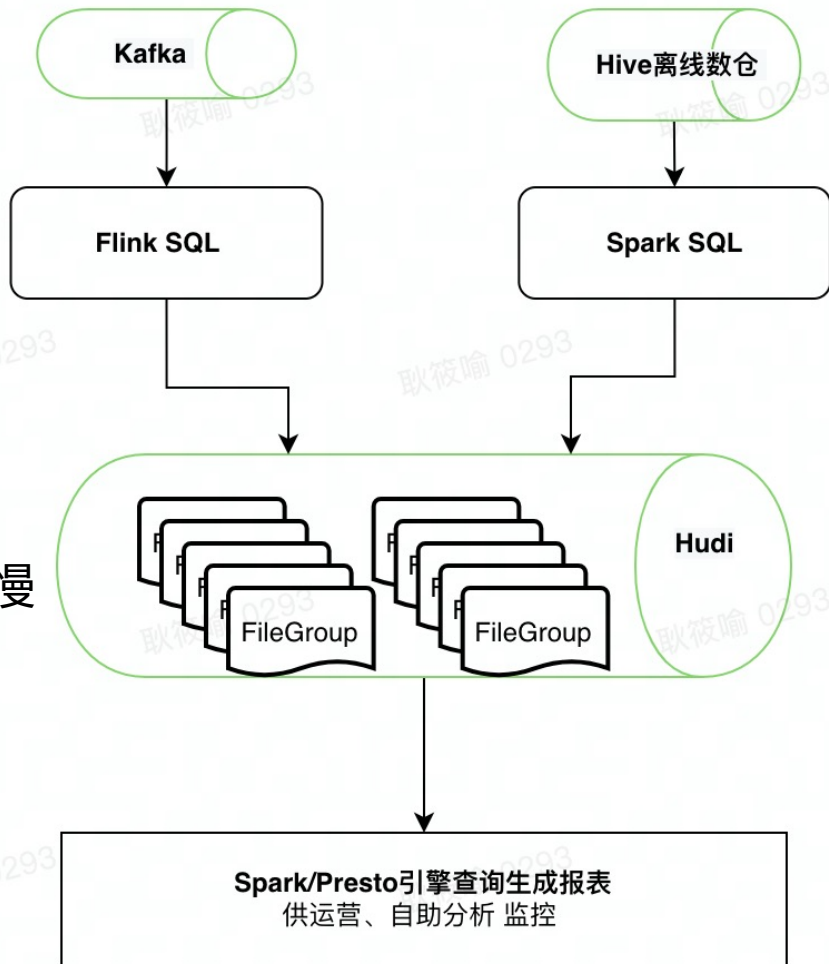
- 实时 Upsert
- 小时/天级批量 BackFill (Upsert)



数据入湖的业务场景

- 单分区 40000 个File Group
- 30 TB 数据量
- 5 千亿条记录数

Bloom Filter 性能非常差，入湖速度慢



Hudi 索引类型

| | 特点 | 支持的版本 |
|---------------------------|--|-------|
| Bloom Filter Index | <ul style="list-style-type: none">轻量级，默认的索引方式索引信息存储在数据文件的 Footer 中仅支持 Spark 写入 | 0.6+ |
| HBase Index | <ul style="list-style-type: none">重量级，依赖 HBase索引信息存储在 HBaseFlink / Spark 均支持 | 0.6+ |
| State | <ul style="list-style-type: none">轻量级索引信息存储在 State 中仅支持 Flink 写入 | 0.7+ |

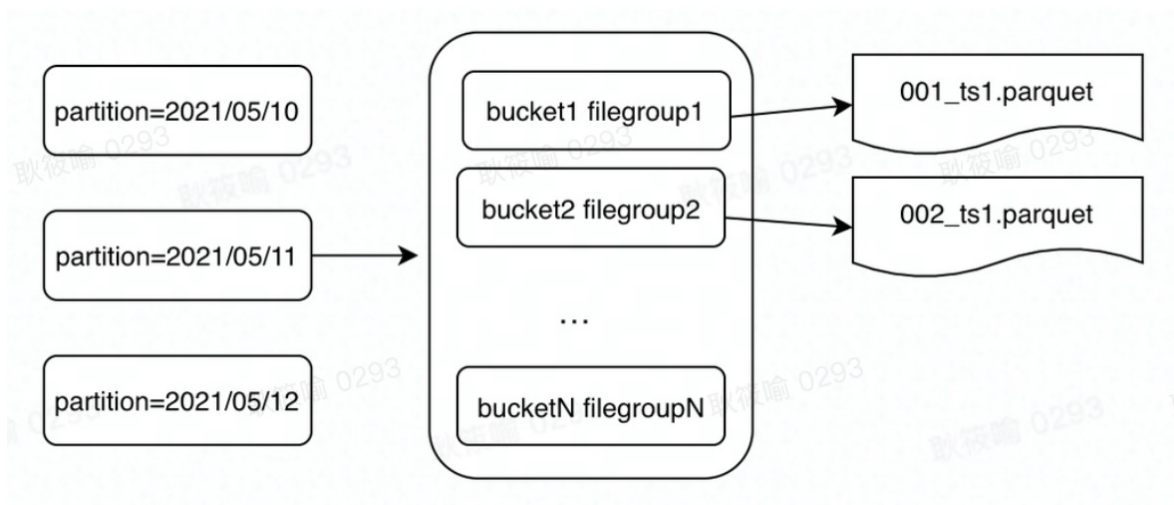
03

字节数据湖索引演进

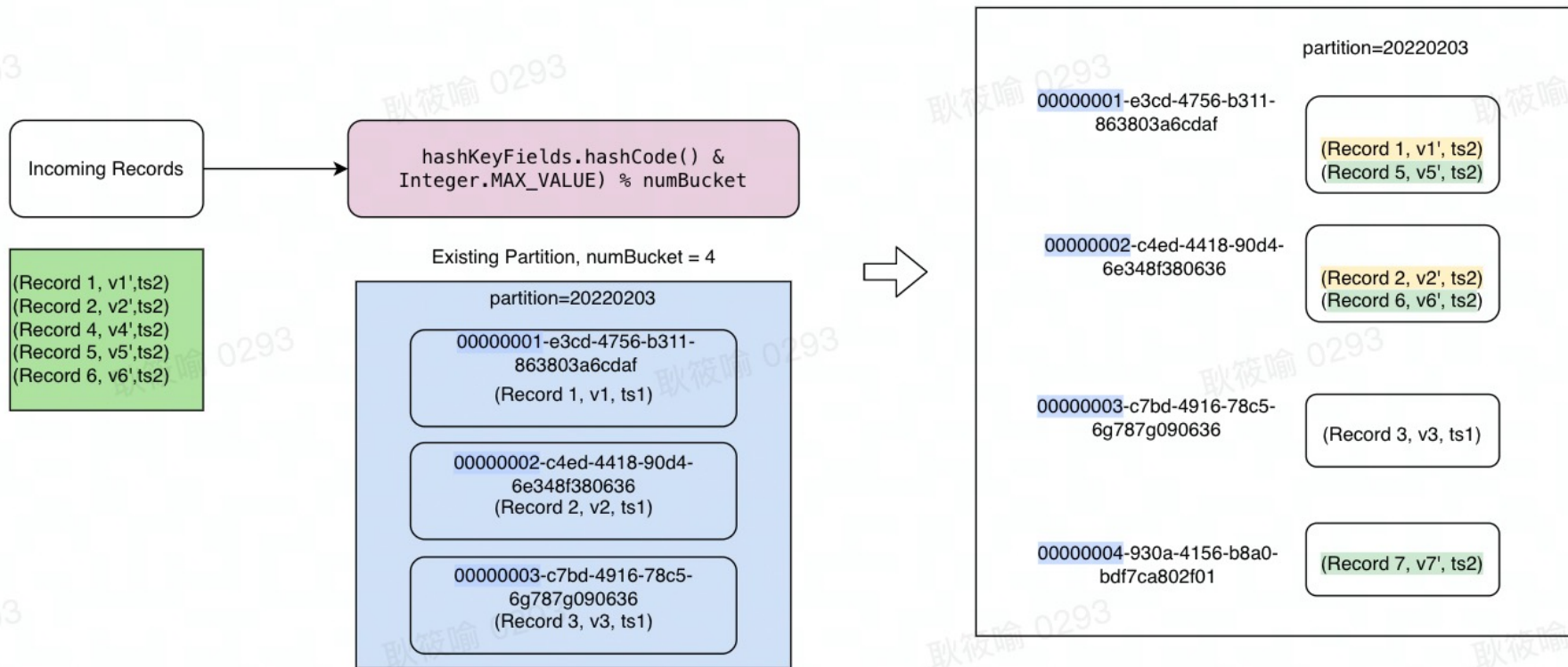


Bucket Index – 基本原理

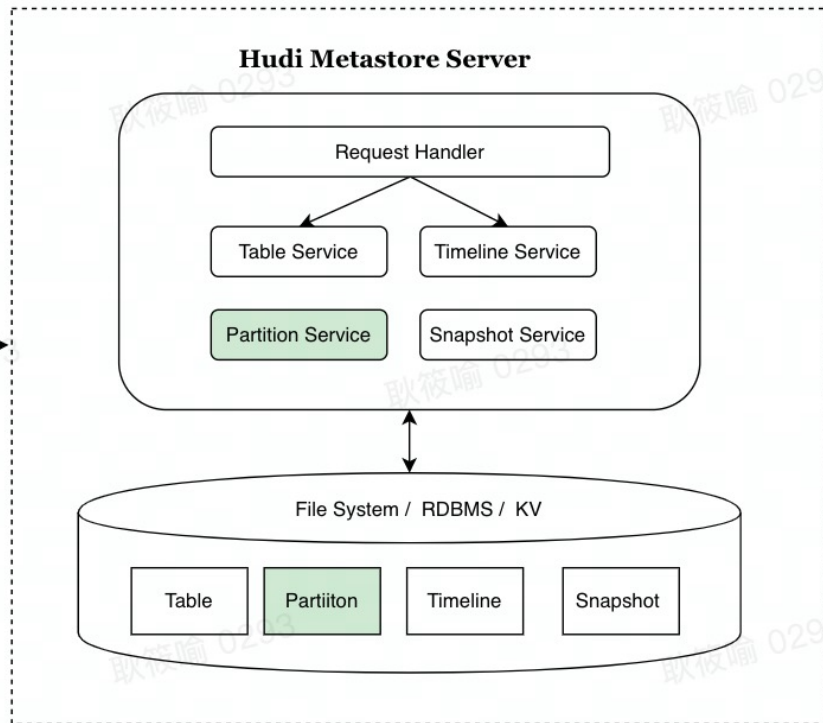
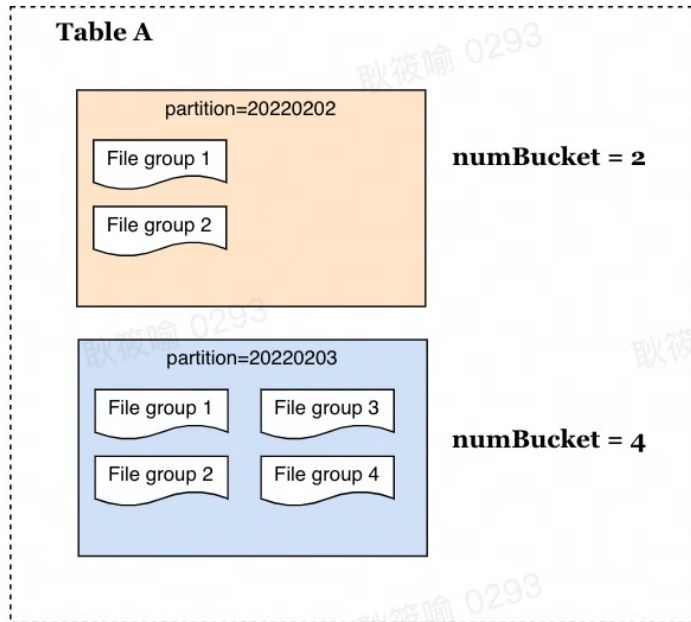
- 一种基于哈希的索引
- 逻辑层面提供 $\text{Key} \leftrightarrow \text{BucketId} \leftrightarrow \text{File GroupId}$ 的映射关系



Bucket Index – 写入流程

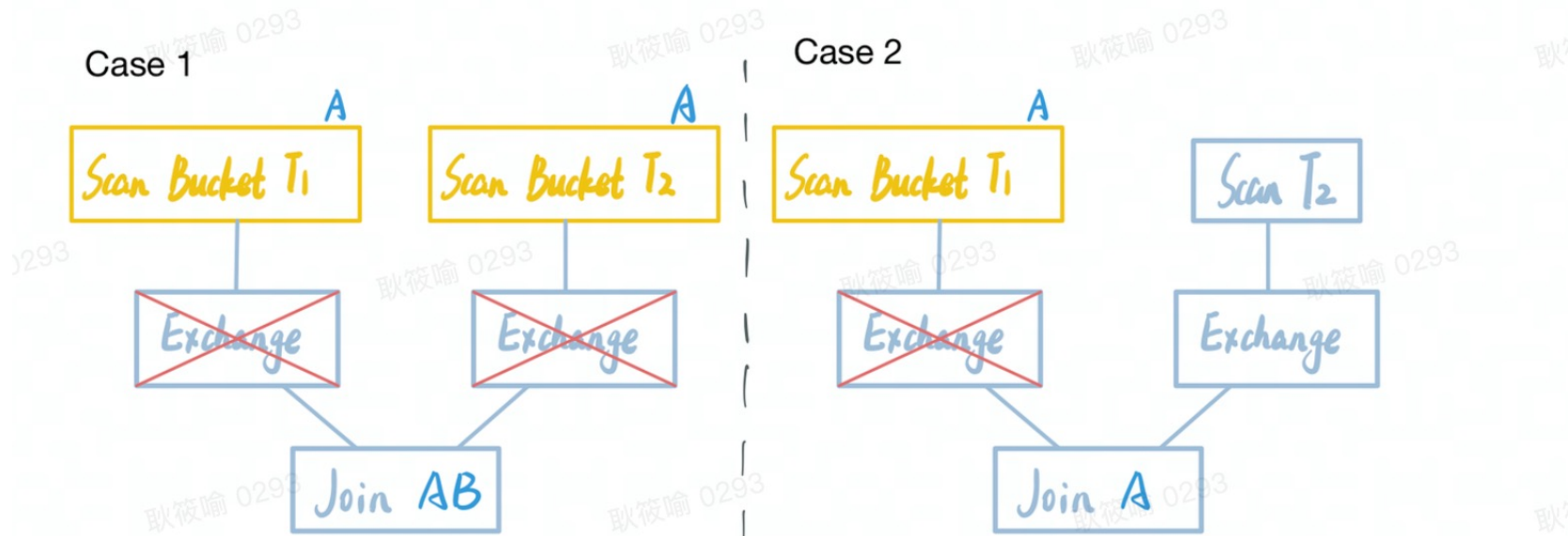


Bucket Index – 分区级 Bucket



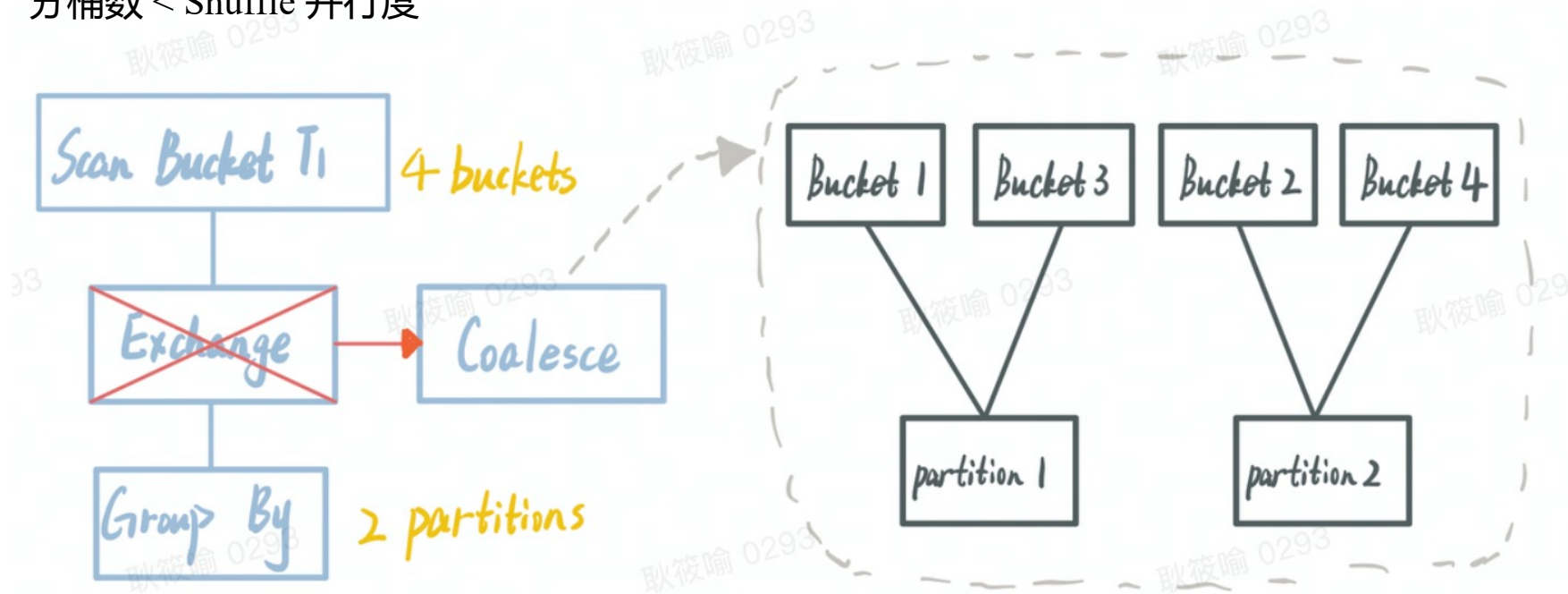
Bucket Index – 查询优化

- Case1: 表 T1/2 按 A 列分桶, AB 列(超集) Join
- Case2: 表 T1按 A 列分桶, A 列 Join



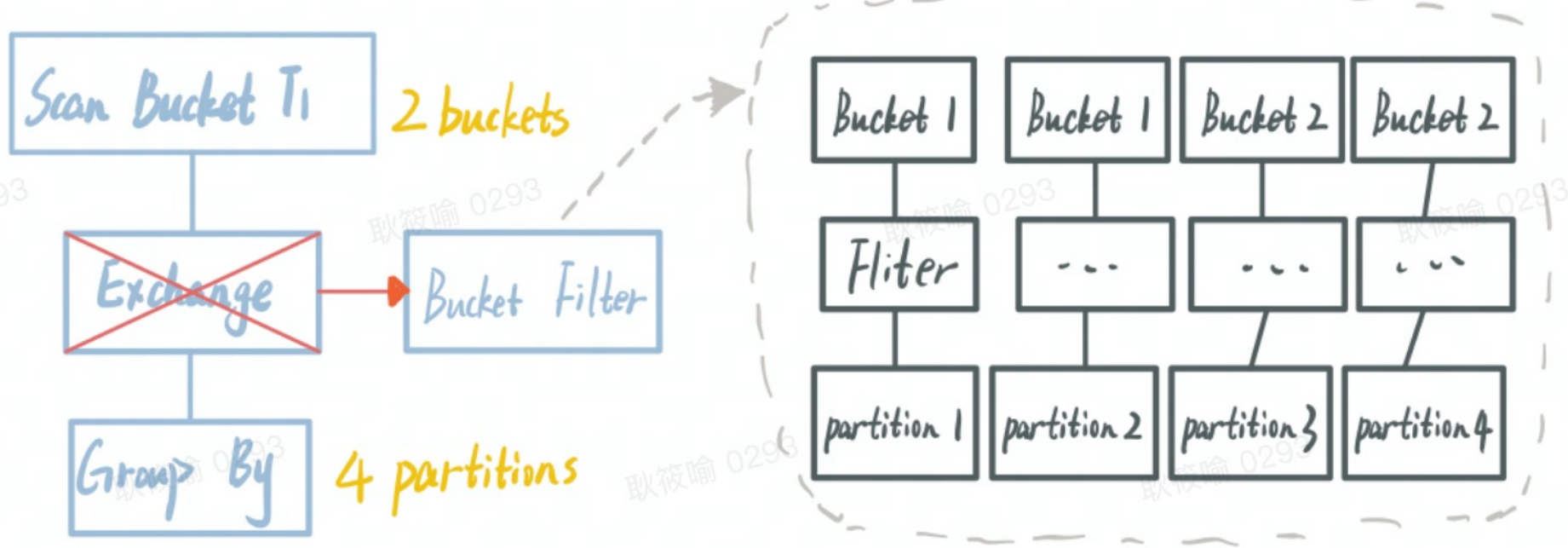
Bucket Index – 查询优化 Coalesce

- 分桶数与 Shuffle 并行度成倍数关系
- 分桶数 < Shuffle 并行度



Bucket Index – 查询优化 Multiple Input Read

- 分桶数与 Shuffle 并行度成倍数关系
- 分桶数 > Shuffle 并行度



Bucket Index – 查询优化 Bucket Pruning

Table 1 (8 buckets)
bucketed by colA

TableScan
(Read files: 1)



Filter(colA == 1)



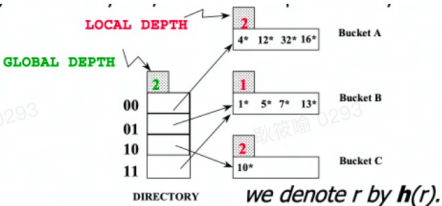
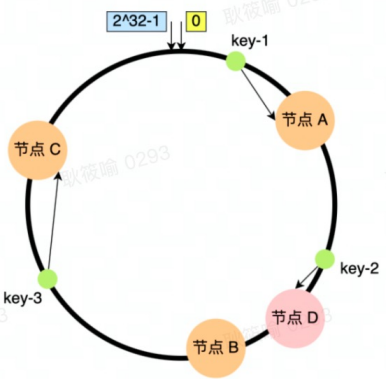
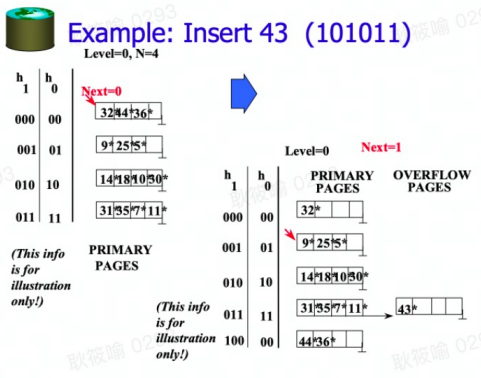
Other Operations

- 点查列 == 分桶列
- 查全部文件 -> 查单个文件

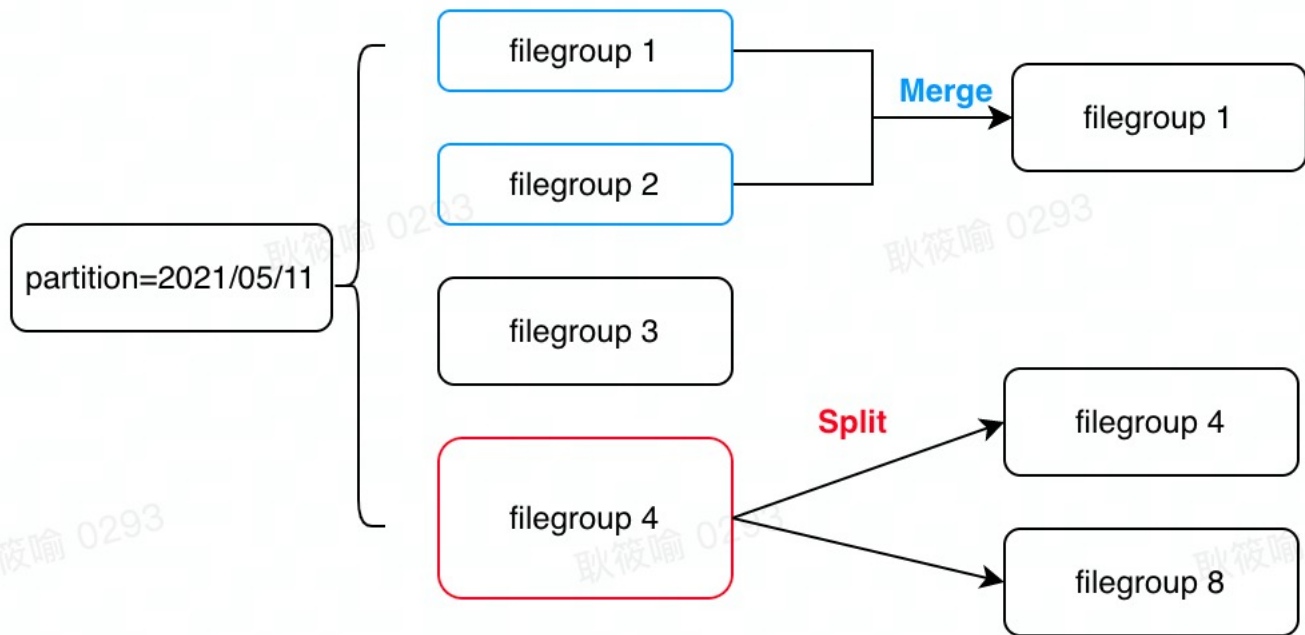
Bucket Index -> Extensible Bucket Index

Bucket Index 可扩展性差？

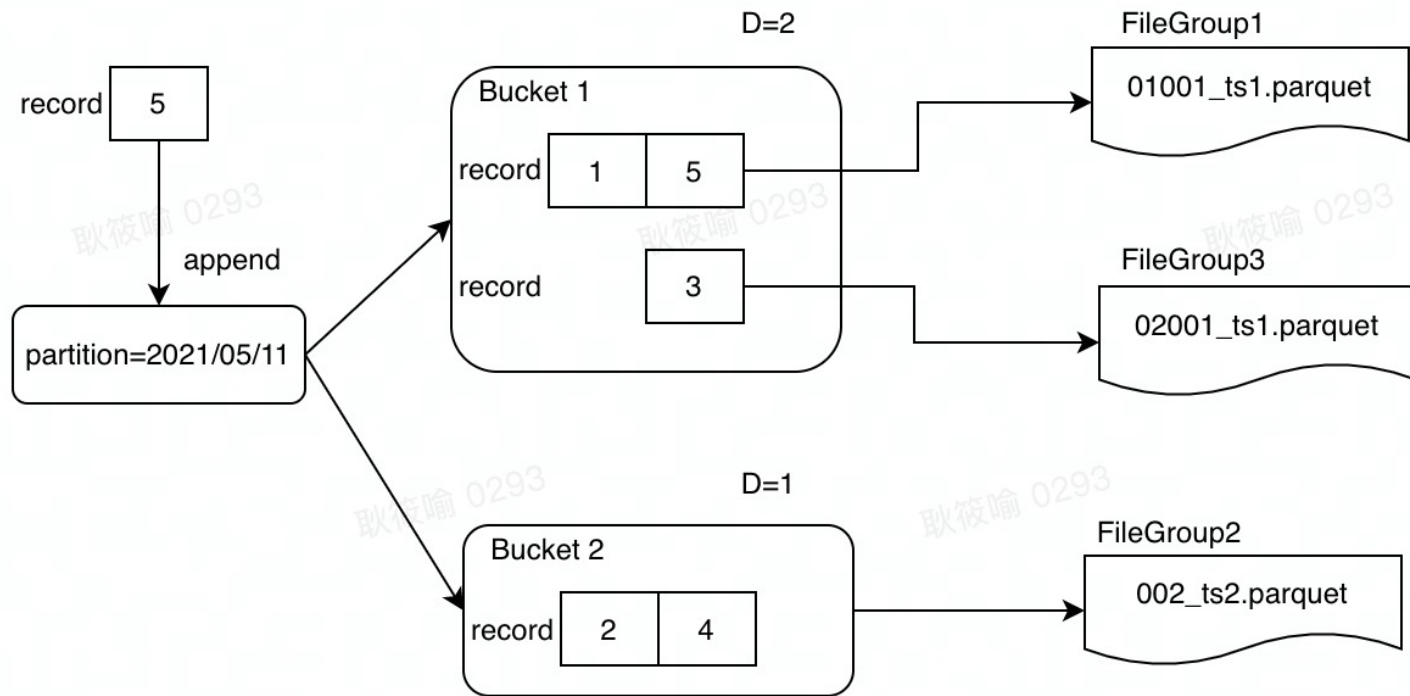
Bucket Index -> Extensible Bucket Index

| | Extensible Hash | Consistent Hash | Linear Hash |
|------|---|--|--|
| 目标问题 | 分桶写满后的扩展性问题 | 修改分桶数后数据重分布 | 能接受小部分桶数据溢出过多 |
| 核心思想 | 拆分/合并桶 | 首尾连接哈希环，加/删桶数据迁移到后/前节点 | round-robin的方式增加新桶 |
| 具体步骤 | <ul style="list-style-type: none"> 全局分桶G+局部分桶L，通过字典维护全局分桶到局部分桶之间的映射关系($G \geq L$) 全局分桶倍数改变  | <ul style="list-style-type: none"> 增加新节点，将部分后继节点的数据迁移到新节点上  | <ul style="list-style-type: none"> round-robin的方式增加新桶  <p>Example: Insert 43 (101011)</p> <p>Level=0, N=4</p> <p>Next=0</p> <p>Level=0 Next=1</p> <p>PRIMARY PAGES</p> <p>OVERFLOW PAGES</p> <p>(This info is for illustration only!)</p> |
| 限制 | 全局分桶G会一直变化，并且每次变化都是 $G/2$ 或者 $G*2$ | 查询无法利用数据分布进行优化 | 只能按序拆桶，实际数据量超限的桶最差需要等待前面N个桶都split之后才能被分割 |
| 优点 | 扩缩容只改变局部分桶的物理分布 | 分桶数改变灵活无限制 | 桶数增长/缩小缓慢，不必维护字典 |

Extensible Bucket Index – 基础原理



Extensible Bucket Index – 基础原理



Non Index – 非主键入湖

比如: 日志入湖

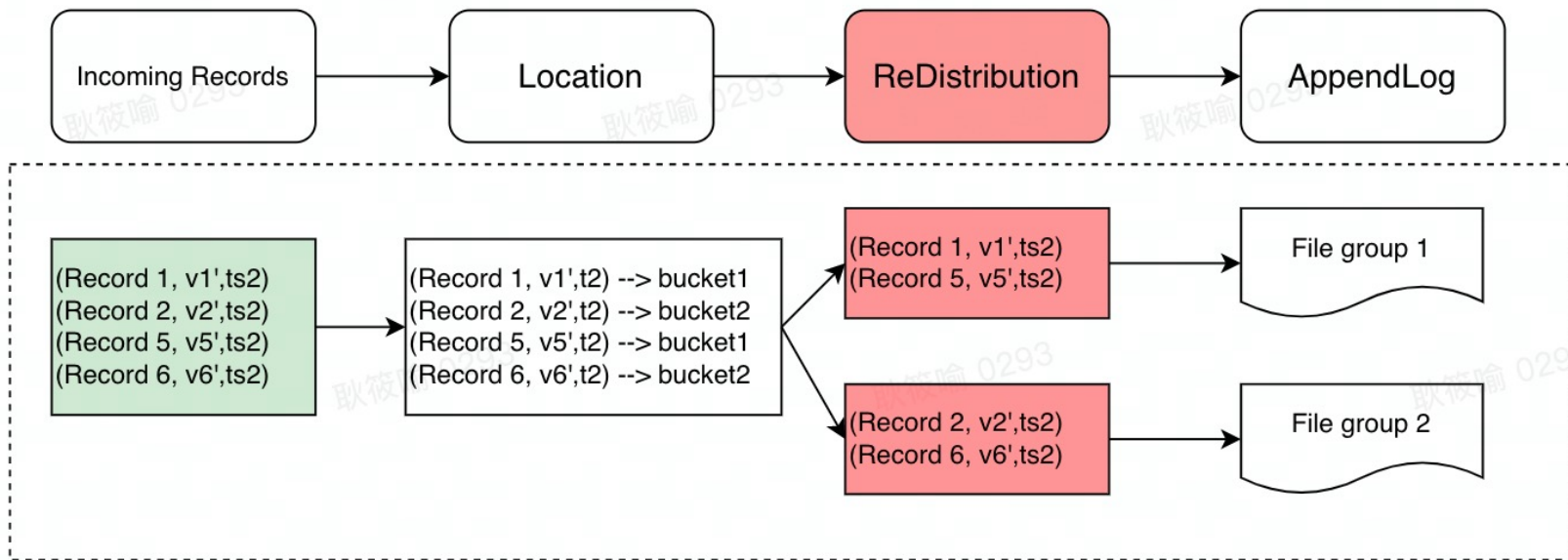
特点：

- UUID -> No Index
- Upsert -> Insert / Append

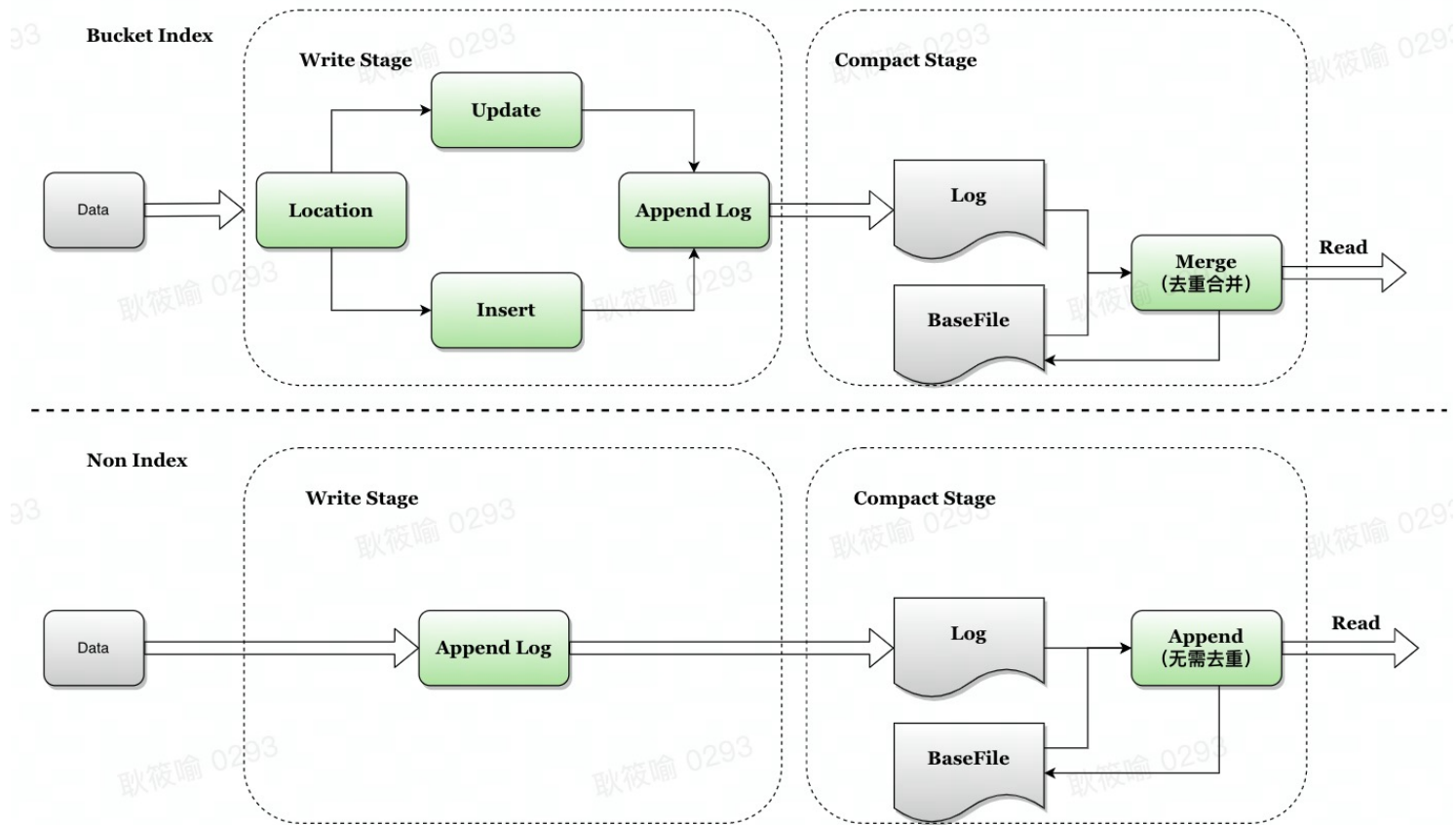
Non Index – 非主键入湖

现有的索引体系必须要 Locate + 数据重新分布

Update



Non Index - 非主键入湖



04

未来规划



未来规划 – 二级索引

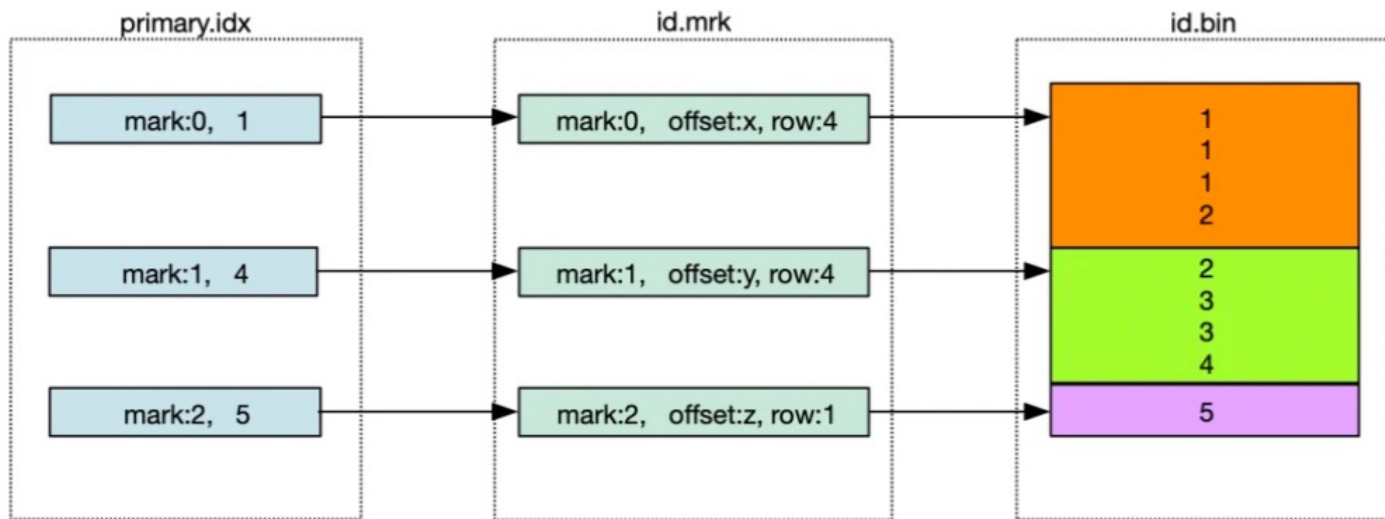
目标：提升非主键列点查性能

基于二级索引列构造 Bloom Filter + Multi-Modal Index

- 支持初始索引异步构建
- 支持事务更新

未来规划 – Range Index

目标：提升点查/范围查询性能



社区工作

- Bucket Index 0.11
- Hudi Metastore [RFC-36] 0.12/1.0
- Table Management Service [RFC-43] 0.12/1.0
- Decouple Avro [RFC-46] 0.12
- Embedded Timeline Server [RFC-50] 1.0
- Flink 支持 Cluster 0.12
- NonIndex [HUDI-2624]

湖仓一体分析LAS



扫码进入官网，即刻了解产品

面向湖仓一体架构的 Serverless 数据处理分析服务，
提供一站式的海量数据存储计算和交互分析能力，
完全兼容 Spark、Presto、Flink 生态，帮助企业轻松完成数据价值洞察。

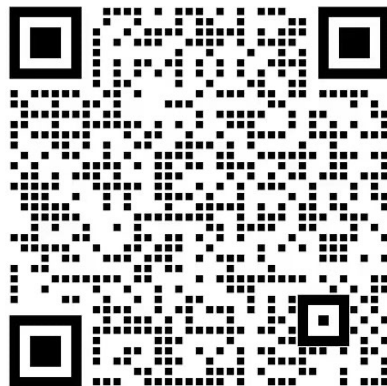
欢迎关注我们

加入我们



扫码关注并回复【招聘】
了解岗位信息

加入官方交流群



更多技术干货、最新活动
加入官方交流群

非常感谢您的观看

 DataFun.

