基于 Apache Doris 的 PB 级实时场景优化与实践

曹俊辉

抖音集团数据库研发工程师

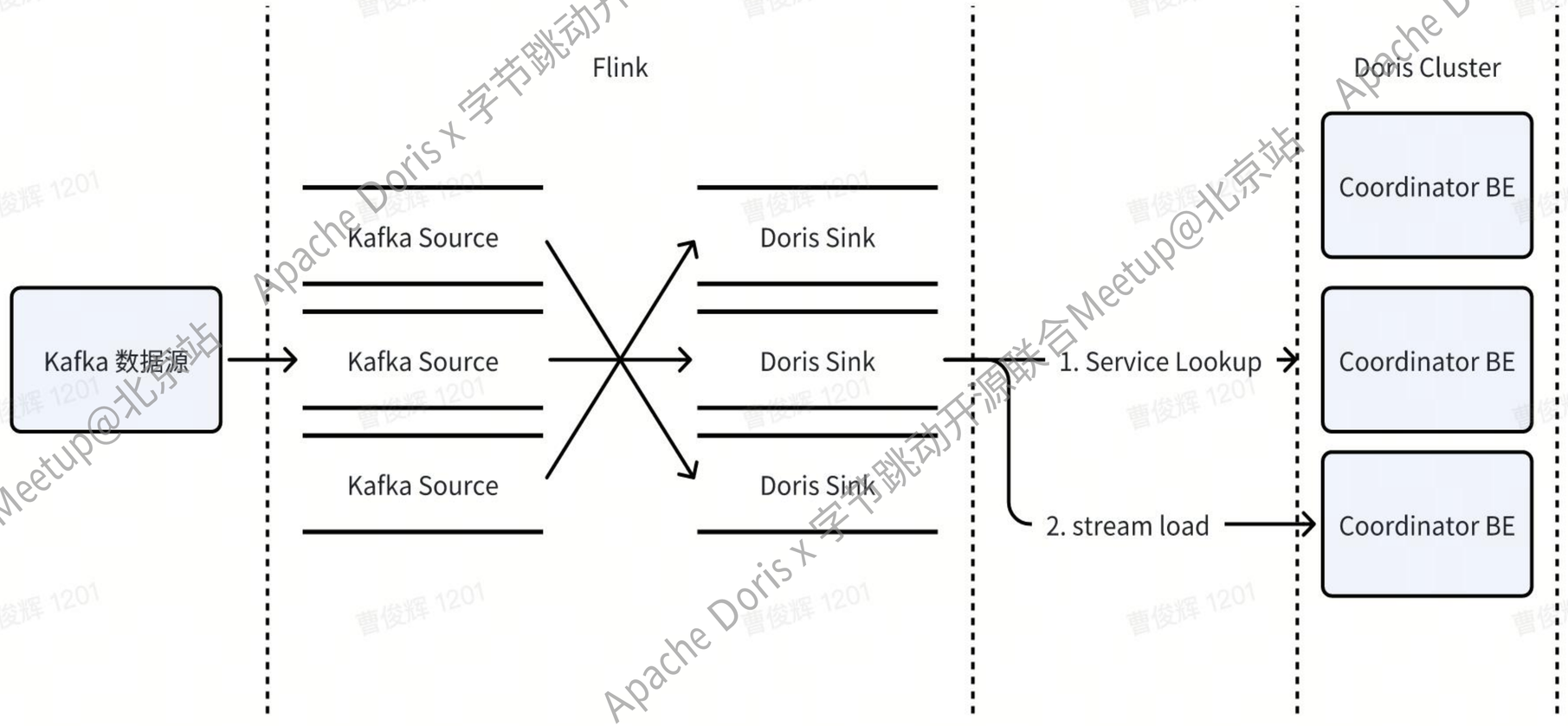
目录

抖音集团对实时写入的性能需求

普通场景		日志 Trace 场景
< 1M row	Topic 流量(每秒峰值)	30M row
<100 TB	平均每天数据量	1 PB
10 s	数据可见性	10 s

抖音集团内部实时写入链路介绍

内部主要使用 Flink + Stream Load 来实现从 Kafka 到 Doris 的写入链路。



日志 Trace 场景表模型与任务设计

```
CREATE TABLE `rtc_data_all_to_doris` (
  `event_key` varchar(4096) NULL COMMENT 'event_key',
  `room_id` varchar(4096) NULL COMMENT 'room_id',
  `device_id` varchar(4096) NULL COMMENT 'device_id',
  `user_id` varchar(4096) NULL COMMENT 'user_id',
  …
   `origin_string` text NULL COMMENT 'origin_string',
  …
) ENGINE = OLAP
DUPLICATE KEY(event_key，room_id,device_id,user_id,…)
PARTITION BY RANGE(`event_date`) (…)
DISTRIBUTED BY RANDOM BUCKETS 2048
PROPERTIES (
"bloom_filter_columns" = "room_id, …"
…
)
```

- 字段数量 1k+，对于不固定的字段使用 JSON
- 使用明细模型
- 单天数据 PB 级，使用小时分区，每个分区 2048 个分桶。
- 业务查询 Pattern 灵活，希望能够灵活添加索引。
- 写入吞吐较高，因此 Flink 任务也需要 2000+ 的 sink 并发。

目录

Doris 1.2 版本上的尝试

相关特性

- SimdJson
- 导入链路向量化
- 单副本导入

性能表现

- 机器规模：3000+ core
- **写入不到 1M row/s**
- 机器 CPU 持续打满
- Compaction 完全跟不上，文件数会堆积到 7、8w (放开限制)

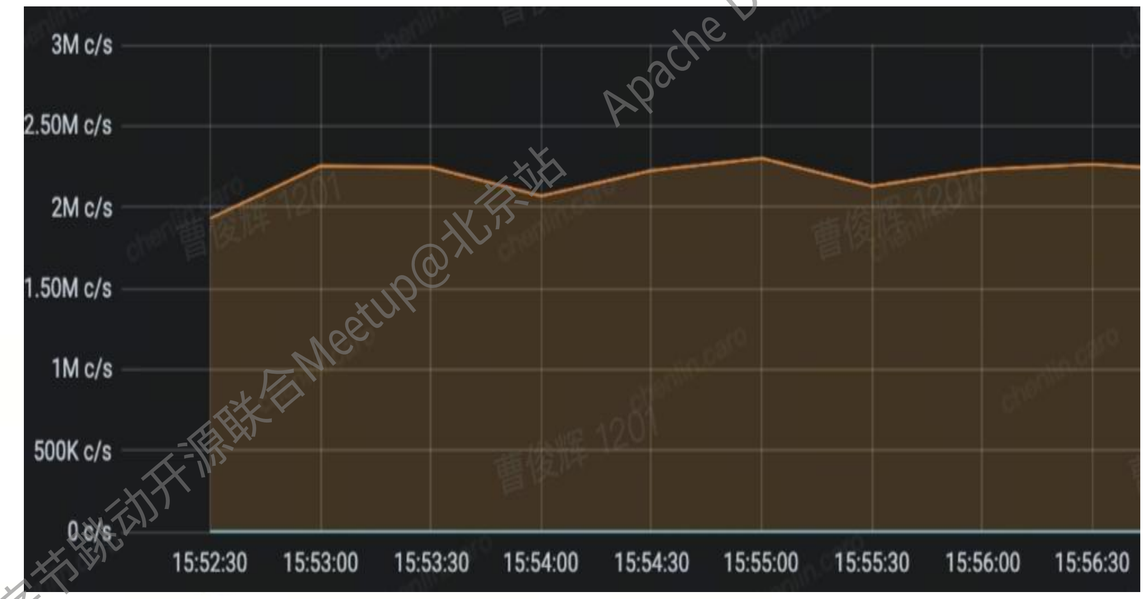
Doris 2.0 版本上的尝试

相关特性

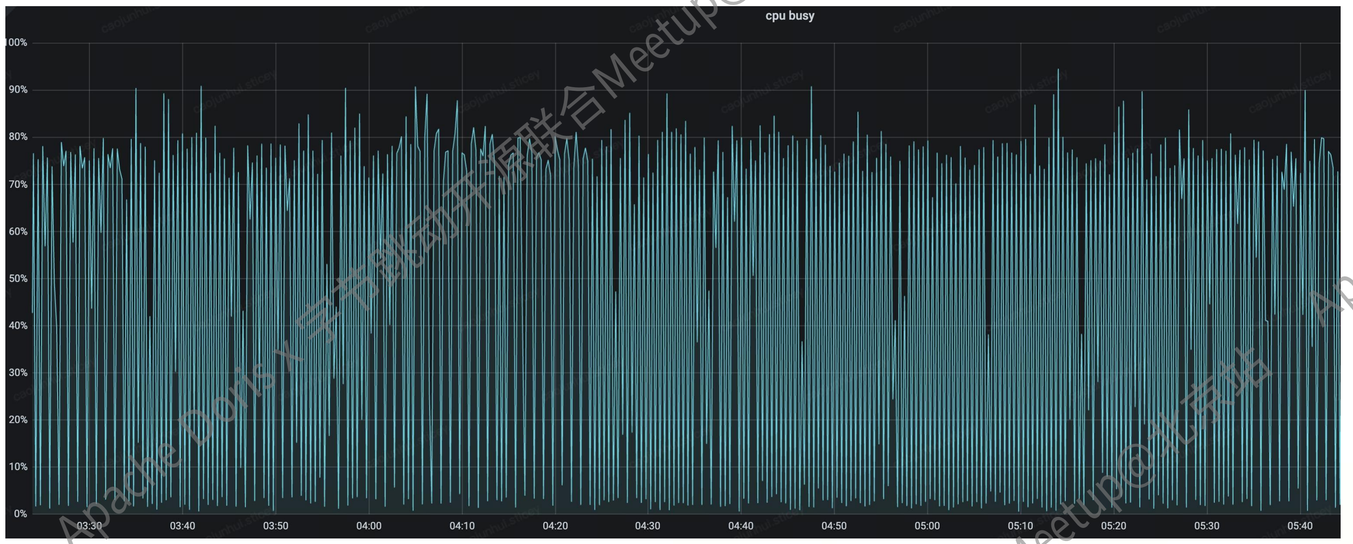
- Vertical Compaction
- Ordered Data Compaction
- Brpc 线程池拆分
- Memtable 延迟排序，减少 CPU 开销

性能表现

- 机器规模：3000+ core
- **写入稳定 2M row/s**
- 机器 CPU 水位较高
- Compaction 水位正常



痛点分析（一）集群 CPU 使用率过高



Compaction

事务粒度较高，文件数堆积较快，Compaction 读开销较大

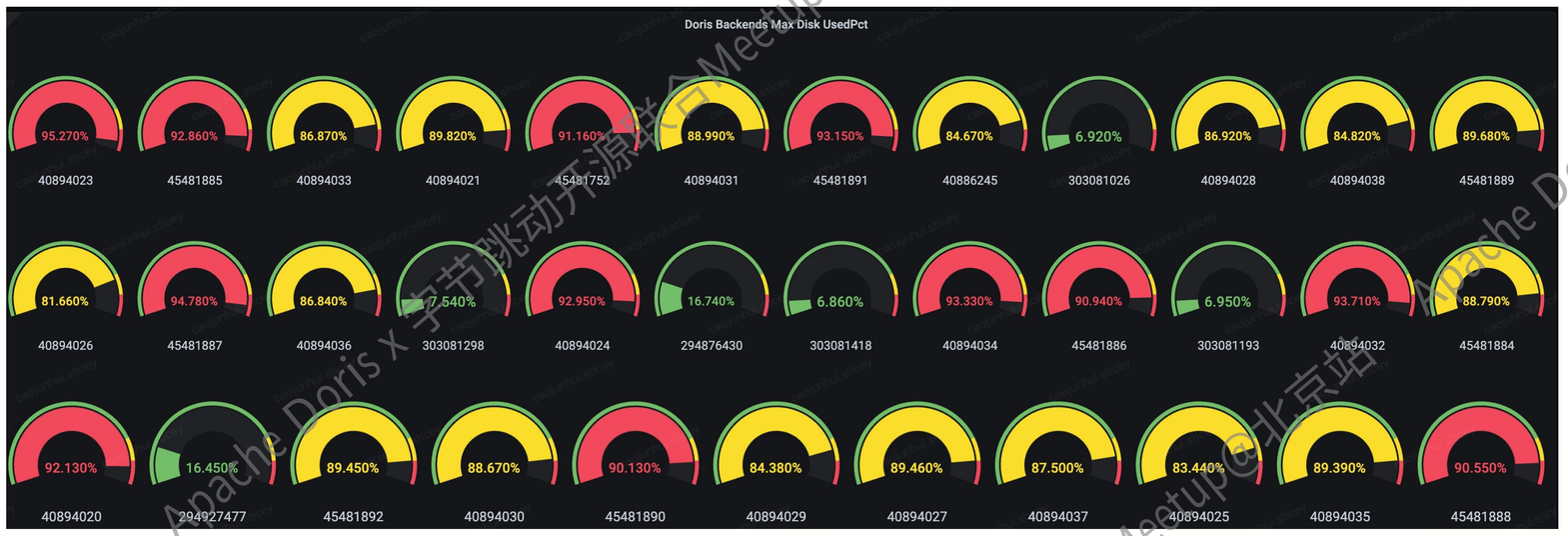
RPC

Coordinator Open/Close 需要发给几乎所有 BE，导致 RPC 开销较重

Coordinator 处理

Coordinator 在 reader、sink 侧处理数据上仍在存在部分没有完成向量化

痛点分析（二）存储成本



存储介质

并发场景下 HDD 写和查性能远不如 SSD
因此需要使用 SSD

JSON 数据

JSON 数据需要保存列名，存储较大
当时 2.0 还不支持 Variant，因此需要用 Protobuf 保存，再通过 Base64 编码写入

磁盘不均

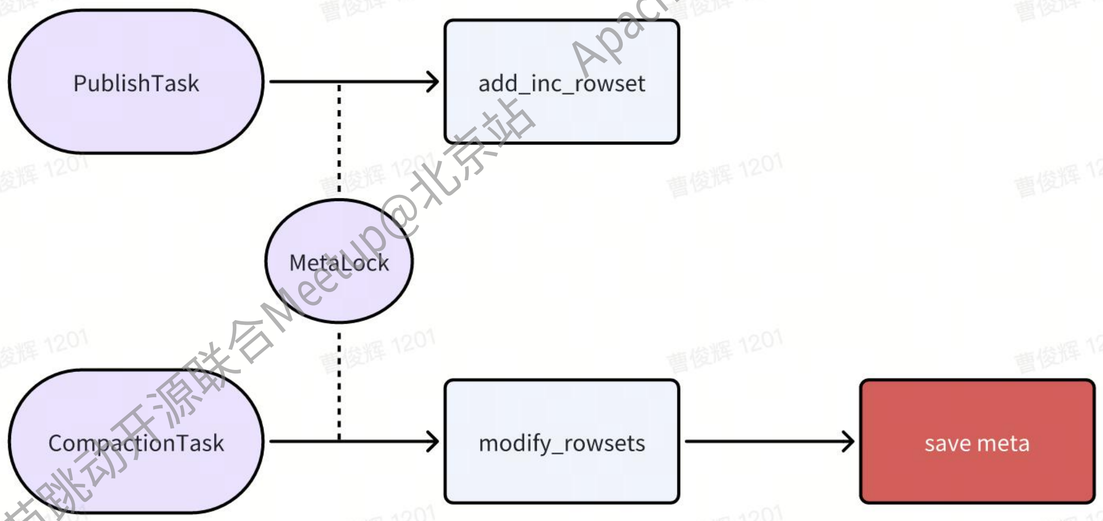
Doris 在节点选盘基于 TabletNum
我们发现实际表现单盘可能会高于平均水位 10%

痛点分析（三）Publish 性能劣化

Publish Task 和 Compaction Task 会产生严重的锁竞争，导致每个 Tablet Publish 平均等待 **10 - 500 ms**，单个导入 Publish 需要等待所有 Tablet 完成，需要花费**数十秒**。

- **请求并发高**：单个导入会涉及到 6 ~ 7 个 Partition 数量的 Publish，集群导入并发在 200 左右，PublishTask 数量级在 2-3 million/s；每个 Tablet 每秒新增 Rowset 200+，CompactionTask 也频繁触发

- **锁竞争**：Publish Task 需要加入新的 Rowset，不需要保存 Tablet Meta ；Compaction Task 需要用新的 Rowset 替换旧的一组 Rowset，需要保存 Tablet Meta

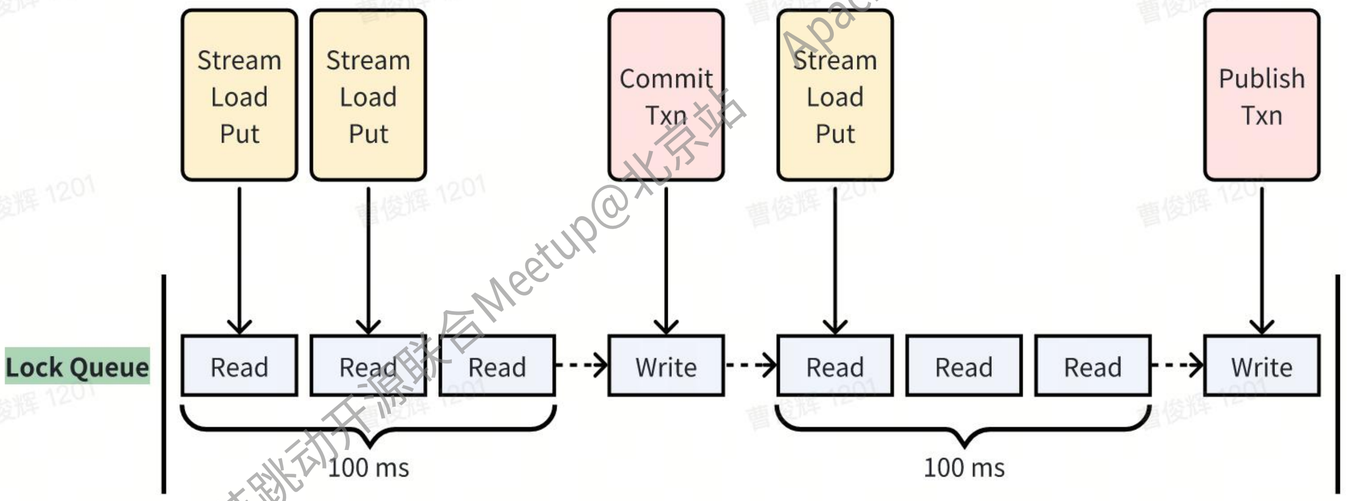


痛点分析（四）Master 节点锁性能不佳

Master 参与导入的步骤主要分为 StreamLoadPut、CommitTxn、PublishTxn，每一步都需要持锁。

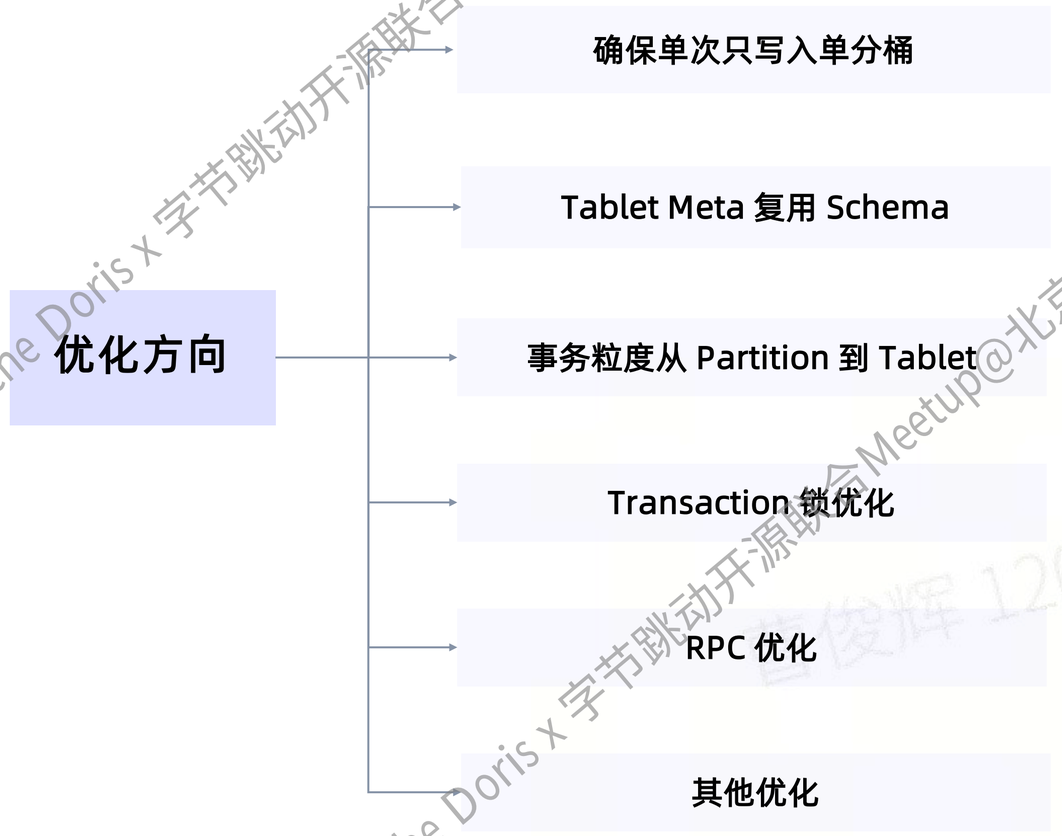
当前 FE 锁粒度主要是 Table，用于保护 Replica 的增删、Replica Version 的更新。

- StreamLoadPut 需要收集所有分区的 Tablet 发送给 Coordinator，耗时会到达 100 ms 左右。
- 读多写多的场景下，ReentrantReadWriteLock 锁排队的机制导致整体性能劣化严重。



目录

基于 Doris 2.0 版本的深度优化



一、确保单次导入只写入单分桶

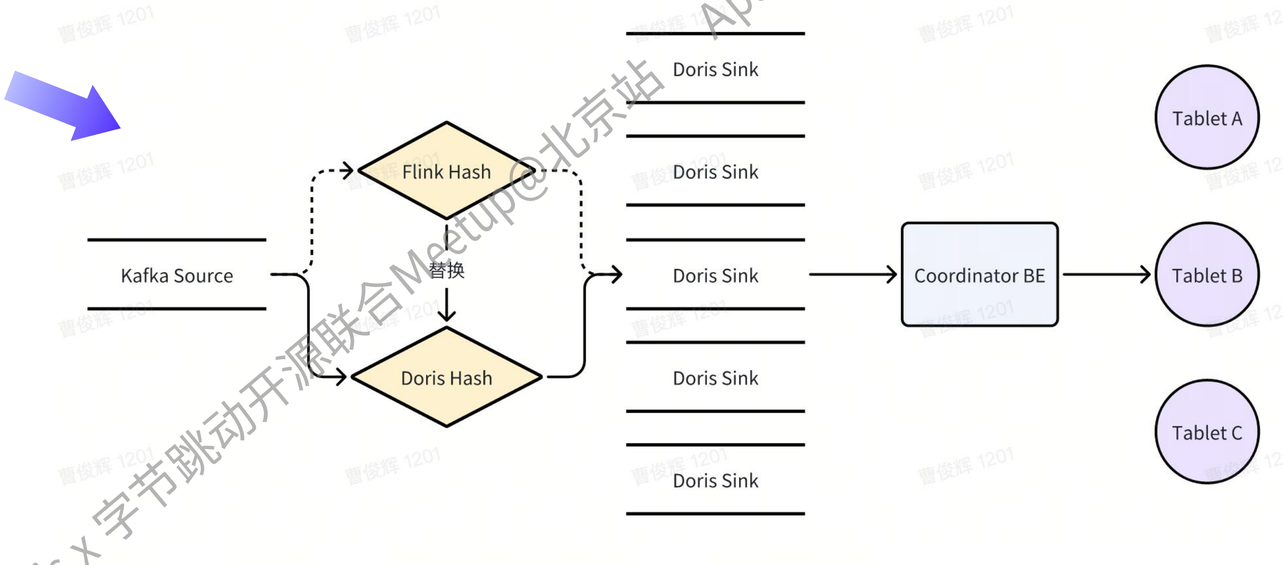
提高 Compaction 和数据写入的效率

需要分桶裁剪优化

- 扩展 Flink 的 Hash 方式，使得数据能够按照 Doris 的分桶逻辑 shuffle 到各个 Sink 中。

不需要分桶裁剪优化

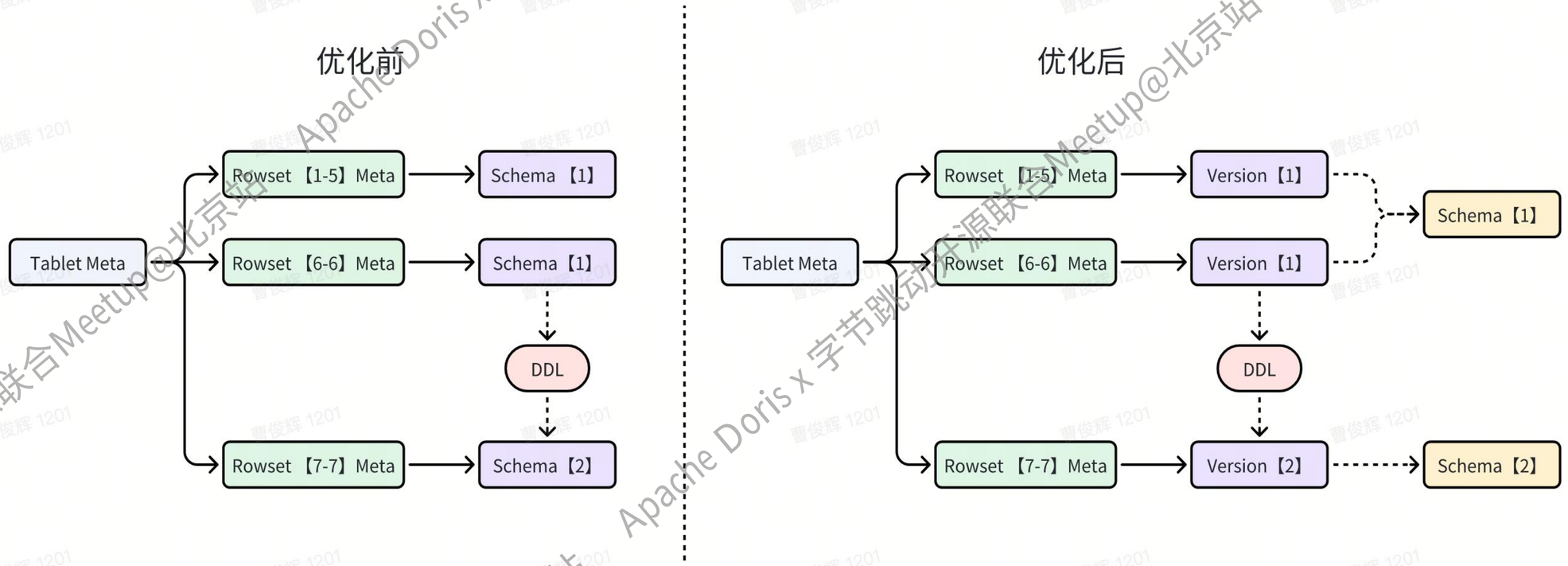
- 使用 Random 模型 + load_to_single_tablet



二、Tablet Meta 复用 Schema

提高 Save Meta 性能、降低锁竞争

- 由于 Light schema change，每个 Rowset 都需要持锁序列化 Schema，开销较大
- Schema 变化通常不频繁，因此可以保存 Schema Version，Schema 单独保存以复用
- 节点重启时通过 Schema Version 加载正确的 Schema



三、事务粒度从 Partition 到 Tablet

为什么需要修改事务粒度

Compaction 频率过高

一个 Tablet 写入会触发整个 Partition 下
所有 Tablet 的 Version + 1
写入频率为 200 ops，因此每秒新增 Rowset 200+

写入 RPC 资源占用高

每次写入需要发送 Open、
Close 请求到所有的节点上

Publish 频率过高

写入频率为 200 ops
单次导入命中 6-7 个 Partition (2048 分桶)
Publish Task 每秒需要下发 2-3 million 个

Tablet MVCC 事务模型

写入侧

导入链路：Node Channel ->Delta Writer

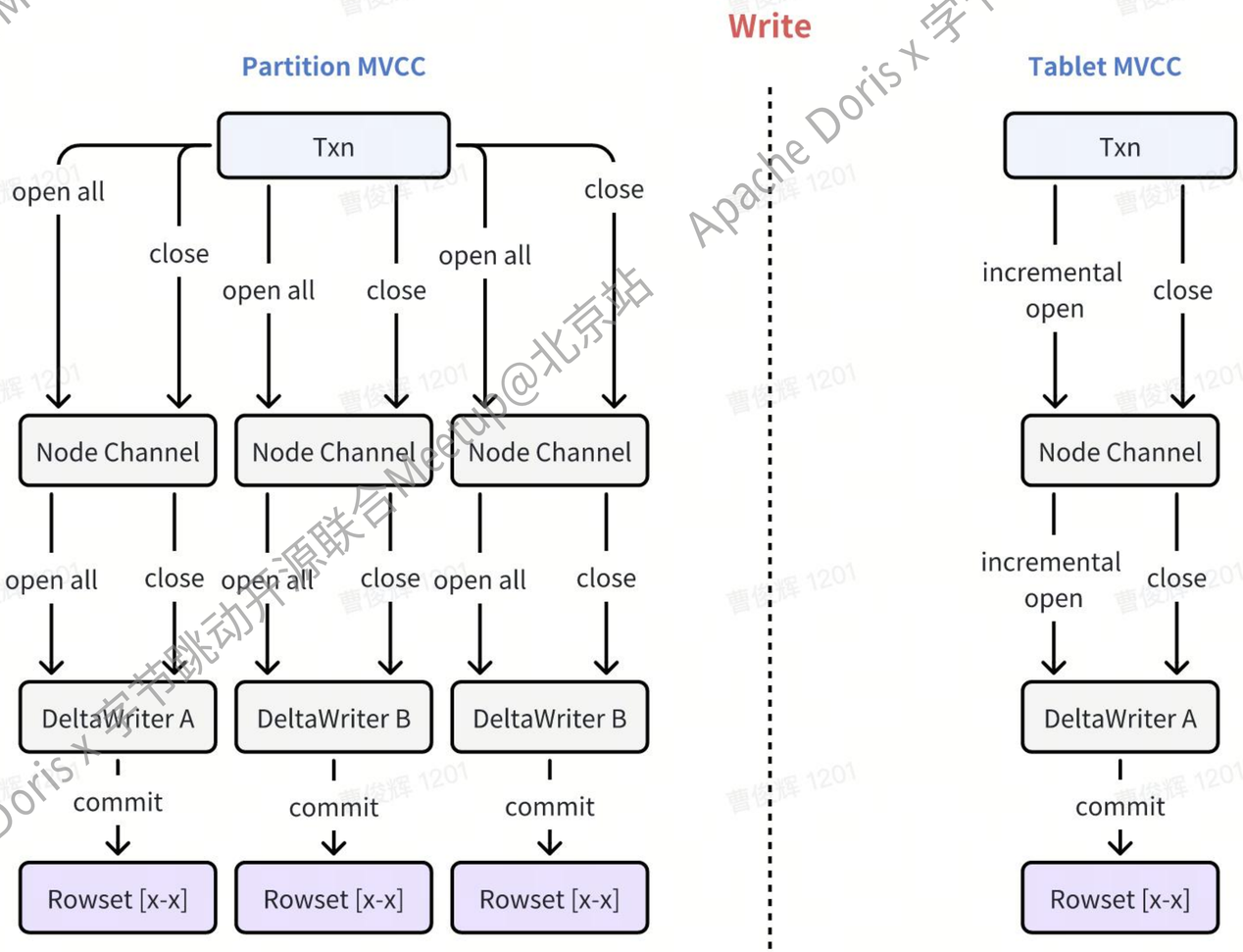
- Node Channel：对应 BE，Coordinator 分发数据到这里
- Delta Writer：对应 Tablet，写入新数据

Partition MVCC

需要打开所有 Tablet，因此需要打开很多 Writer。
这里 Writer 会根据是否同 partition 有写入来 commit

Tablet MVCC

路由到对应 Tablet 才会打开对应的 Writer
Node Channel 只有参与写入的节点



Tablet MVCC 事务模型

Master FE 侧

Doris 的 MVCC 机制主要通过 Publish 实现

FE 根据 Partition / Tablet Version 来确定读取的数据版本

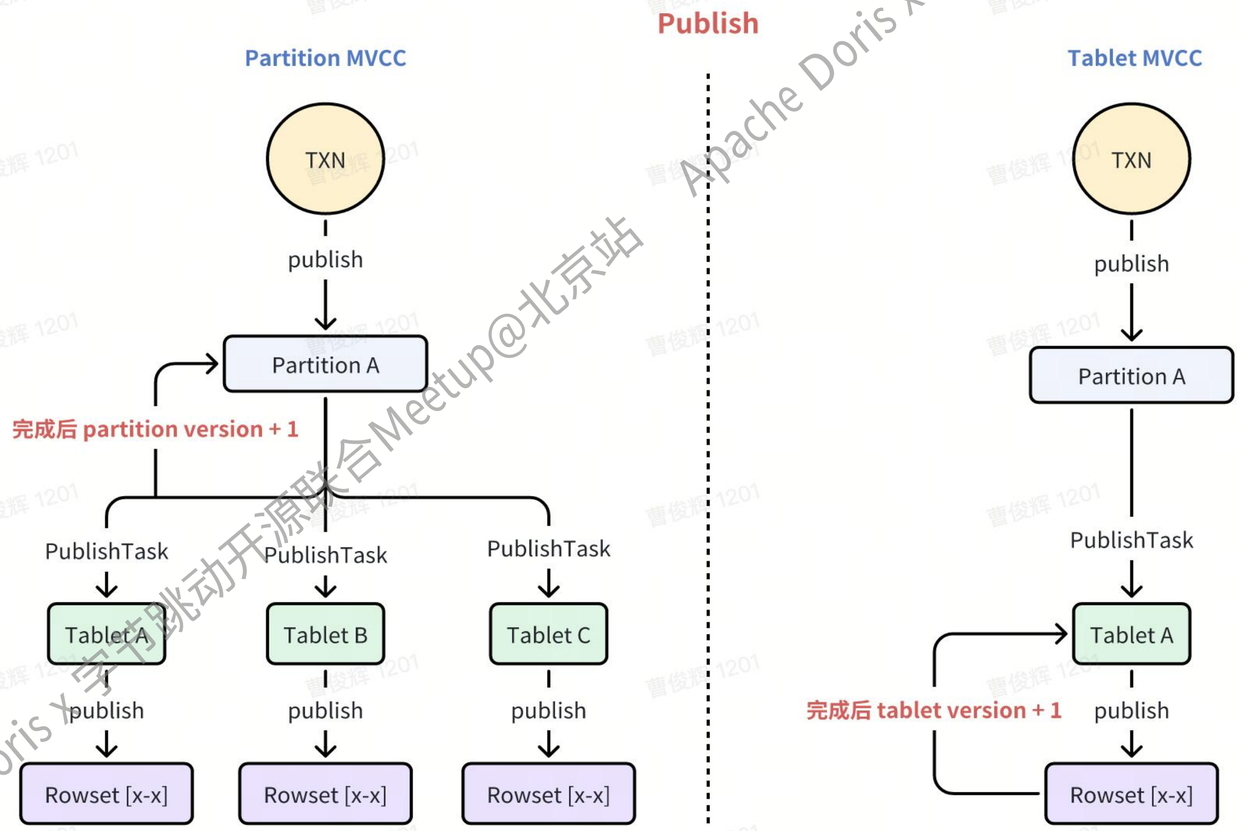
1. Master 在 BE 写入完成后，会下发 Publish 任务
2. Master 确认大多数 Tablet 可见后，会更新可查询 Version

Partition MVCC

需要将该 Partition 下所有 Tablet 都进行 Publish
大多数 Tablet 可见后，会更新 Partition 的 VisibleVersion

Tablet MVCC

路由到对应 Tablet 才会打开对应的 Writer
只需要 Publish 参与写入的 Tablet
大多数 Tablet 可见后，会更新每个 Tablet 的 VisibleVersion



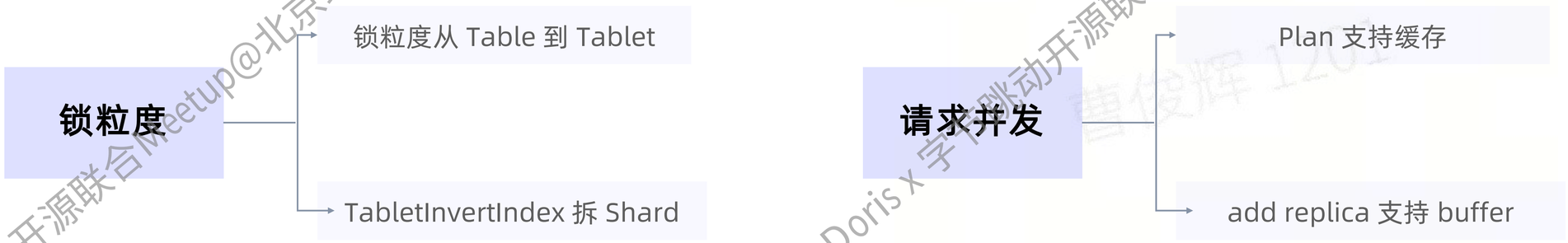
四、Transaction 锁优化

为什么要优化？

使用 Tablet MVCC 模型后，集群资源使用率大幅降低，写入并发和吞吐逐渐上升。

但 FE 出现了严重的锁瓶颈，**Plan、Commit、Publish 经常需要等待几十秒。**

优化方向



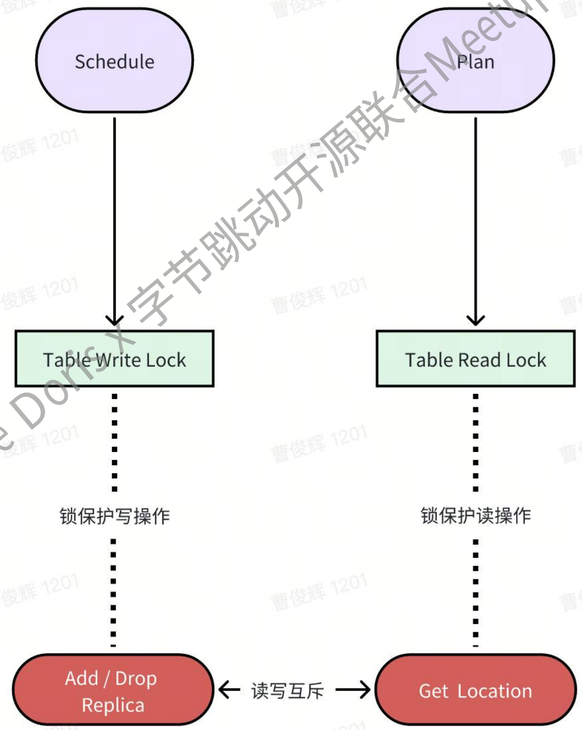
Transaction 锁粒度从 Table 到 Tablet

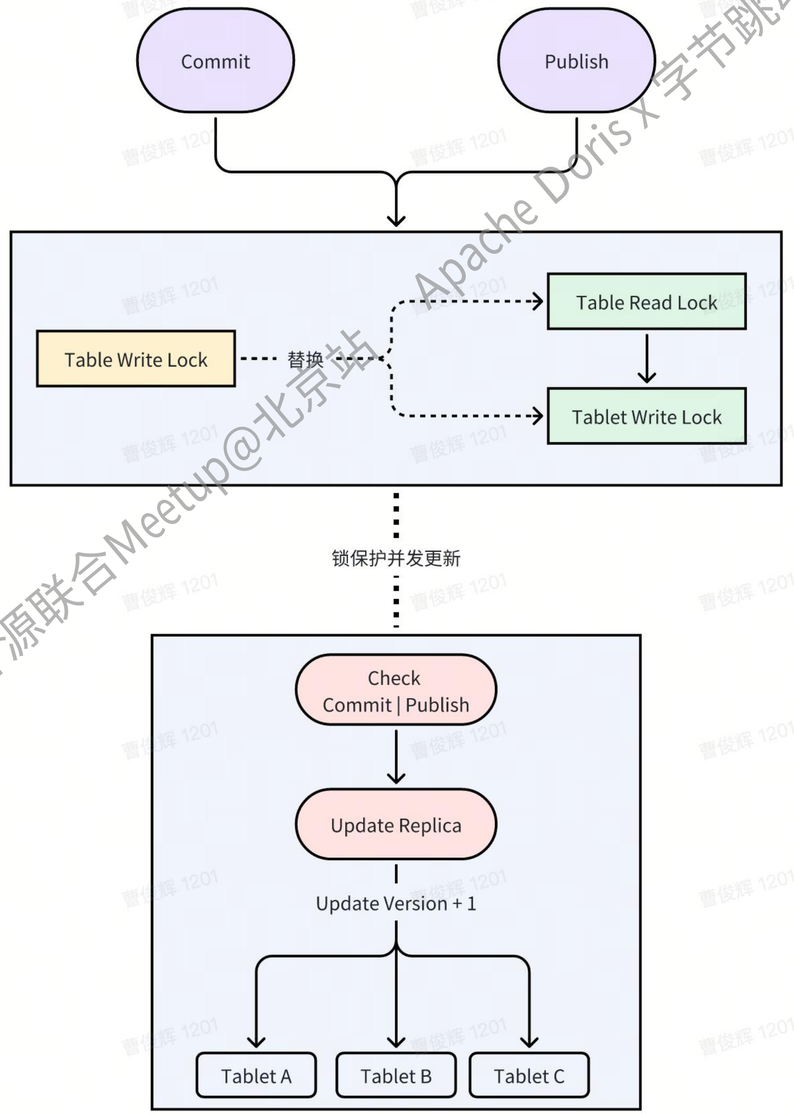
Table Lock

- 保护 Tablet 的添加 / 删除
- 保护表级别元数据

Tablet Lock

- 保护 Tablet Version





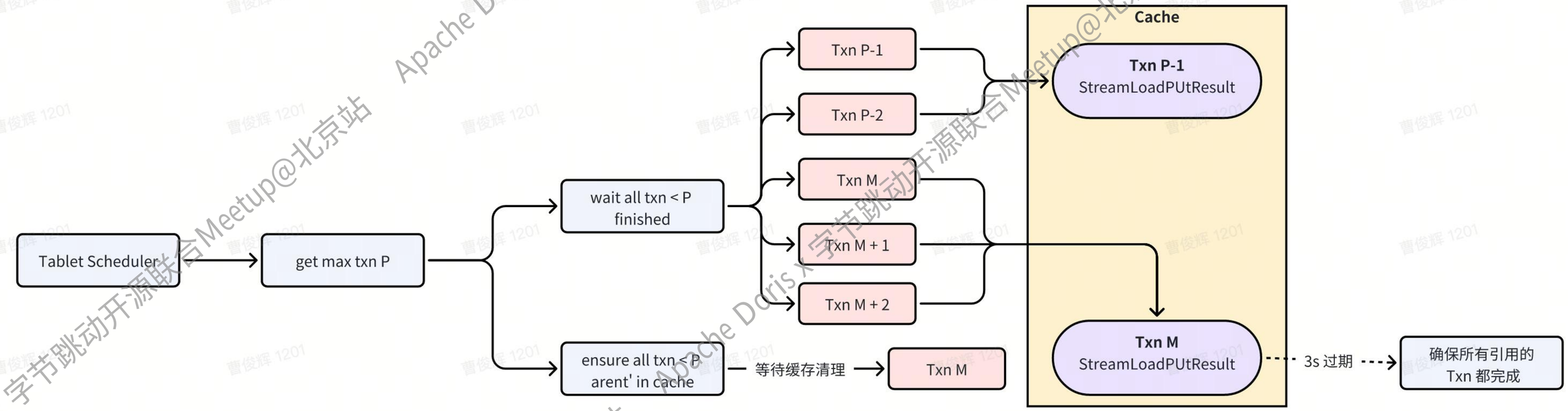
Plan 缓存

缓存策略

Cache Map 保存 Transaction 和对应的 Plan 结果

正确性保证

使用当前全局 Txn ID 作为 watermark，副本删除前会检查 watermark，保证不影响现有导入



五、RPC 性能优化

1. 更换 Thrift 执行模型：TThreadPoolServer => TThreadedSelectorServer

- RPC 并发较高，默认的执行模型会使用较多的线程资源，对资源消耗和稳定性有较大影响
- 新执行模型有单独的网络 IO 线程，对稳定性提升较大

2. 拆分 FE RPC 线程池，BE 侧不再同步等待

- Plan / Commit / Publish 操作提交到等待队列就算成功，完成后通知 BE
- 可以调整读写请求的执行顺序，不再受读写锁影响。

六、其他优化

1. 查询性能

- Light Schema Change 支持添加 BloomFilter 索引

2. 写入性能

- 补齐写入链路未向量化的部分：添加 NodeChannel、Fill Missing Column 等

3. 存储成本

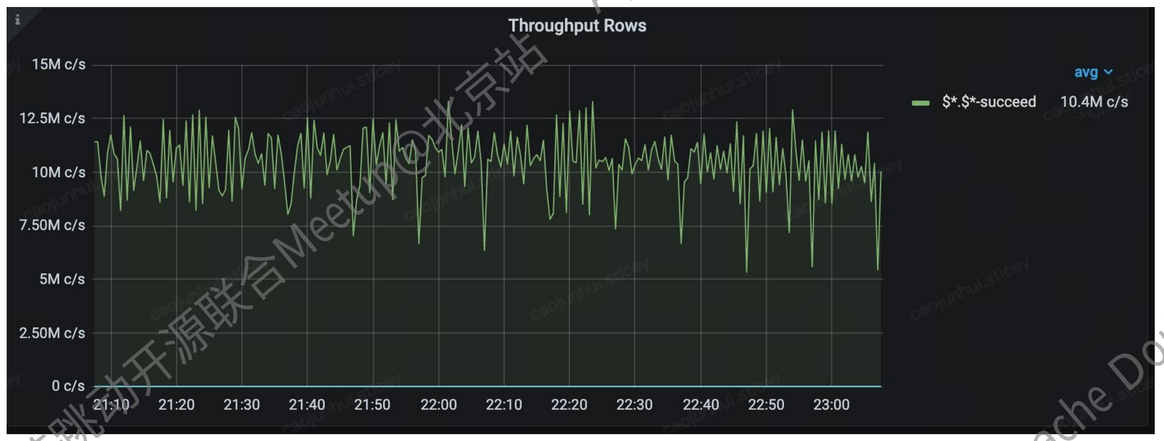
- Doris 内部支持 Bytes 列，相比存储 Base64 编码减少 33% 的成本
- 选盘时基于 Slot 进行 Tablet 分配，缓解磁盘不均
- 使用 Disk Balance

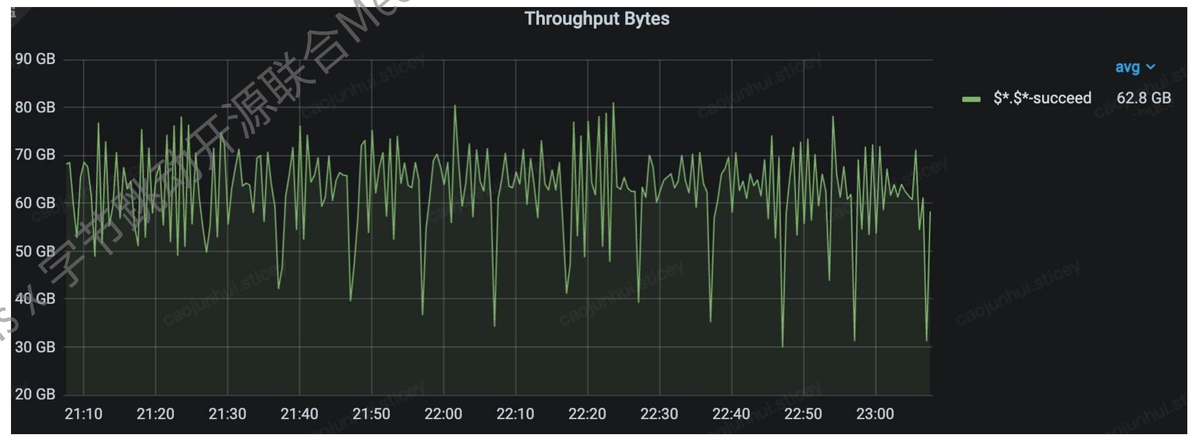
目录

最终结果与结论

Trace 场景能够完全支持，标志着 Doris 几乎能扛住绝大部分场景的导入性能需求

- **上线后**：平均 60G/s，10M row/s
- **集群规格**：10000+ core，7PB 存储
- **当前峰值**：可以用 3000+ core 的HDD 达到 30M row/s，90G/s





抖音集团内部业务现状

实时数仓

在 Doris 内部做 ETL，
同时可以直接进行高效查询

Serving

为大量的广告主、电商/本地生活服务商家、直播达人提供高 QPS 和低延迟的数据分析服务

数据湖

DorisOnEs 、DorisOnHive 等

业务规模：30w+ core

目录

未来工作

资源弹性
冷热分离

存算分离

引入社区的存算分离版本
增加资源弹性
降低冷数据的存储成本

开发提效

实时数仓

优化实时数仓场景
的开发模式

资源隔离

稳定性

通过资源隔离
提高数据服务稳定性

Thanks !

字节跳动开源

微信扫描二维码，关注我的公众号