Tencent 腾讯 | CSIG 云与智慧产业事业群

ClickHouse 在日志检索分析方向架构融合与内核演进实践

郑天祺 （Amos Bird）

自我介绍

中科院计算所博士，腾讯云大数据技术研发

https://github.com/amosbird

活跃 8 年的 ClickHouse 贡献者，600+ PR

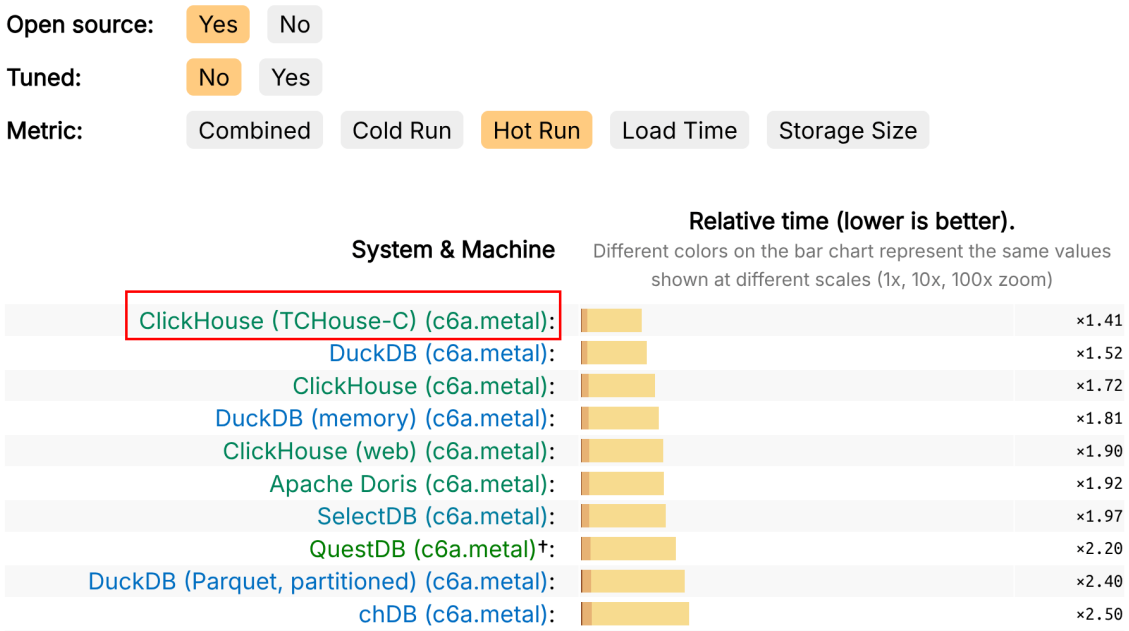
- Projections，String Hash Map 等功能

腾讯云大数据统一 ClickHouse 内核： Titan

- 基于 ClickHouse 打造实时分析检索引擎底座

近期社区贡献：助力 ClickHouse 性能超越 DuckDB

https://github.com/ClickHouse/ClickBench/pull/412



目录

日志场景介绍 & ES 与 CK 痛点

日志场景规模与特点：

- 日志场景数据量大、价值密度低，用户追求**极致降本**
- 强时序性，**写多读少**，无相关性检索需求，无更新需求
- 既需要**快速检索**（定位问题、过滤条件查询），也需要**高效分析**（聚合统计、趋势洞察）
- ES 因**检索功能丰富与生态易用性**成为了日志生态的入门首选系统

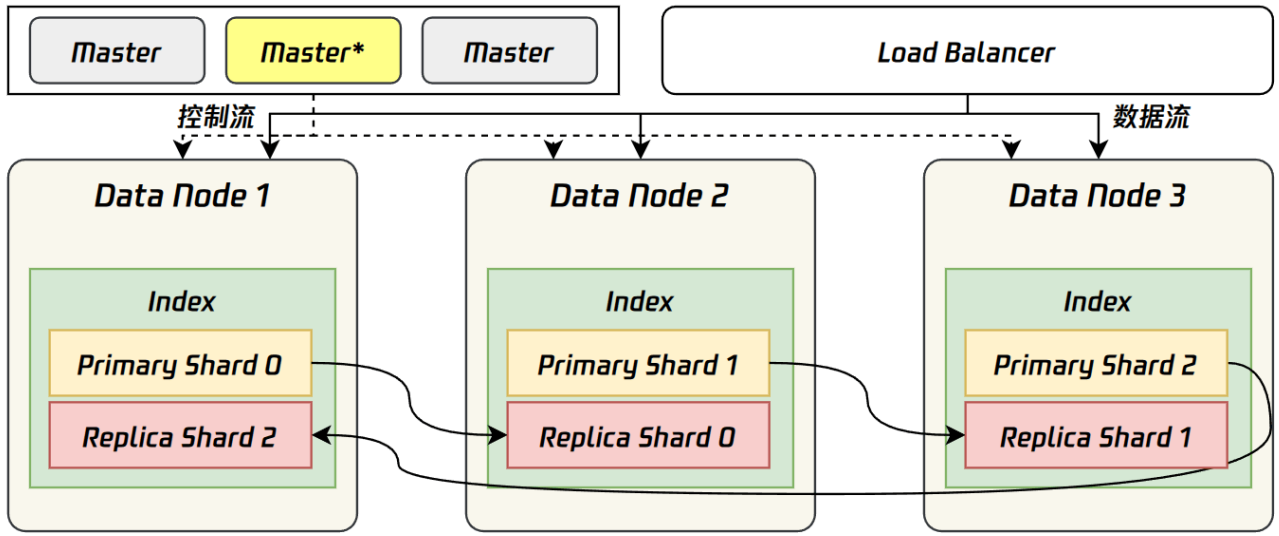
ES 与 CK 痛点对比：

产品	优势	劣势
Elasticsearch（简称 ES）	1、**日志生态完善（ELK）** 2、检索功能和易用性强 3、分布式管理和扩展性强	1、写入性能弱 2、内存压力大 3、存储成本高
ClickHouse（中文社区简称 CK）	1、**单机性能强悍** 2、写入性能高 3、存储成本低	1、分布式能力弱 2、检索性能差，索引能力弱 3、缺乏统一日志生态

架构对比

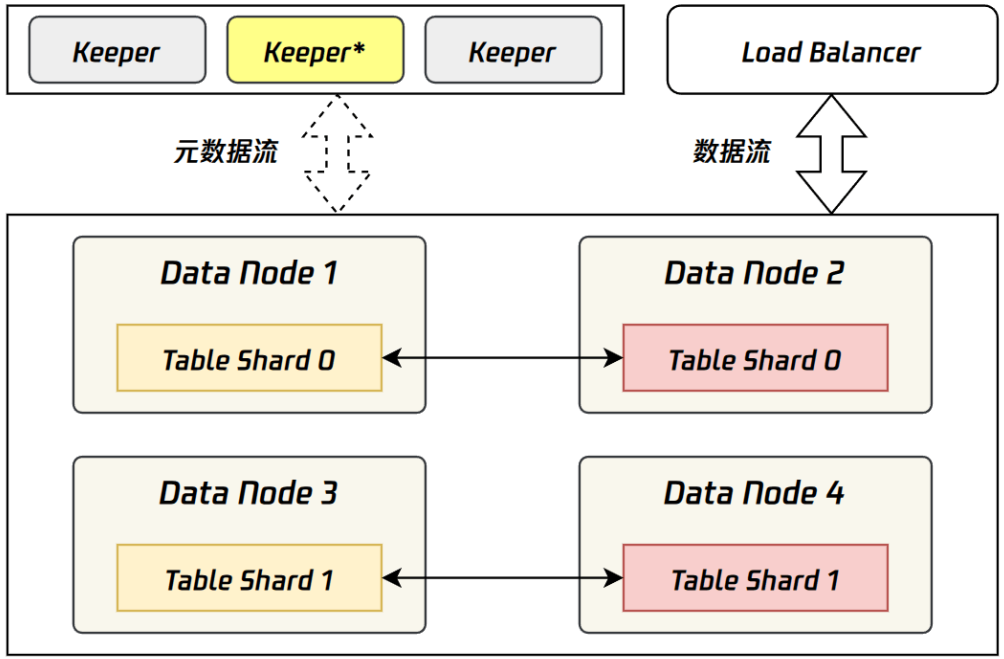
Elasticsearch:

- 主从架构，P2P 接入模式
- 表级 Sharding，可自均衡
- First-Class 分布式设计，使用体验与单机无区别



ClickHouse:

- 多主 P2P 架构，P2P 接入模式
- 节点级别 Sharding，手动均衡
- Second-Class 分布式设计，手动组建集群…



Elasticsearch 在分布式架构中的易用性优势明显

性能对比

Elasticsearch:

- 关闭行存
- 关闭字段索引
- patch 代码关闭了所有内置元数据列（_id 等）
- refresh_interval 30 秒刷盘，translog 5 秒异步刷盘
- 单机 16 个 shard 并行写入

ClickHouse:

```
CREATE TABLE default.http_logs (
    `@timestamp` DateTime,
    `clientip` LowCardinality(String),
    `request` LowCardinality(String),
    `status` Int32,
    `size` Int32)
ENGINE = MergeTree ORDER BY tuple() SETTINGS index_granularity = 8192
```

Esrally http_logs 测试集：http://benchmarks.elasticsearch.org.s3.amazonaws.com/corpora/http_logs

规模：2.5 亿条，大小：31.1 GB，单条日志平均：约 150 bytes，写入程序 24 并发，HTTP POST 单请求 10mb

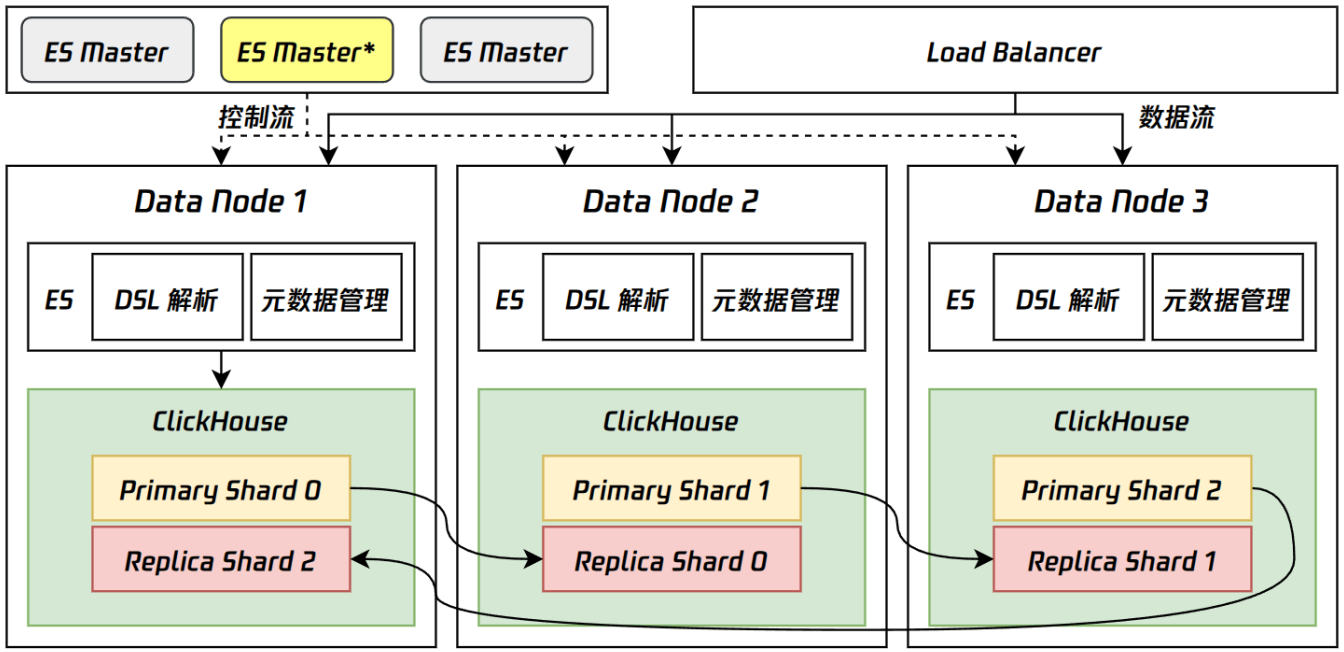
	ES 原生	ES 精简	ClickHouse	ClickHouse 字典
写入耗时	251s	220s	51.2s	38.9s
存储大小	20.2GB	2.3GB	3.9GB	1.8GB

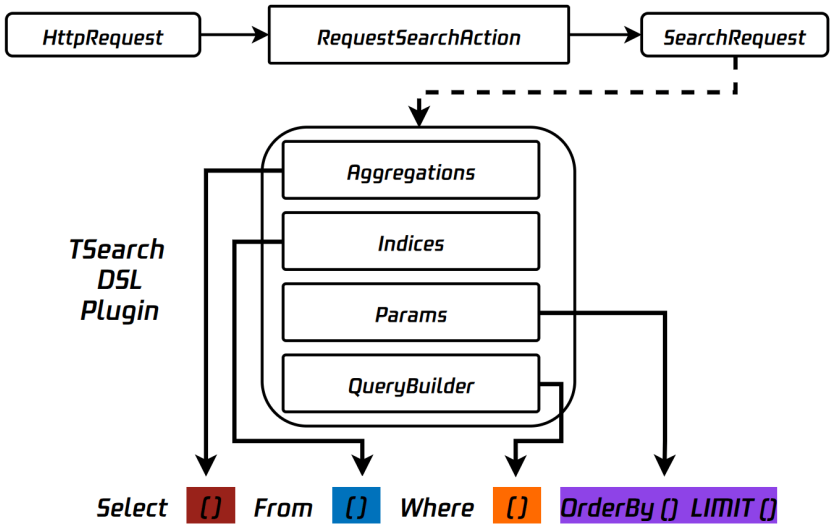
ClickHouse 在数据写入和存储成本方面具有明显优势

ES & CK 融合与优化

融合架构 1.0

- 融合创新：ES 分布式管理能力 + ClickHouse 单机存算能力
- 生态兼容：兼容 ELK 生态，ES DSL 查询，几乎无业务改造成本

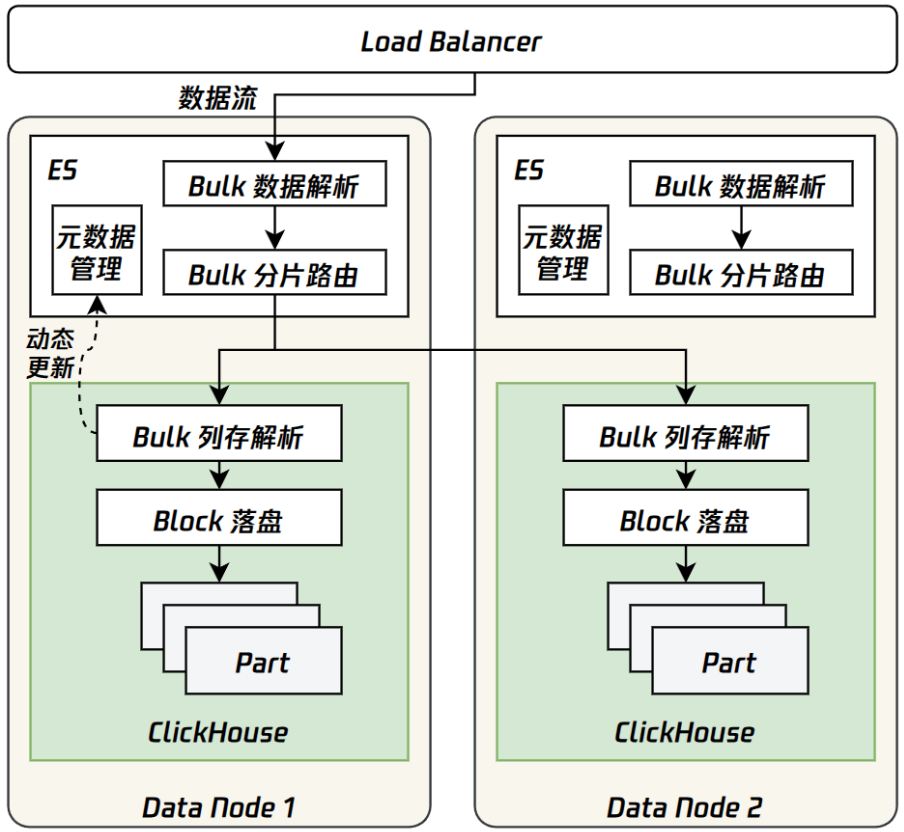




融合架构 1.0 的损耗

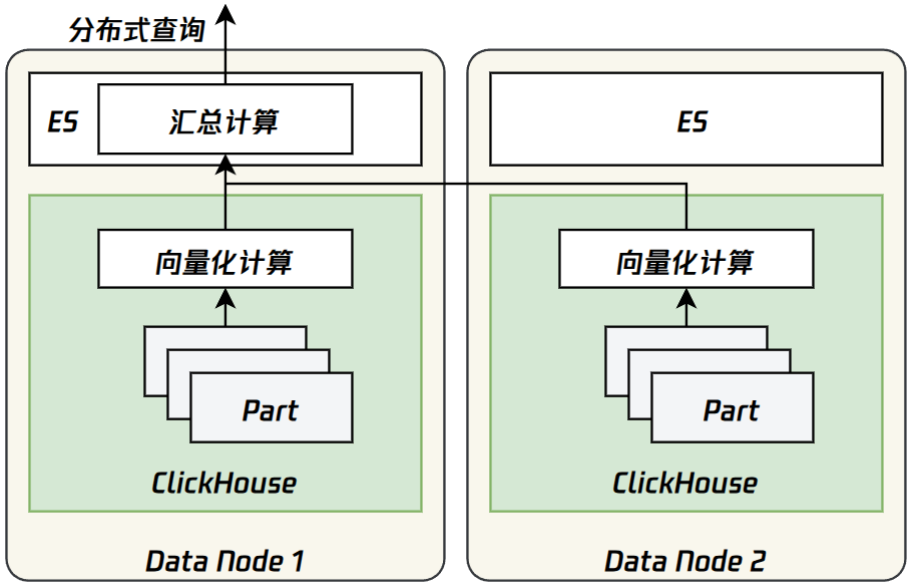
写入损耗

1. 写入请求重复解析：ES、主分片引擎、从分片引擎各解析一次
2. ES 基于 JVM 解析写入请求性能受限
3. 数据流经 ES JVM 转发带来额外开销
4. 流量不均，碎片写入问题等等



查询损耗

1. 分布式查询依赖 ES JVM 汇总计算，性能受限
2. 自研引擎 SQL 生态出口受限，功能受损

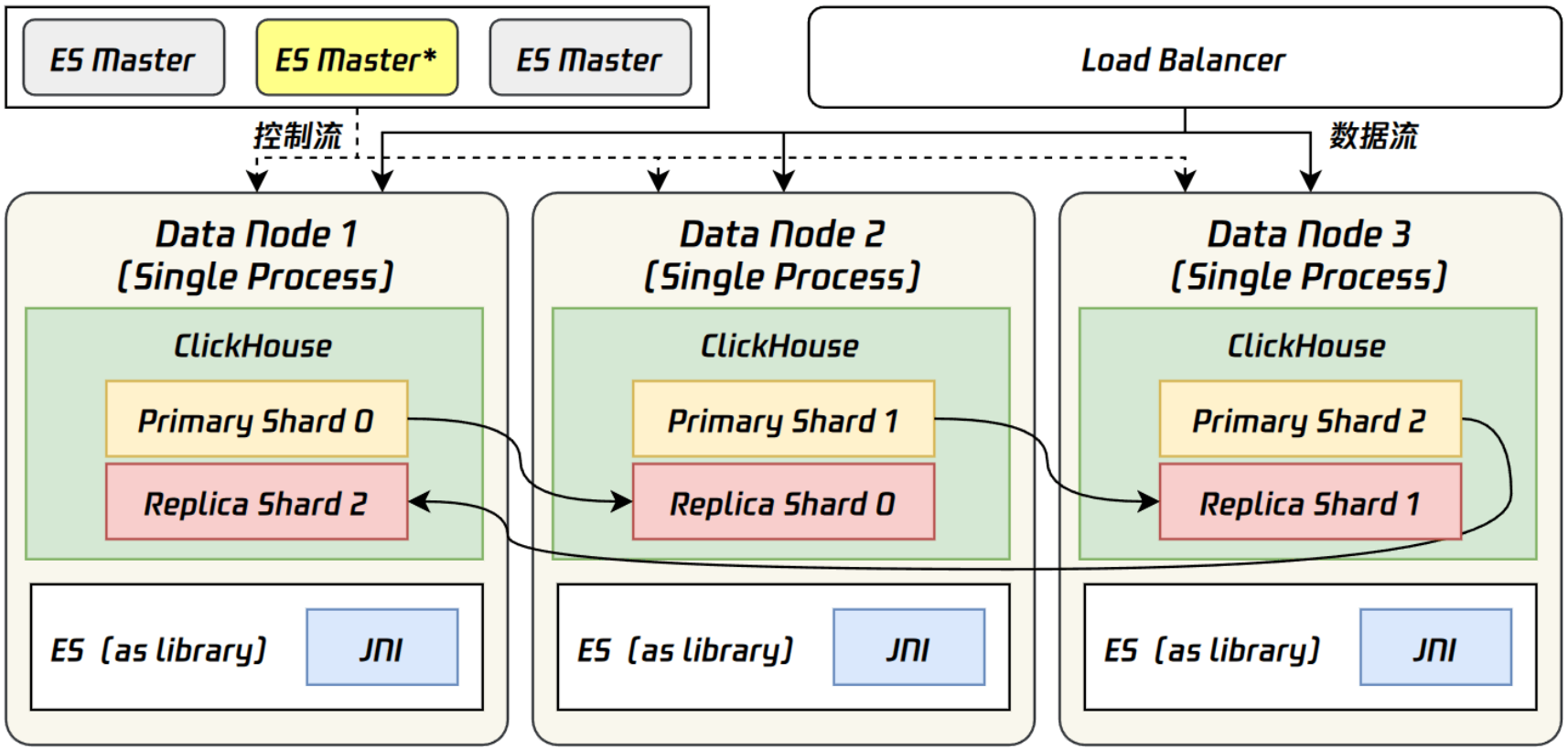


根因：ClickHouse 未被充分利用

融合架构 2.0

新增优势

- ClickHouse 驱动：ClickHouse 闭环数据面处理，与 ES 无损融合
- 双生态融合：兼容 ELK 生态与 SQL 生态，支持多场景日志检索分析



ES 元数据建模 & Sharding

ClickHouse 回放 ES 元数据

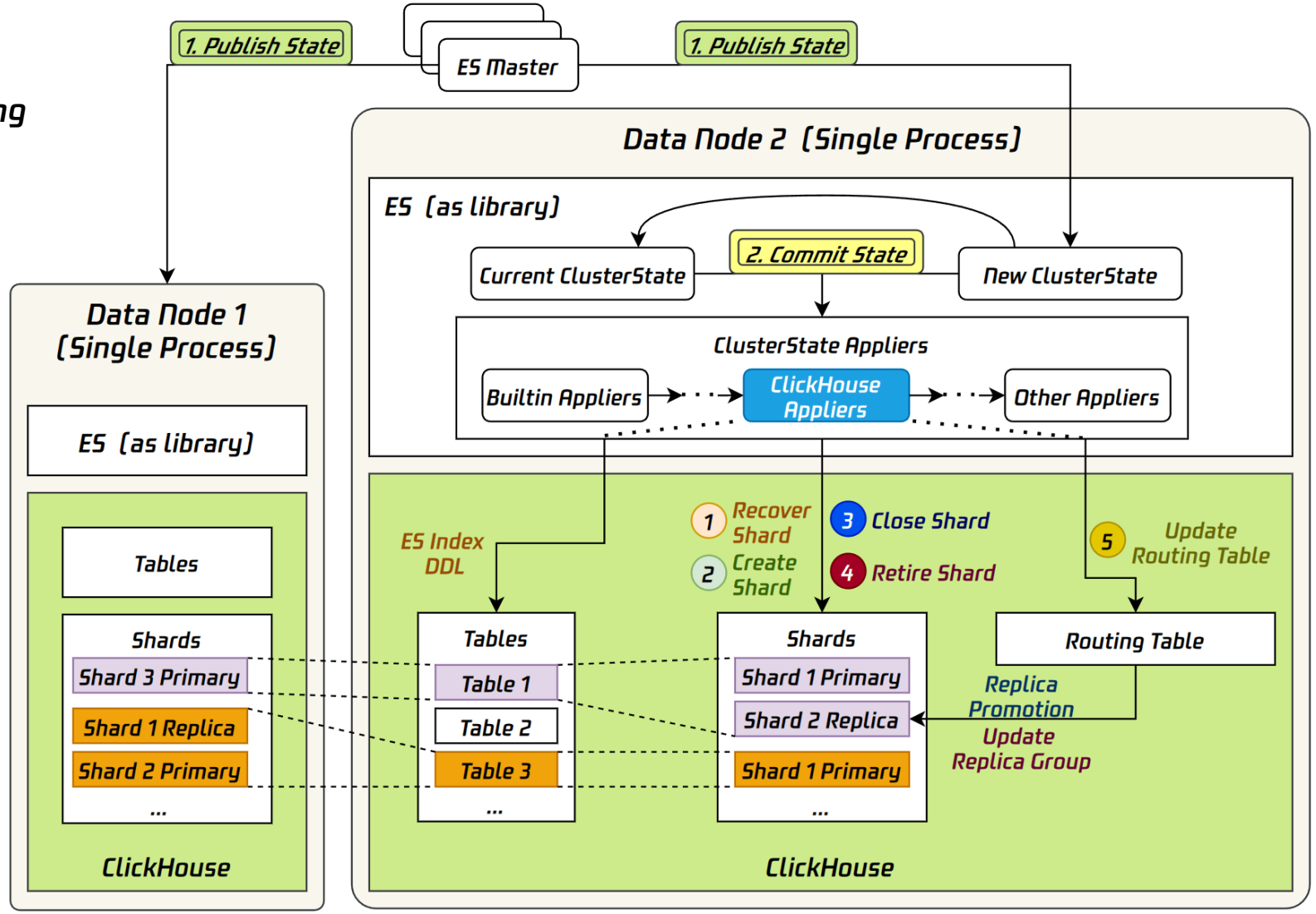
- 引入 ES Index Table，Schema <-> Mapping
- 引入 Table Shard 概念，对应 ES Shard
- 接收 ES ClusterState，生成集群拓扑

副本同步与漂移

- 对齐 ES 分片恢复状态机
- 构建本地 Manifest 存储管理 Part 元数据
- 数据目录与 ES Shard 目录一一映射

收益

- ClickHouse 分布式能力提升（no more ZK）
- ES DSL & CK SQL 双生态写入查询



ES 元数据建模 & Sharding

ClickHouse 回放 ES 元数据

- 引入 ES Index Table，Schema <-> Mapping
- 引入 Table Shard 概念，对应 ES Shard
- 接收 ES ClusterState，生成集群拓扑

```
data/nodes/0/indices/0hUyi0nCRHKgCl7MsKSDcg/68
├── index
│   ├── segments_2
│   └── write.lock
├── _state
│   ├── retention-leases-7.st
│   └── state-0.st
└── translog
    ├── translog-2.tlog
    └── translog.ckp
```

ES Shard 目录

（保存 ES Shard allocation 状态）

副本同步与漂移

- 对齐 ES 分片恢复状态机
- 构建本地 Manifest 存储管理 Part 元数据与 WAL
- 数据目录与 ES Shard 目录一一映射

```
data/rocksdb/data/data/0hUyi0nCRHKgCl7MsKSDcg_68/
├── 000022.sst
├── 000056.log
├── 000058.sst
├── CURRENT
├── IDENTITY
├── LOCK
├── LOG
├── MANIFEST-000004
└── OPTIONS-000011
```

ClickHouse Shard Manifest/WAL 目录

收益

- ClickHouse 分布式能力提升（no more ZK）
- ES DSL & CK SQL 双生态写入查询

```
data/data/data/0hUyi0nCRHKgCl7MsKSDcg_68/
├── 1b21e681-6eb0-4061-a845-ad3d054d09e9
├── 1f75931e-2bf9-4dcb-98fc-bfaaadbfcc82
├── 2a262b85-cc21-4166-988b-28d176ef3881
├── detached
└── format_version.txt
```

ClickHouse Shard Data 目录

ES Bulk 写入优化

ES Bulk 协议痛点问题

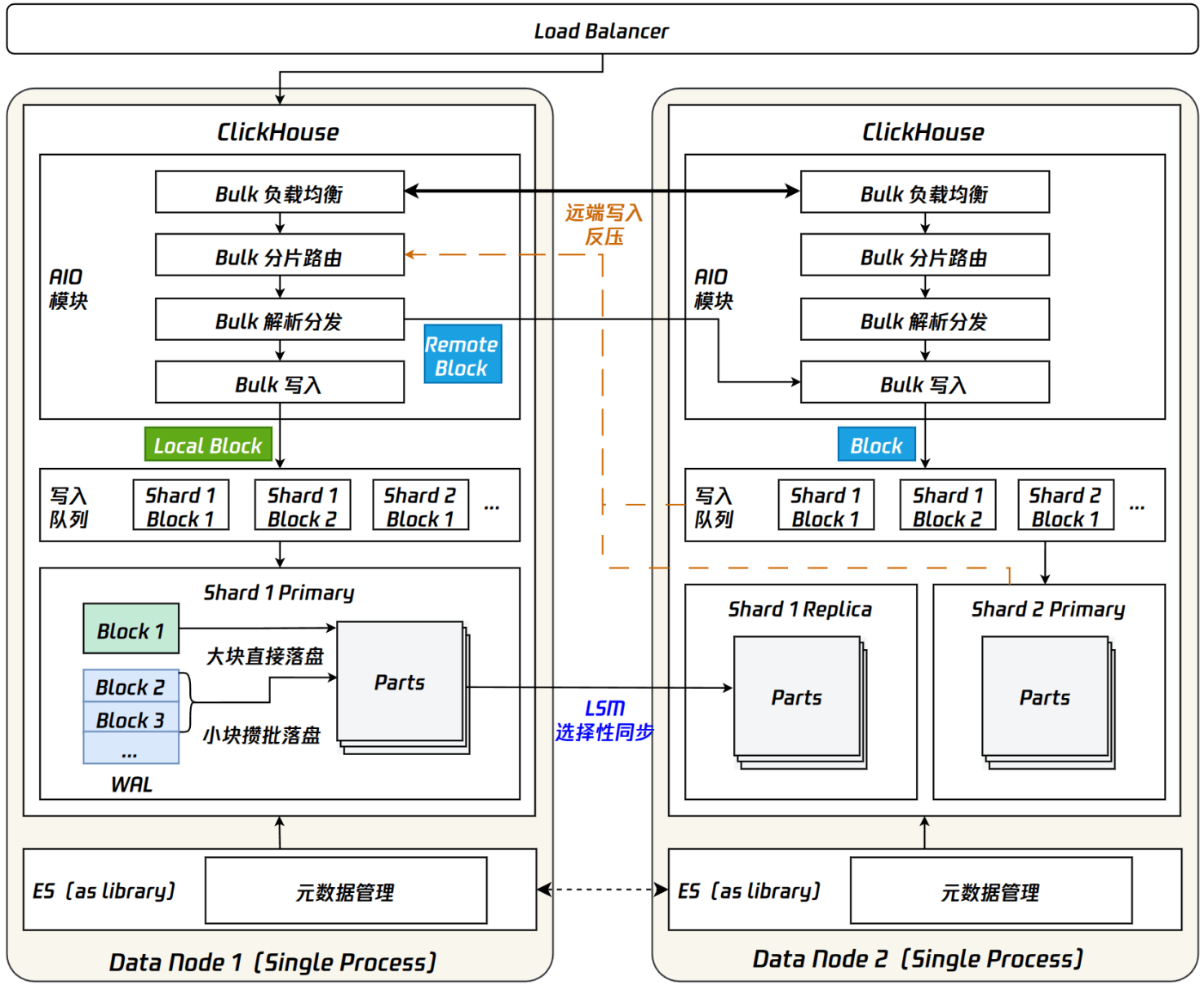
- 分布式路由开销高，负载偏斜风险大
- 单 Bulk 多表写入，拆分为多个小批量请求，易碎片化

Bulk 流量优化

- 节点与分片双维度均衡
- 系统负载感知与多级反压流控

Bulk 存储优化

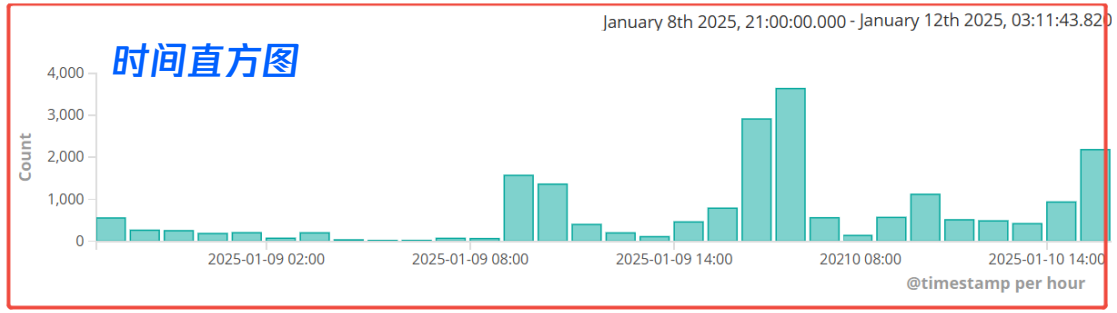
- LSM Merge 选择性同步降低写放大
- 小块数据 WAL 合并生成 Part，大块数据短路写



日志多写少查？别被表象迷惑：查询才是用户感受核心

日志场景典型看板：Kibana Discover 页面

- 时间直方图：按时间区间展示命中条数，用于时间圈选
- 日志 TopN 检索：展示查询命中的原始日志



SQL 需要两条语句来渲染

时间直方图聚合

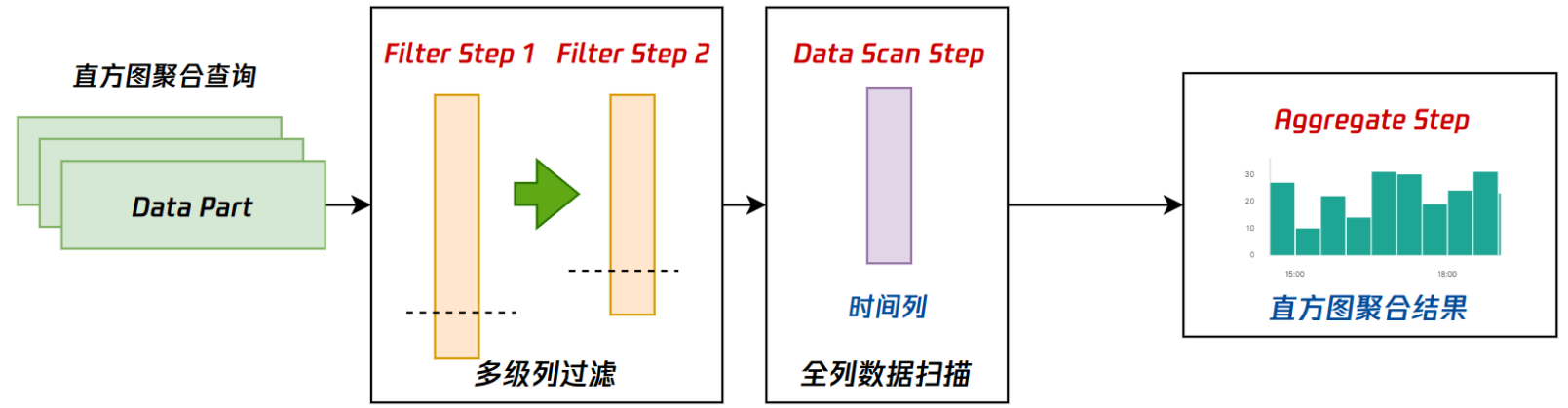
```
SELECT
  toStartOfHour(timestamp),
  COUNT(*)
FROM logs
WHERE cluster_name = 'es-xxx'
  AND timestamp >= '2025-01-08T21:00:00.000Z'
  AND timestamp <= '2025-01-12T03:11:43.820Z'
GROUP BY hour
ORDER BY hour;
```

TopN 检索

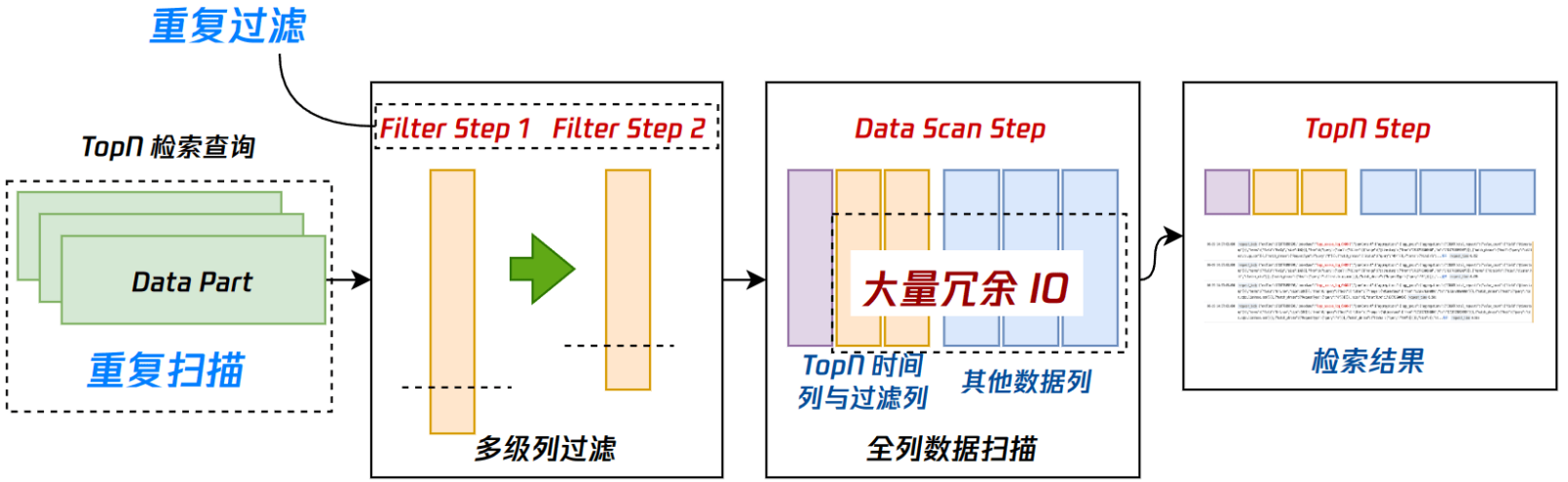
```
SELECT *
FROM logs
WHERE cluster_name = 'es-xxx'
  AND timestamp >= '2025-01-08T21:00:00.000Z'
  AND timestamp <= '2025-01-12T03:11:43.820Z'
ORDER BY timestamp DESC
LIMIT 100;
```

重复扫描，重复过滤

ClickHouse 检索涉及大量冗余 IO




```
SELECT
  toStartOfHour(timestamp),
  COUNT(*)
FROM logs
WHERE cluster_name = 'es-xxx'
  AND timestamp >= '2025-01-08T21:00:00.000Z'
  AND timestamp <= '2025-01-12T03:11:43.820Z'
GROUP BY hour
ORDER BY hour;
```



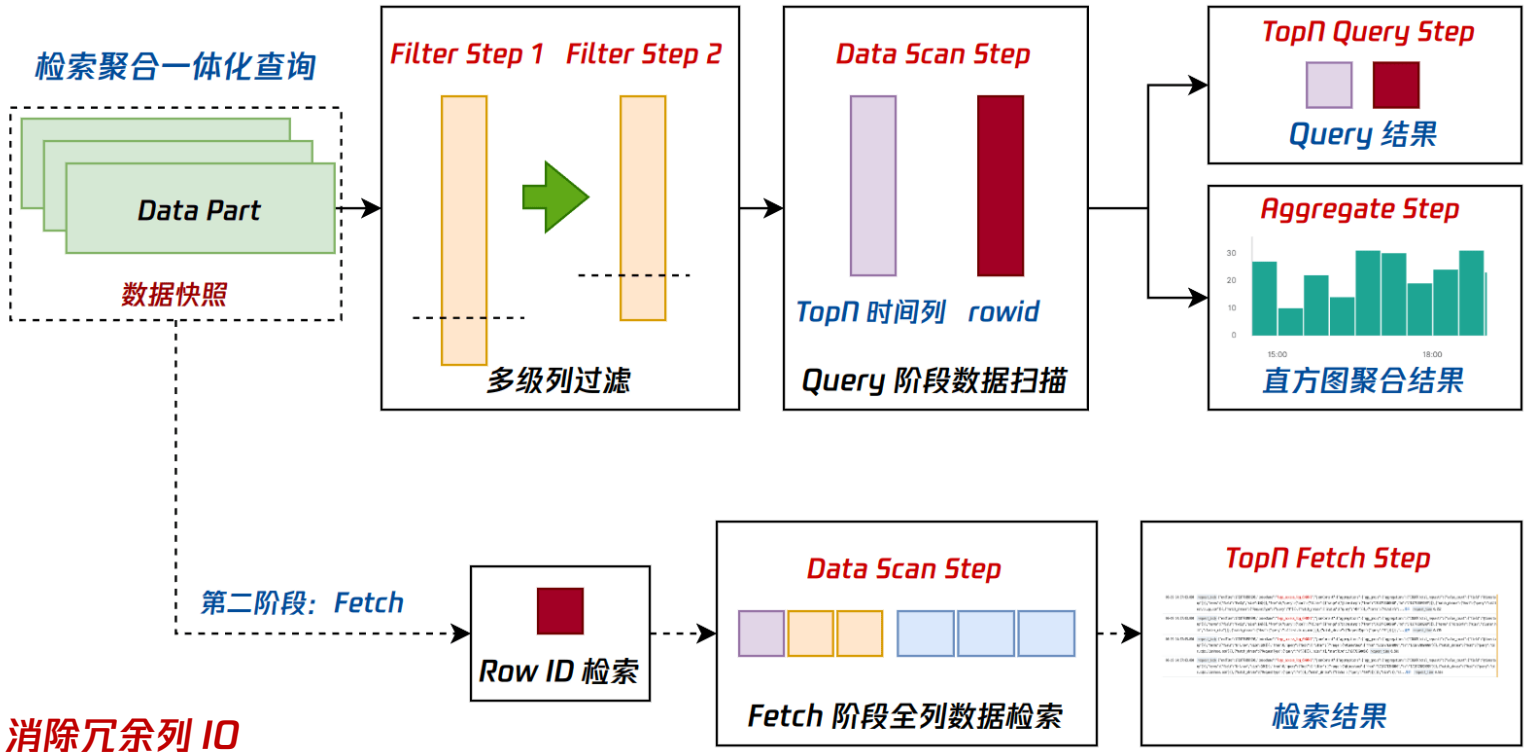

```
SELECT *
FROM logs
WHERE cluster_name = 'es-xxx'
  AND timestamp >= '2025-01-08T21:00:00.000Z'
  AND timestamp <= '2025-01-12T03:11:43.820Z'
ORDER BY timestamp DESC
LIMIT 100;
```

根因：列存向量化引擎依赖列对齐进行 TopN 运算，缺少延迟物化导致大量的冗余列 IO

解法：SQL 检索聚合一体化

扩展 SQL 执行模式实现检索聚合一体化

```
SELECT
    topN(timestamp),
    timeBucketByHour(timestamp)
FROM logs
WHERE cluster_name = 'es-xxx'
    AND timestamp >= '2025-01-08T21:00:00.000Z'
    AND timestamp <= '2025-01-12T03:11:43.820Z'
```



TopN 两阶段检索数据

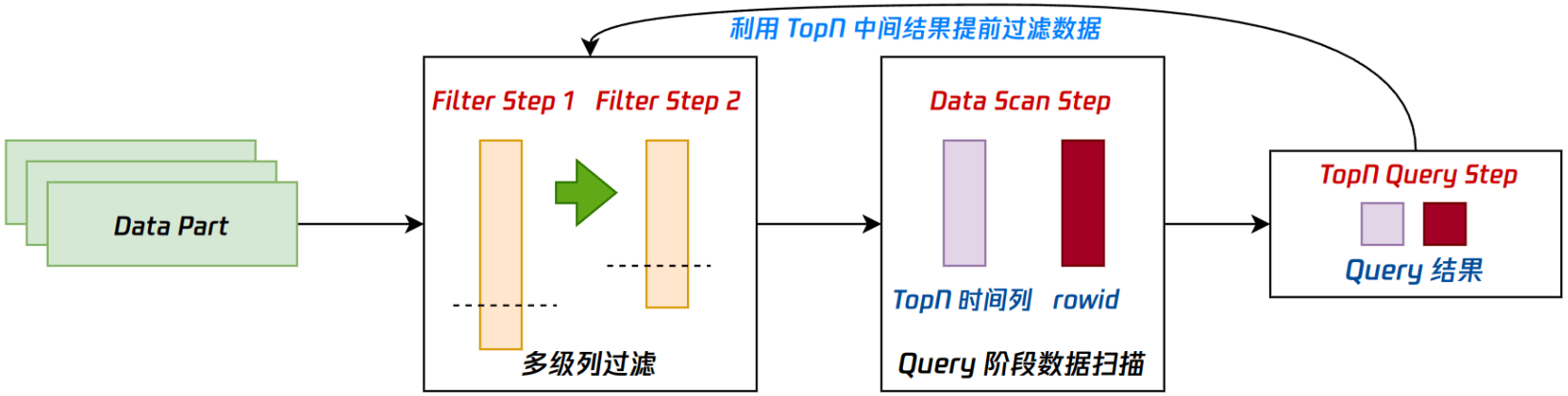
- Query 阶段：返回 TopN row_id
- Fetch 阶段：根据 row_id 快照中读取 TopN 数据，消除冗余列 IO

Row ID Fetch 能力已贡献社区：

https://github.com/ClickHouse/ClickHouse/pull/58224 Primary key analysis for _part_offset

https://github.com/ClickHouse/ClickHouse/pull/79417 Primary key analysis for _part_starting_offset + _part_offset

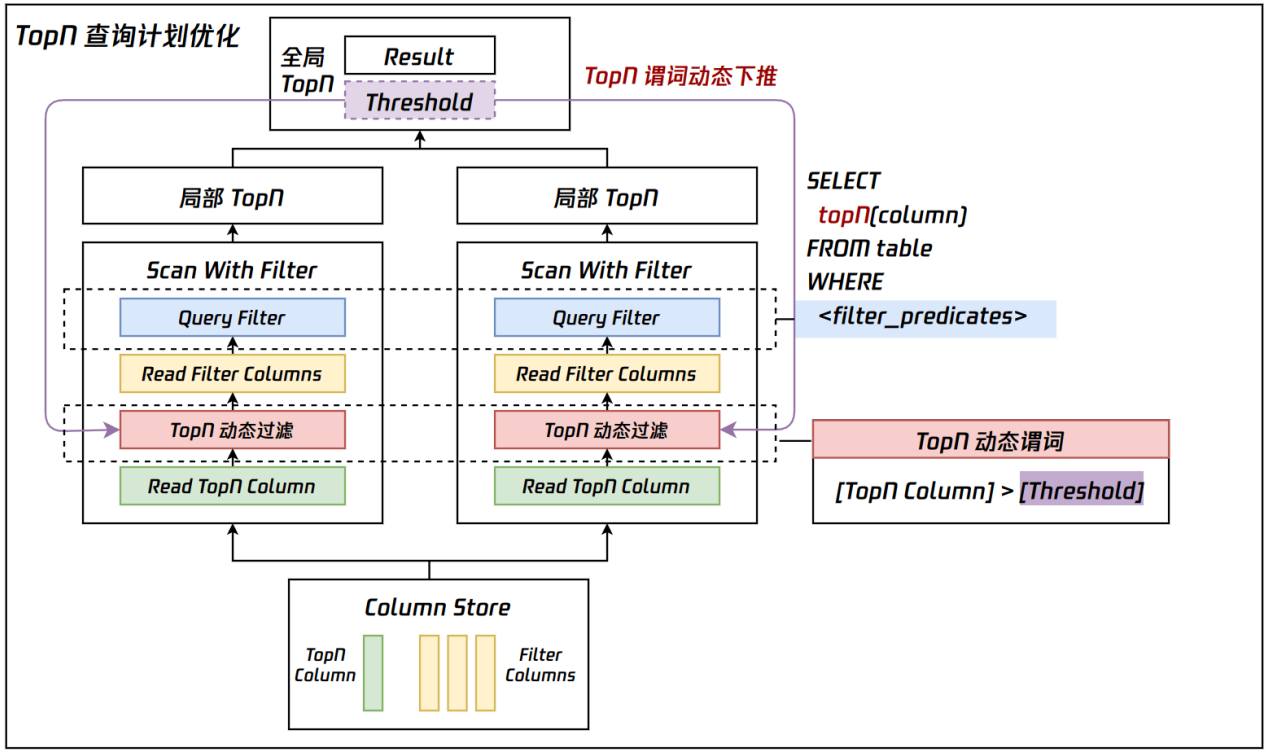
进一步挖掘 TopN 检索优化空间



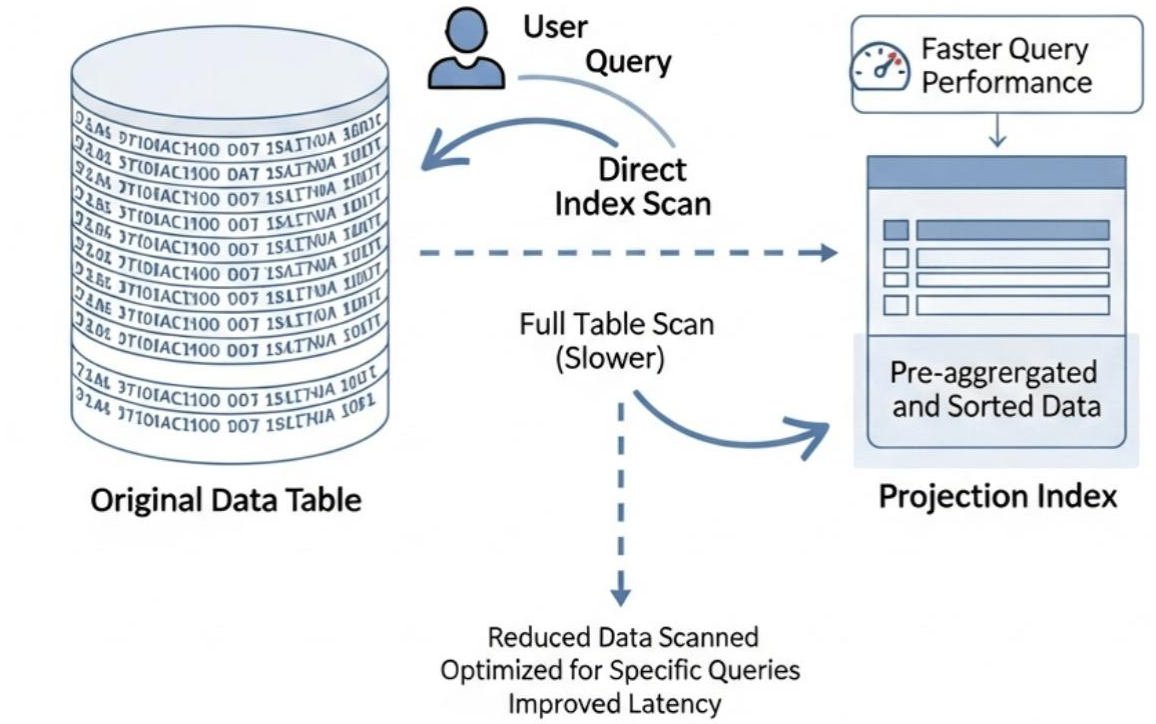
TopN 动态阈值下推过滤

- 无锁更新全局 TopN 阈值，实现高并发下的无损下推过滤
- 基于多级列过滤框架前置 TopN 动态过滤条件，快速淘汰不满足阈值的数据

ClickBench Q23	优化前	优化后	性能提升
单线程	43.234s	4.418s	9.8x
16 线程	2.345s	0.412s	5.7x



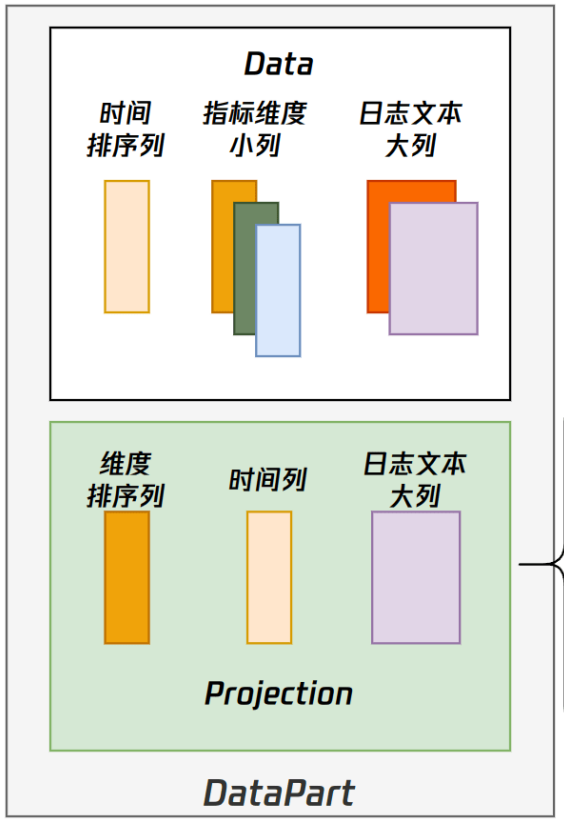
Projection Index



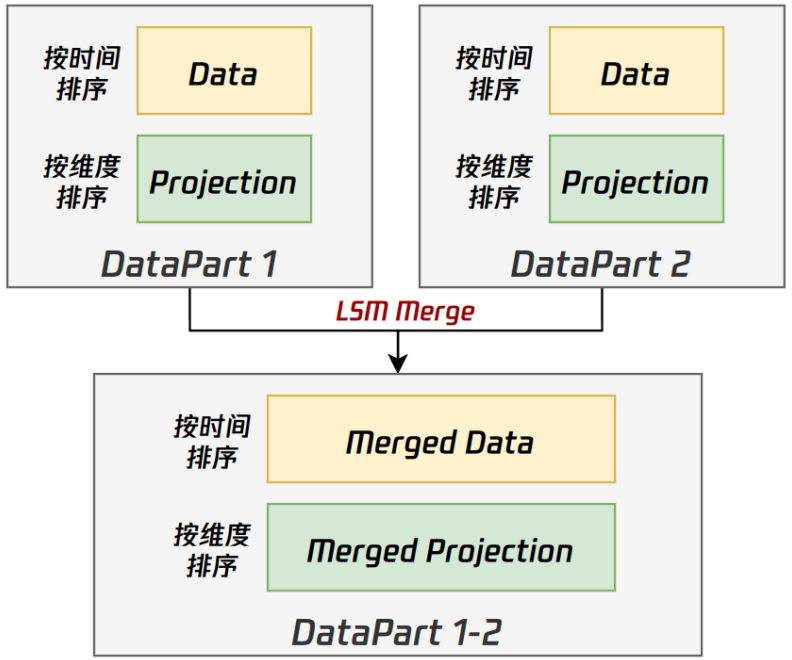
Projection 介绍

- 一种特殊的、物理存储的数据结构，由原始表的部分数据经过预计算（聚合、排序）生成
- 不是视图: Projection 存储实际数据，而非逻辑定义
- 不是物化视图：Projection 与原始表数据一同存储和管理（位于相同 Part 内）
- Projection 通过数据冗余支持多主键加速，但需复制查询涉及的所有列，存储开销大

timestamp 主键 + small_col1 副主键，额外冗余存储 big_col1



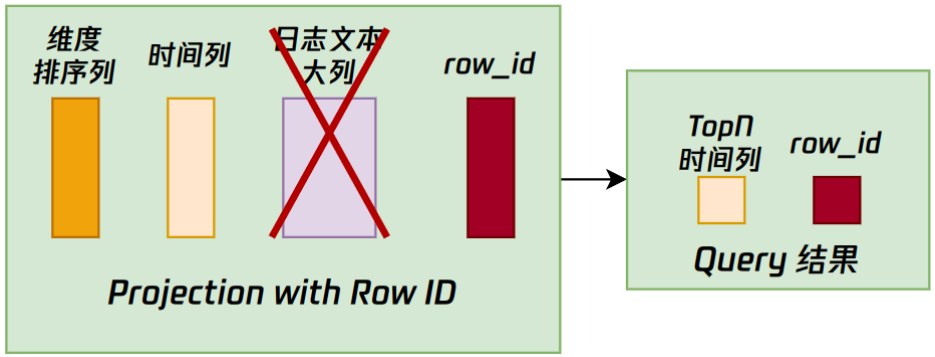

```
CREATE TABLE data
(
    timestamp DateTime,
    small_col1 int,
    small_col2 int,
    small_col3 int,
    big_col1 String,
    big_col2 String,
    PROJECTION p_small_col1
    (
        SELECT
            small_col1,
            timestamp,
            big_col1
        ORDER BY
            small_col1
    )
)
ORDER BY timestamp
```



Projection Index 原理：存储并维护 row_id

- 扩展 Projection 支持 _part_offset（row_id）存储（https://github.com/ClickHouse/ClickHouse/pull/78429）
- 基于 row_id 进行 fetch（again）（https://github.com/ClickHouse/ClickHouse/pull/58224）

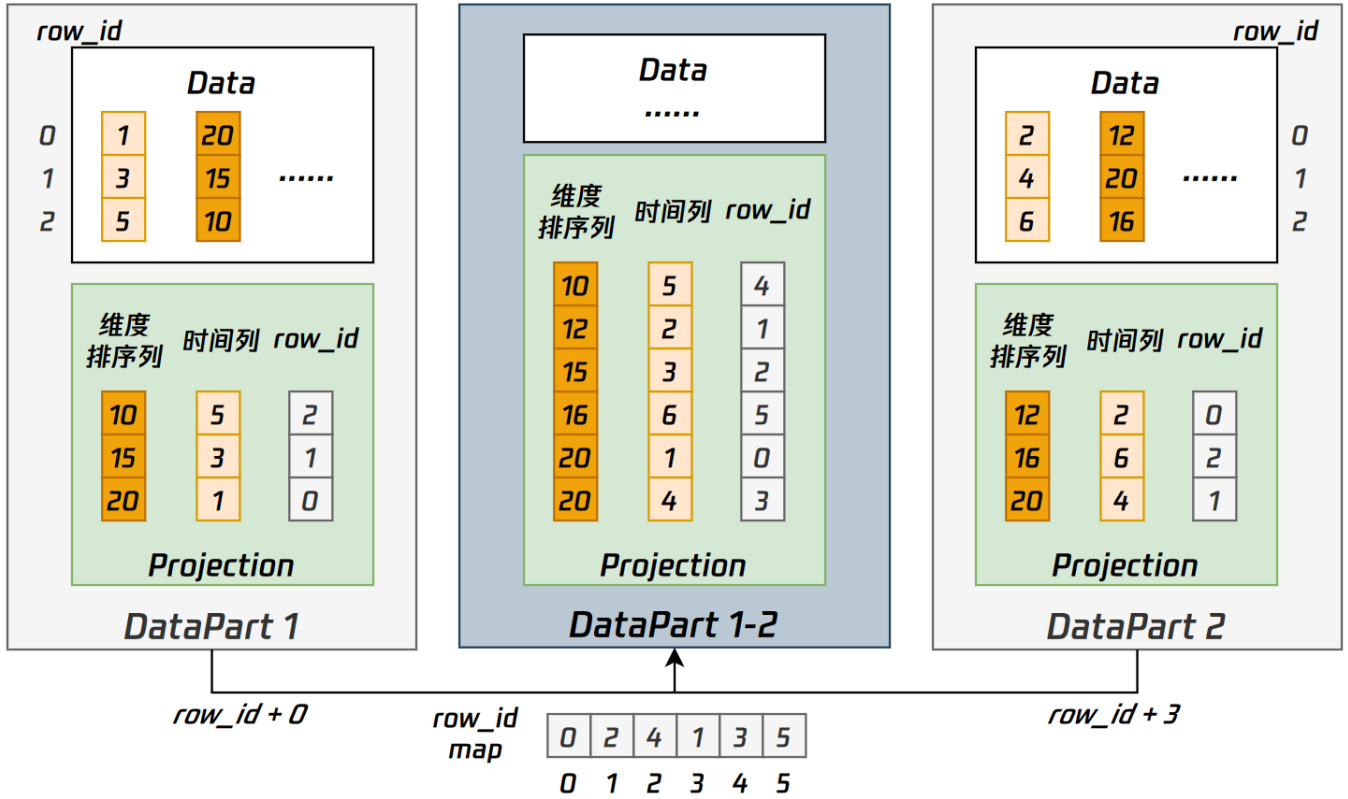
Projection + row_id：去除冗余大列



Projection Index 样例查询

```
SELECT * FROM data
WHERE (_part_starting_offset + _part_offset) IN (
    SELECT _part_starting_offset + _part_offset
    FROM data
    WHERE small_col1 = 42)
```

- 采用 FOR 编码结合 BitPacking 算法对 RowID Mapping 压缩
- 3 亿行数据的 RowID Mapping 仅需 500MB 内存，生产环境通常低于 100MB

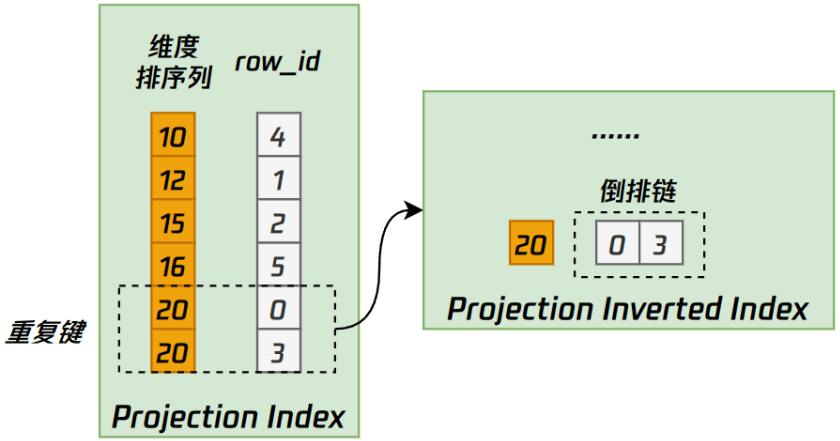


Projection Index 扩展倒排 & 全文索引

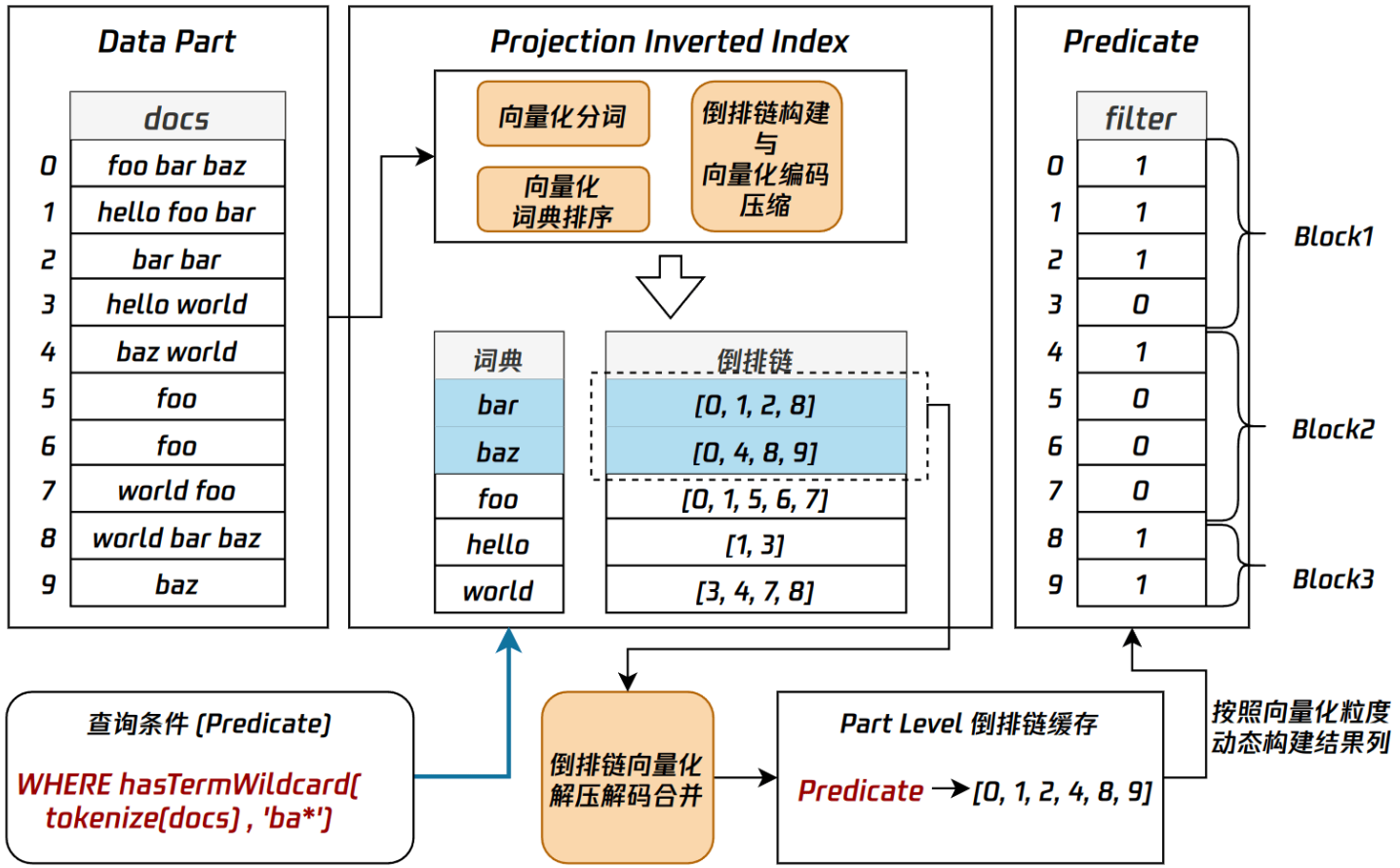
聚合 _part_offset 形成倒排表（Posting List）

From a thousand feet:

```
PROJECTION p_idx (
    SELECT groupArray(_part_offset)
    GROUP BY  small_col1
)
```



分词 + 聚合 = 全文索引

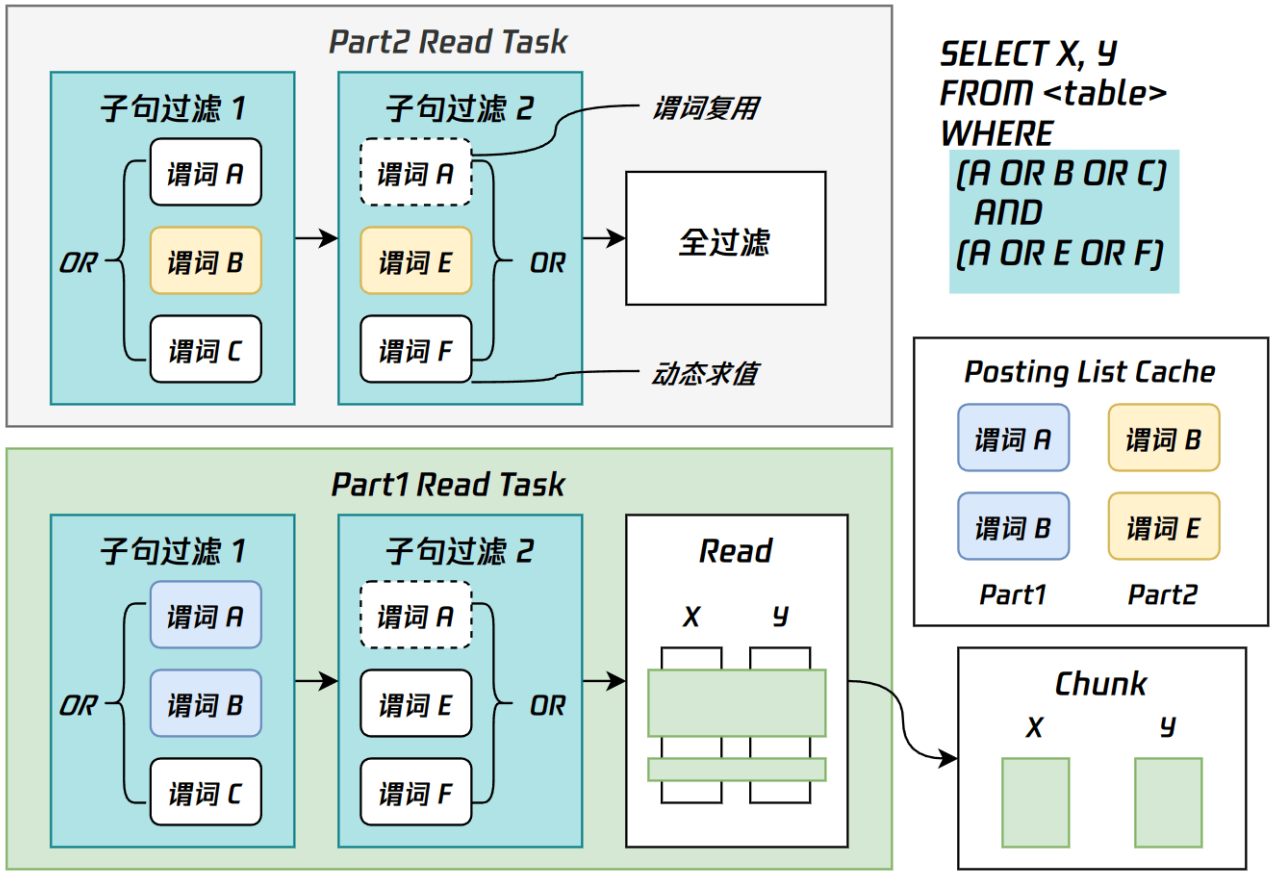


倒排链的具体实现借鉴了 Lucene101PostingsFormat

Projection Index 查询框架

扩展 MergeTree 读链路，动态应用 Skip Index & Projection Index

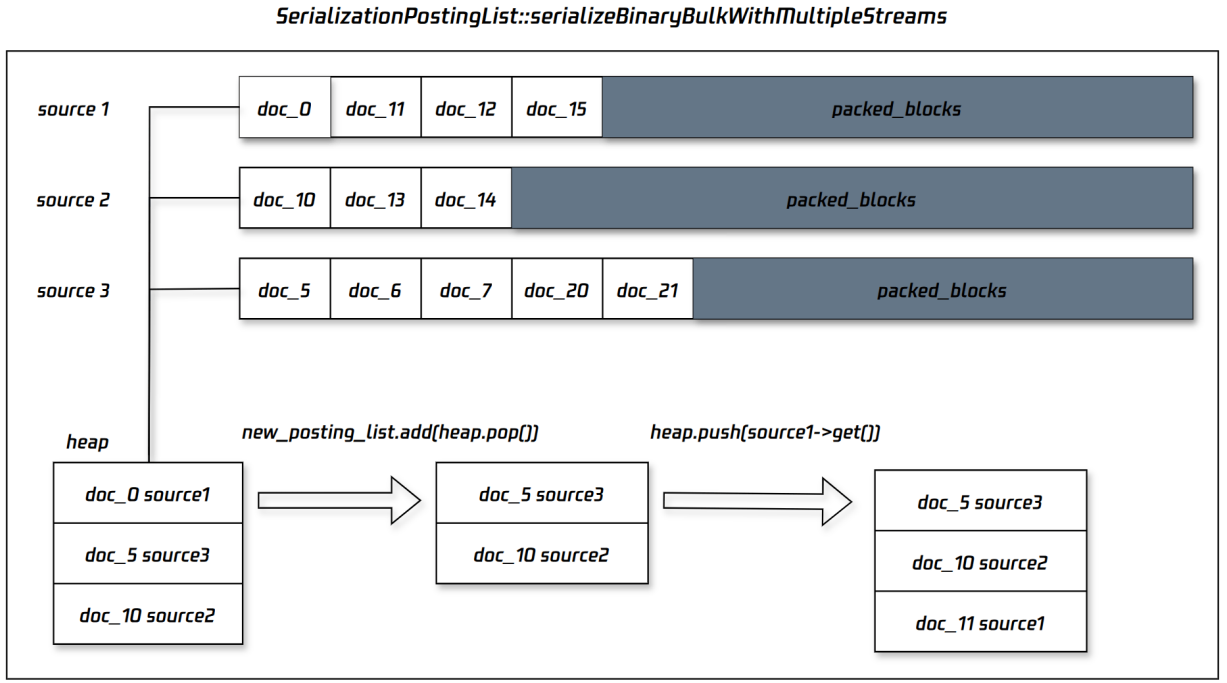
https://github.com/ClickHouse/ClickHouse/pull/81526
https://github.com/ClickHouse/ClickHouse/pull/81021 Under Review



替代基于 Skip Index 的全文索引

- 核心优势：Projection Index 支持流式合并与流式重建索引，写入性能大幅提升

1. PostingList 设计为 AggregateFunction，利用 Aggregate Projection 进行聚合合并
2. 构造 ColumnReadBuffer 将反序列化与合并过程推迟到序列化当中流式完成
3. 重写并精简 Lucene 倒排链结构，去除词频以及其他打分所需信息



- Projection Index 是表，方便观测与扩展，能复用所有 DataPart 上的优化，例如为词典增加 BloomFilter

```
DESCRIBE TABLE mergeTreeProjection(default, table, projection_index)

┌─name─────────┬─type──────┬─default_type─┬─default_expression─┬─comment─┬─codec_expression─┬─ttl_expression─┐
│ term         │ String    │              │                    │         │                  │                │
│ posting_list │ PostingList │              │                    │         │ PostingListZSTD  │                │
└──────────────┴───────────┴──────────────┴────────────────────┴─────────┴──────────────────┴────────────────┘
```

性能对比

```
CREATE OR REPLACE TABLE hits_url (URL TEXT NOT NULL)
ENGINE = MergeTree ORDER BY ();
--
INSERT INTO hits_url SELECT URL FROM hits;

Elapsed: 28.690 sec. Processed 100.00 million rows
Peak memory usage: 559.48 MiB.
```

```
SELECT count() FROM hits_url
WHERE hasToken(URL, 'http')
SETTINGS max_threads = 1

┌─count()─┐
│ 98773547 │
└──────────┘
```

```
SELECT count() FROM hits_url
WHERE hasToken(URL, 'non_exist_url')
SETTINGS max_threads = 1

┌─count()─┐
│       0 │
└──────────┘
```

	No Index	Inverted Index V1	Inverted Index V2	Projection Index
写入性能	28.69s/559.48MB	326.41s/1.19GB	158.52s/562.13MB	115.3s/560.82MB
重建性能	-	339.57s/850.62MB	147.91s/160.24MB	107.21/100.42MB
存储容量	-	1.8GB	2.5GB	1.1GB
高召回查询时延（单线程）	6.67s	6.52s	0.21s	0.22s
低召回查询时延（单线程）	6.93s	0.02s	0.07s	0.04s

Projection Index 总结

更优的写入与重建性能，更少的存储空间，取得最好的查询性能

- 当前进展
- Projection Index Step 3：实现 Projection Index Reader，已提交 PR，review 中
- Projection Index Step 4：实现 Projection Index Expression Analyzer，物化表达式，PR 已 ready
- Full Text Projection Index：基于 Lucene 结构构建 PostingList，PR 准备中
- Full Text with PhraseSearch：PostingList 存储词位置信息用于短语检索，PR 准备中
- ...

感谢倾听