ByteHouse | 火山引擎数智平台

云原生数据仓库

ByteHouse

性能白皮书（企业版）

2024.03

在数据处理和分析的领域，提升查询效率始终是一项关键挑战。在 OLAP 领域，性能的关键需求在于能够快速进行数据检索，支持实时分析，具备处理大规模数据的能力，轻松应对复杂查询，提供快速响应，具备良好的可扩展性，高效处理并发操作，以及实现高效的数据压缩和存储。这些方面对于满足高效、准确的数据分析需求至关重要。

ByteHouse 是火山引擎自主研发的云原生数据仓库产品，它全面继承了开源 ClickHouse 的高性能和强大的分析能力，并在架构上遵循新一代云原生理念进行全面重构，实现了容器化、存储计算分离、多租户管理和读写分离等功能。在可扩展性、稳定性、可运维性、性能以及资源利用率等方面都有显著提升。截至 2022 年 2 月，ByteHouse 在字节跳动内部的部署规模超过 18000 台，单集群超过 2400 台。它经过了内部数百个应用场景和数万用户的锤炼，并在多个外部企业客户中得到了广泛应用。

本文将介绍 ByteHouse 企业版的一系列优化措施。这些改进旨在缩短查询执行时间、优化资源利用，提供更流畅的数据分析体验。通过智能优化算法和先进的执行技术，ByteHouse 能够更好地应对各种复杂的查询场景。

为了让大家亲身感受这些优化带来的效果，我们提供了使用 SSB 100G、TPC-H 100G、TPC-DS 100G 数据集的性能测试步骤。您可以按照这些步骤进行测试，亲自验证 ByteHouse 企业版在查询效率方面的显著提升。

目 录
CONTENTS

数据分析场景

实时数仓 01
1.1 Kafka & Flink 01
1.2 CDC（Change Data Capture）实时数据同步 01
1.3 Upsert & 部分列更新 02

复杂查询 02
2.1 优化器 02
2.2 执行层 04

宽表查询 06
3.1 预聚合 06
3.2 Zero copy 优化 06
3.3 全局字典 07
3.4 UncompressedCache 优化 07

人群圈选 & 用户画像 07
4.1 BitEngine/BitMap 07

行为分析 08
5.1 留存分析函数 08
5.2 漏斗分析函数 09

高并发场景 10

Vector 向量检索 10

GIS时空分析 12

标准测试集性能参照

SSB 性能测试 15
1.1 测试结果 15
1.2 测试环境 17
1.3 测试步骤 17

TPC-H 性能测试 39
1.1 测试结果 39
1.2 测试环境 41
1.3 测试步骤 41

TPC-DS 性能测试 57
1.1 测试结果 57
1.2 测试环境 64
1.3 测试步骤 64

数据分析场景

实时数仓

Kafka & Flink

HaKafka：更稳定的高可用 Kafka 消费引擎

ByteHouse 的 HaKafka Engine 是一款自研表引擎，在数据实时消费性能不降级的基础上解决了 Kafka 消费的高可用问题，提供了 low-level 消费模式，保证了 At-least-once 消费语义。用户可以通过 ByteHouse 控制台可视化创建实时导入任务。

Flink Connector

Flink Connector for ByteHouse 连接器专用于通过 Flink 将数据加载到 ByteHouse，目前 Flink Connector 已经支持 通过 Table API&SQL 和 Flink DataStreamAPI 两种方式来连接 ByteHouse 并处理数据，详情请参见产品手册（https://www.volcengine.com/docs/6464/1198033）。

CDC （Change Data Capture ）实时数据同步

DES 数据快车服务（ByteHouse 插件）

数据快车服务（DES，Data Express Service）是一个用于将多源异构数据源和数据结构导入到 ByteHouse 的服务，通过提供数据集成、结构映射、高效导入、安全可靠等功能，帮助用户快速、准确地将各种类型的数据（如关系型数据库、日志文件、对象存储等）导入到 ByteHouse 中进行后续的处理和分析。

使用 DES 可实现数据秒级同步到目标端，用户可以根据业务需求选择不同规格的独享资源以享受更高性能的同步体验，同步性能可达到 25 万 records/s 以上，详情请参见产品手册 。目前，数据快车的 CDC 同步任务已支持 MySQL / PostgreSQL 数据源的历史或增量数据同步。

内置 MaterializedMySQL 引擎

为了强化实时数仓的能力，便于将 MySQL 中的表映射到 ByteHouse 企业版中，ByteHouse 引入了 MaterializedMySQL 数据库引擎，ByteHouse 服务作为 MySQL 副本，可以读取 Binlog 并执行 DDL 和 DML 请求，实现了基于 MySQL Binlog 机制的业务数据库实时同步功能。

ByteHouse 企业版在实现 MaterializedMySQL 时，底层引擎采用了自研的 HaUniqueMergeTree 引擎，支持自定义版本字段以及根据 UNIQUE KEY 实时删除数据功能，无需引入其他额外字段。同时，ByteHouse 增强了 MaterializedMySQL 引擎的稳定性和易用性。

Upsert & 部分列更新

当前，很多企业在分析型业务中对数据去重有很强的诉求，为此，ByteHouse 研发了 HaUniqueMergeTree 表引擎，保留了 ClickHouse 高效的查询性能，还支持主键实时更新的场景，ByteHouse 在落盘即去重，解决了社区版 查询时去重的效率问题，可以帮助业务更轻松地开发实时分析应用。

同时，在 ByteHouse 中，我们可以在 HaUniqueMergeTree 表中实现部分列更新的功能。部分列更新模式在指定变量 enable_unique_partial_update = 1 后，允许以部分列更新的模式进行写入。

```
SQL
-- 以部分列更新的模式进行写入，并主动制定更新列示例
SET enable_unique_partial_update = 1;

-- 主动指定 _update_columns_
INSERT INTO t1 (k, c1, m1, a1, _update_columns_) VALUES (1, 20, {'k2':2}, ['world'], 'k,c1,m1,a1');
┌─k─┬─c1─┬──c2──┬─c3───┬──c4──┬─m1──────┬─a1──────┐
│ 1 │ 20 │ 3.14 │ NULL │ NULL │ {'k2':2} │ ['world'] │
└───┴────┴──────┴──────┴──────┴─────────┴─────────┘
```

复杂查询

优化器

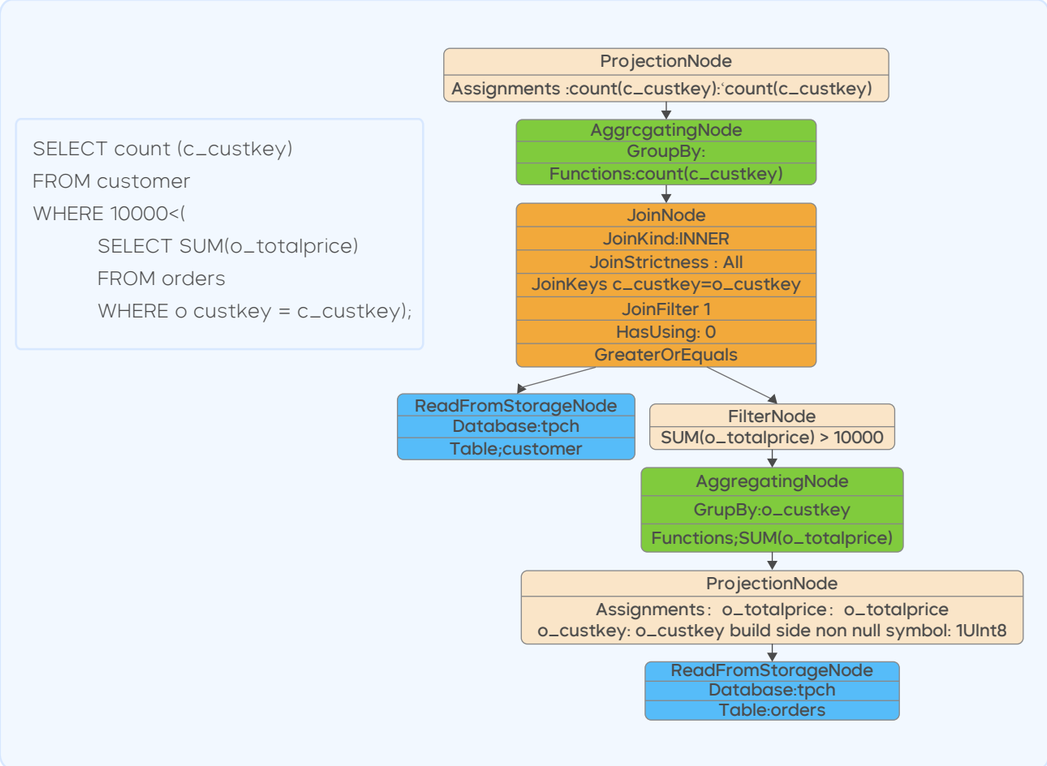
数据库优化器是DBMS中一个核心组件，它负责分析查询语句，并根据表的结构、索引等信息来生成最优的执行计划。通过优化查询执行计划，可以提高查询的执行效率，减少资源消耗，提升系统性能。为了提升在复杂场景的查询性能，ByteHouse 的自研优化器进行了大量的优化，主要包括四个大的优化方向：RBO（基于规则的优化能力），CBO（基于代价的优化能力），分布式计划优化以及一些高阶优化能力。

RBO

优化器实现了常见的优化规则。例如列裁剪、分区裁剪、表达式简化、子查询解关联、谓词下推、冗余算子消除、Outer-JOIN 转 INNER-JOIN、算子下推存储、分布式算子拆分等常见的启发式优化能力。

解关联

很多 OLAP 引擎不支持相关子查询，在语法分析阶段就会报错。优化器实现了完整的解关联能力，对于关联查询可以转换为常见的 join agg filter 等算子执行，下图就是一个简单的解关联例子。对于一些特殊类型的关联查询也可以利用 window 算子执行，更加快速简洁。



非等值 Join 优化

在很多引擎中，带有非等值条件的 join 需要通过多个算子来组合执行(inner join + filter + group-by)，而在 ByteHouse 中，支持非等值 join 之后可以直接在 join 算子中完成非等值条件的执行。

优化器会对一些关联子查询转成非等值 join 来执行，相较于转成其他常见的算子（inner join, filter, agg）性能有一倍以上的提升。

CBO

基于 Cascade 搜索框架，利用 Graph Partition 技术实现了高效的 Join 枚举算法，以及基于 Histogram 的代价估算。

Join Reorder

对 10 表级别规模的 Join Reorder 问题，能够全量枚举并寻求最优解，同时针对大于 10 表规模的 Join Reorder 支持启发式枚举并寻求最优解。CBO 支持基于规则扩展搜索空间，除了常见的 Join Reorder 问题以外，还支持 Outer/Semi/Anti-Join 的 Reorder。

分布式计划生成

业界主流实现分为两个阶段，首先寻求最优的单机版计划，然后将其分布式化。但是这样的设计流程，不能提前考虑分布式系统的特点，可能会导致网络延迟、数据分布不均衡，并导致可扩展性限制等问题。我们的方案则是将这两个阶段融合在一起，在整个 CBO 寻求最优解的过程中，会结合分布式计划的诉求，从代价的角度选择最优的分布式计划。同时在 Join/Aggregate 过程中，也支持 Partition 属性展开。

另外，我们也在 CBO 中实现了对于 Aggregate/Join Reorder，Magic Set Placement 等相关能力。对于 CTE 的实现方式也基于 Cost 进行选择，在 inline，shared 和 partial inline 之间做权衡，选出最优的计划。在 tpcds 等 benchmark 中都有一定的应用。

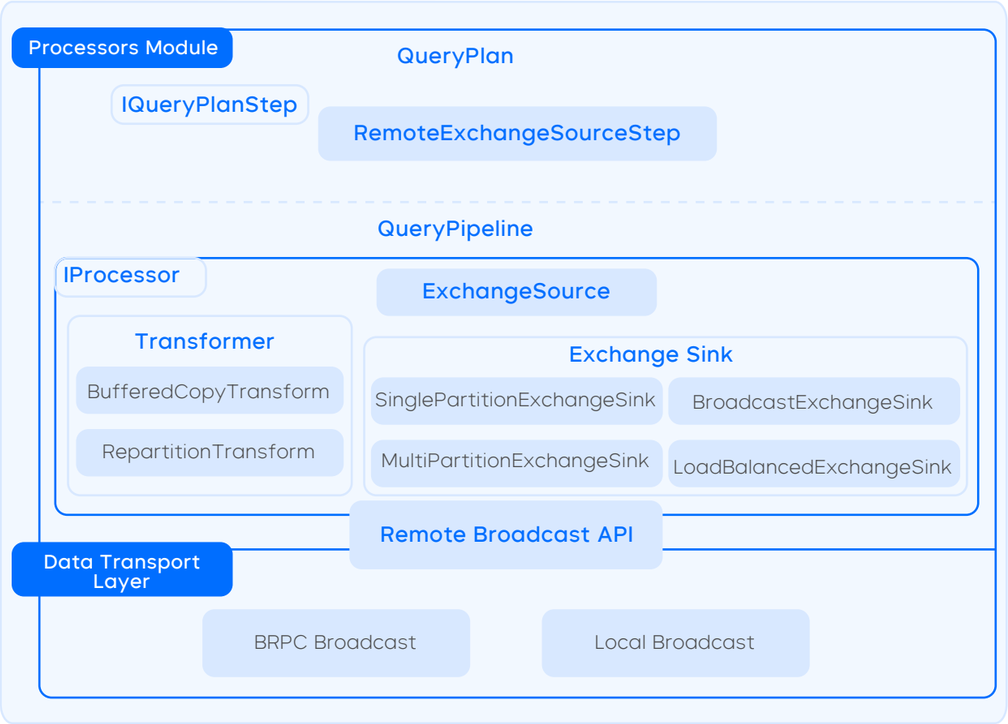
执行层

Exchange

在执行层中，数据的传入和传出依赖于 Exchange 模块。Exchange 是数据在 PlanSegment 实例之间进行数据交换的逻辑概念。在具体实现上，我们将其分为数据传输层和算子层。

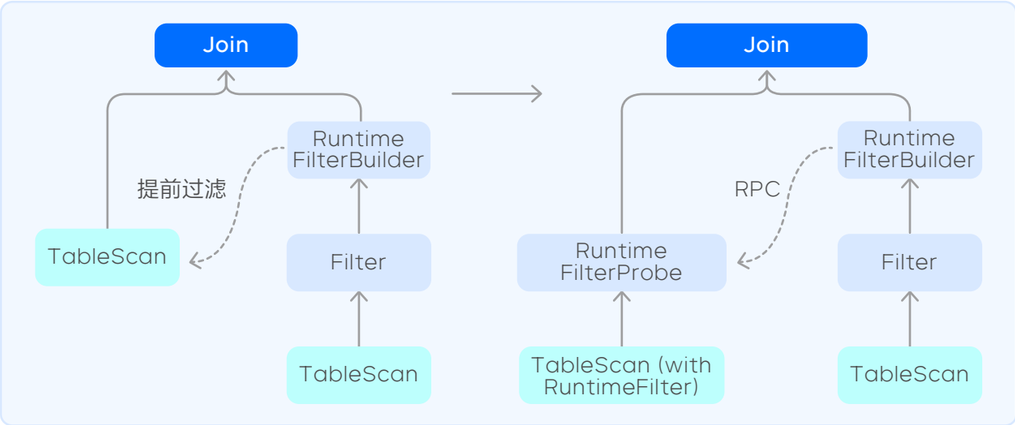
数据传输层主要基于定义的 Receiver/Sender 接口，同进程传输基于队列，跨进程基于 BRPC Stream，支持保序、状态码传输、压缩和连接池复用。简单来说，就是在大集群上的两个节点之间只建立一个连接，所有查询都在这个连接上通信。由于我们使用连接池，实际上两个节点之间是固定数量的连接，这样稳定性更高。

算子层支持四种场景：一对多的 Broadcast、多对多的 Repartition、多对一的 Gather，以及本进程之间的 Round-Robin。ByteHouse中另外还做了一些优化，包括避免 Broadcast 重复序列化、提升 Repartition 性能以及 sink 攒批。在大集群下，通过一个 ExchangeSource 读取多个 receiver 的数据来降低线程数。



RuntimeFilter 优化

在 OLAP 场景中，Join 是一个十分常见的操作，也是查询的性能瓶颈。Runtime Filter 是在 Join 的 Probe 端提前过滤掉那些不会命中 Join 的数据，减少存储扫描、网络传输和算子计算的数据，是优化复杂查询的重要手段之一。



Runtime Filter 也称为 Dynamic Filter，是一种在执行引擎中动态构建 Filter 的能力。它的目的是通过在 Join 的 Probe 端提前过滤掉那些不会命中 Join 的输入数据，来大幅减少 Join 中的数据传输和计算，从而减少整体的执行时间。

例如在 Hash Join 的过程中，当右表 HashTable 构建完成后，将根据 JoinKey + HashTable 生成 RuntimeFilter，并将其作用于左表的 Scan 节点，从而减少左表的扫描数据量，可进一步减少磁盘 IO、网络数据传输等，最终使得 Query 的性能显著提高。

ByteHouse 支持根据不同的场景生成最优的 RuntimeFilter，优化了生成和 Apply 的流程，同时支持 Distributed 和 Local 的 RuntimeFilter，在较大规模集群上也自适应的支持 Shuffle-Aware 的 RuntimeFilter。

并行化重构 +Bucket 表

ClickHouse 内部的并行执行在 Agg 和 Join 上存在一些性能瓶颈，主要表现在不能很好地结合数据上下游的数据特征，以及在中间执行过程中引入了不必要的 Final Merge 和数据 Shuffle。

在此基础上，ByteHouse 对 ClickHouse 进行了并行化重构，支持 Aggregation Streaming 并结合 HashTable Two-Level 的并行和数据的分布特征，有效减少了中间出现的 Agg Final Merge 和 ConcurrentHashJoin 中的数据 Shuffle，从而显著提升了整体 Query 的执行性能。

宽表查询

预聚合

物化视图

物化视图存储了 SQL 查询语句包含的数据，并提供更新机制。物化视图最重要的功能就是查询加速。用查询物化视图来替代直接查询数据表，可以避免对数据进行再次的计算与聚合，能够以空间换时间的方式节省查询时间，达到查询加速和简化查询逻辑的目的。

数据仓库中存在大量在大型表上执行复杂的查询，并会消耗大量资源和时间。物化视图可以通过预计算的结果回答查询，消除昂贵的 Join 和聚合计算所带来的开销，大幅度改善查询处理时间，降低系统负载。对于可以预见并反复使用相同子查询结果的查询，物化视图非常高效。

Projection

Projection 是按照 ByteHouse 的存算分离架构进行设计的，Projecton 数据由分布式存储统一进行管理，而针对 projection 的查询和计算则在无状态的计算节点上进行。相比于社区版，ByteHouse Projection 实现了以下优势：

- 对于 Projection 数据的存储节点和计算节点可以独立扩展，即可以根据不同业务对于 Projection 的使用需求，增加存储或者计算节点。
- 当进行 Projection 查询时，可以根据不同 Projection 的数据查询量来分配计算节点的资源，从而实现资源的隔离和优化，提高查询效率。
- Projection 的元数据存储十分轻量，在业务数据急剧变化的时候，计算节点可以做到业务无感知扩缩容，无需额外的 Projection 数据迁移。

Zero copy 优化

Zero copy 是一种优化技术，用于在数据传输过程中减少数据的拷贝次数，从而提高数据传输的效率和性能。 Zero copy 可以避免原来前期需要的大量 deep copy 操作。在减少 deep copy 降低查询计算 overhead 的同时，也可以避免内存 IO 吞吐过早达到内存 IO 带宽上限，从而缓解高并发下内存 IO 吞吐带宽瓶颈带来的问题。

全局字典

全局字典以全局字典编码的方式进行数据的读写计算，针对 Agg、Function和Exchange 实现了基于编码值的处理，将基于非定长字符串类型的计算改变为基于定长整型编码值的计算，以此提升计算效率。

UncompressedCache 优化

UncompressedCache 优化提高了多线程下 cache 访问的效率。社区 OLAP 多个 cache 模块使用 LRU cache，在多线程并发场景中存在锁竞争激烈，影响 cache 效率的情况。而 UncompressedCache 优化解决了这一问题，提高了系统的性能和响应能力。

人群圈选 & 用户画像

BitEngine/BitMap

BitEngine 是一个高效集合数据 处理模型，它是查询分析数据库 ClickHouse 的一部分。BitEngine 底层基于 MergeTree Family 存储引擎，并在此基础上引入了 BitMap64 类型，开发了系列相关运算函数。BitEngine 提供的 BitMap64 类型适合表达具有特定关系的大量实体 ID 的集合，将集合的交并补运算转化为 bitmap 之间的交并补运算，从而达到远超普通查询的性能指标。已上线业务的测试表明，使用 BitEngine 相比普通和 Array 或者用户表方式，在查询速度上有 10-50 倍不等的提升，详情可参见官方文档（https://www.volcengine.com/docs/6464/1172136）。

BitMap 索引（BitMap index）目前用于 Array + Int8/16/32/64/String UInt/8/16/32/64 / String 类型，它为 Array 数组中每个元素构建一个 bitmap-index，在查询的时候，我们使用 arraySetCheck(column, (item)) 检测 item 是否在数组里的时候会使用 bitmap-index 做过滤，从而避免大量的 array 数据读取，目前大部分 Array 类型的过滤操作都使用了 bitmap-index，性能有 4-10 倍不等的提升。

人群圈选分析是客户画像平台(CDP)中的核心功能。分析师利用各种标签组合，挑选出最合适的人群，进而进行广告推送，达到精准投放的效果。同时由于人群查询在不同标签组合下的结果集大小不同，在一次广告投放中，分析师需要经过多次的逻辑调整，以获得"更合适"的人群包。在这种高频的操作下，画像平台通常会遇到两方面的问题：

- 第一，由于此类查询分析是临时性的，各种标签组合数巨大，离线预计算无法满足此类灵活性。
- 第二，由于此类查询是实时场景，查询性能变得非常关键， 通常一次查询在分钟级，耗时较长，无法满足分析师需求。

目前，从 ByteHouse 实测数据表现上看，在 10 亿级用户测试数据下，ByteHouse 的人群查询 P99 小于 10s，展现了优异的性能。

行为分析

随着经营成本的大幅增加，企业必须改变过去粗放型的营销与运营方式，尤其是在产品制造、销售，市场营销以及客户服务等各个方面都需要向更加科学、高效的方向转变。行为分析是数字化营销中非常重要的一个指标。以电商业务的用户运营为例，通常需要整合各类业务数据，按照用户粒度进行数据加工，并形成结构化的数据，继而进行标签计算，以便能够精准描绘画像特征。

ByteHouse 根据用户行为分析使用场景，定制了部分函数，包括留存分析类函数 genArrayIf()/ retention2()、路径分析类函数 pathSplit()/ pathCount()，以及漏斗分析函数 finderFunnel()/ funnelRep() 等。漏斗分析和留存分析是常用的转化分析方法，在用户行为分析和 App 数据分析的流量分析、产品目标转化等数据运营和分析中得到广泛应用，使用这些函数相比拼装 SQL 或者 ClickHouse 原生函数更为高效。

留存分析函数

留存分析函数通常用于分析用户在一段时间内的留存情况。这些函数可以帮助我们了解用户在特定时间段后是否仍然活跃或继续使用产品或服务。

通过对留存率数据分析数据，可以识别用户行为模式，并通过持续不断的优化产品和数据监测，可以提高用户满意度和忠诚度，增加留存率。