51CTO WOT

World Of Tech 2024

WOT全球技术创新大会

2024

智启新纪
慧创万物

51CTO WOT

新一代云原生数仓的设计原理与最佳实践

董一峰
火山引擎
ByteHouse 技术专家

ByteHouse 是字节跳动旗下产品，多年业务打磨、技术积累的输出载体

数据库 Database

基础型软件，全行业通用

泛互（电商、游戏、web3）、金融、汽车、零售消费、制造 ……

分析型 OLAP

用户画像　　人群圈选

行为分析　　IoT 分析

实时数仓　　实时风控

……

分布式 Distributed

2000+ Node

并行计算MPP

云原生 Cloud native

存算分离

容器化（计算资源）

灵活弹性 – 秒级、在线、业务无损

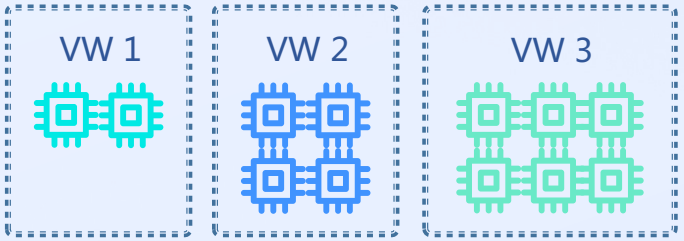
ByteHouse 适用场景

实时吞吐（IoT）	BI 报表	离/在线复杂分析	湖+仓 联邦分析	人群圈选 & 行为分析	以图搜图（向量）
- Upsert，部分列更新 - 自研uniqueMergeTree引擎，写入即去重	- 预聚合：MV物化视图（单/多表）、Projection - 结果/中间结果/数据cache	- 自研优化器：CBO、RBO； - Runtime Filter 执行引擎的动态filter - 自研分布式缓存 - ELT：BSP，算子落盘	- Native Reader（ORC，Parquet） - DataFebric，外表多表物化 - 外表schema 自同步 - 优化器（CBO）	- BitEngine / BitMap64 - 内置分析函数：留存分析/路径分析/漏斗分析 等	- 算法库： hnswlib/faiss - 算法：HNSW、HNSW_SQ、Flat、IVF_Flat、IVF_PQ、IVF_PQ_FS+Refine等 - 高效向量检索执行链路：IO 优化 + 冗余计算消除 - Vector Index Cache 向量索引缓存

** 性能表现与资源规模强相关，上述性能表现来自于域内业务、toB客户生成环境真实表现*

ByteHouse 弹性能力

计算
On demand



PaaS 模式（容器化）

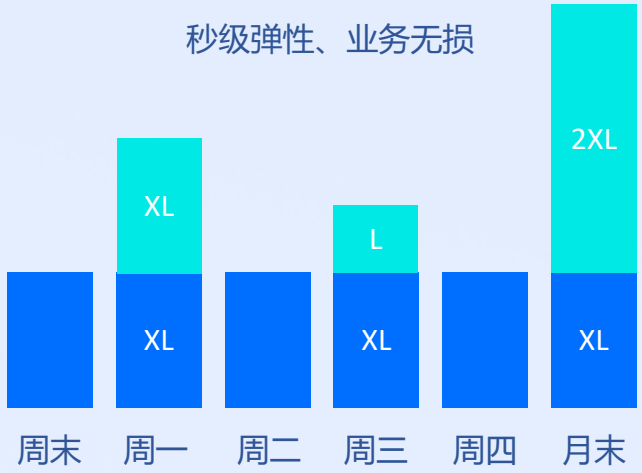
租户专属计算资源、 按需启停、
灵活定制弹性计划、无损业务

存储
Infinite

Serverless 模式

对象存储（低成本）、无限容量、
弹性按需

示例：弹性计划



性价比

随用随开，不用即停，账单更透明
按资源用量计费，用多少算多少

灵活弹性

容器化、计算Stateless
秒级弹性（扩容、缩容）
无损业务

性能稳定（SLA）

资源隔离，读写隔离、应用隔离
计算组资源PaaS化，租户专属

架构设计

服务层（Cloud Service）

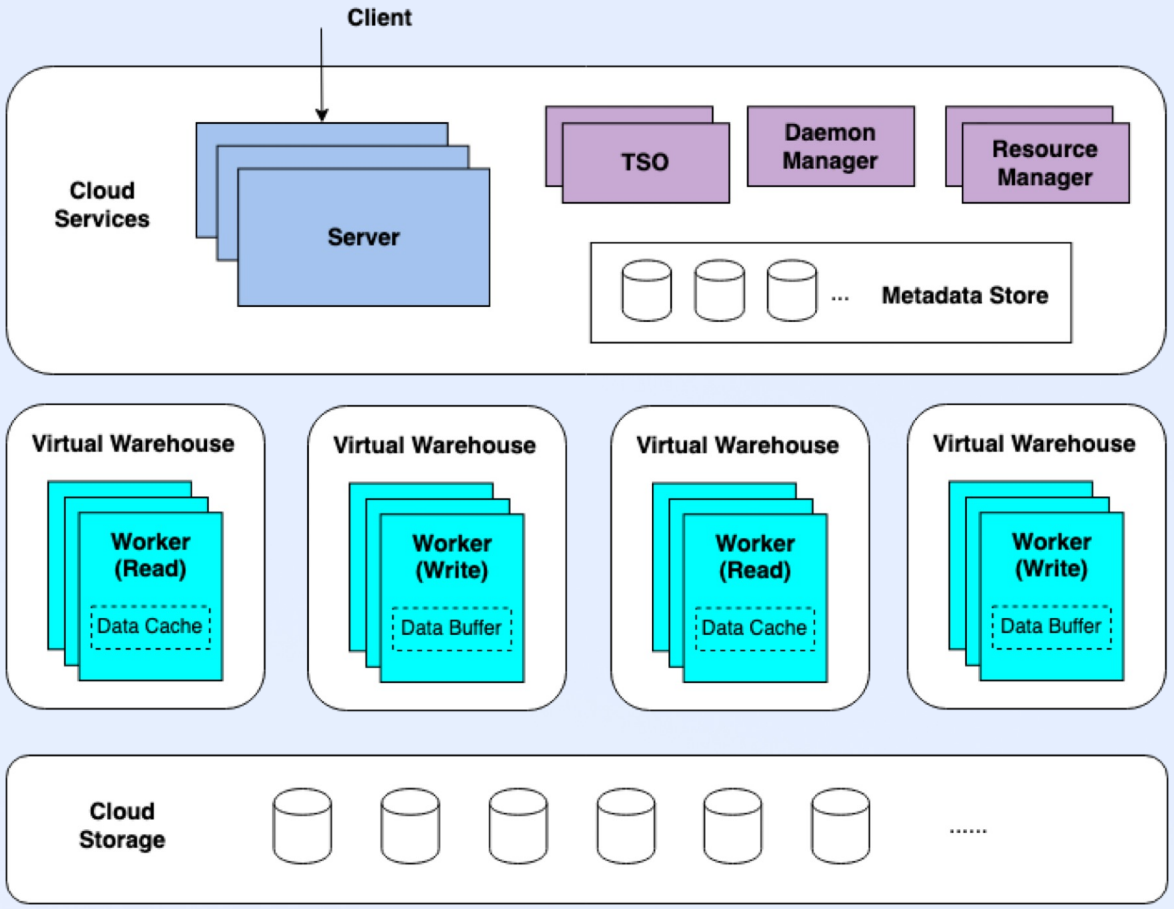
- MetaDate：FoundationDB/ByteKV
- Server：表元数据缓存、查询SQL解析、计划生成、调度和下发
- Resource Manager：服务发现、负载心跳检测
- TSO：全局唯一单调递增的时间戳
- Daemon Manager：调度和管理任务

计算组（Virtual Warehouse，VW）

- Worker：查询片段的执行，后台任务的执行、Local Disk Cache
- 每个表可以设置Read VW（查询）和Write VW（导入和Merge)

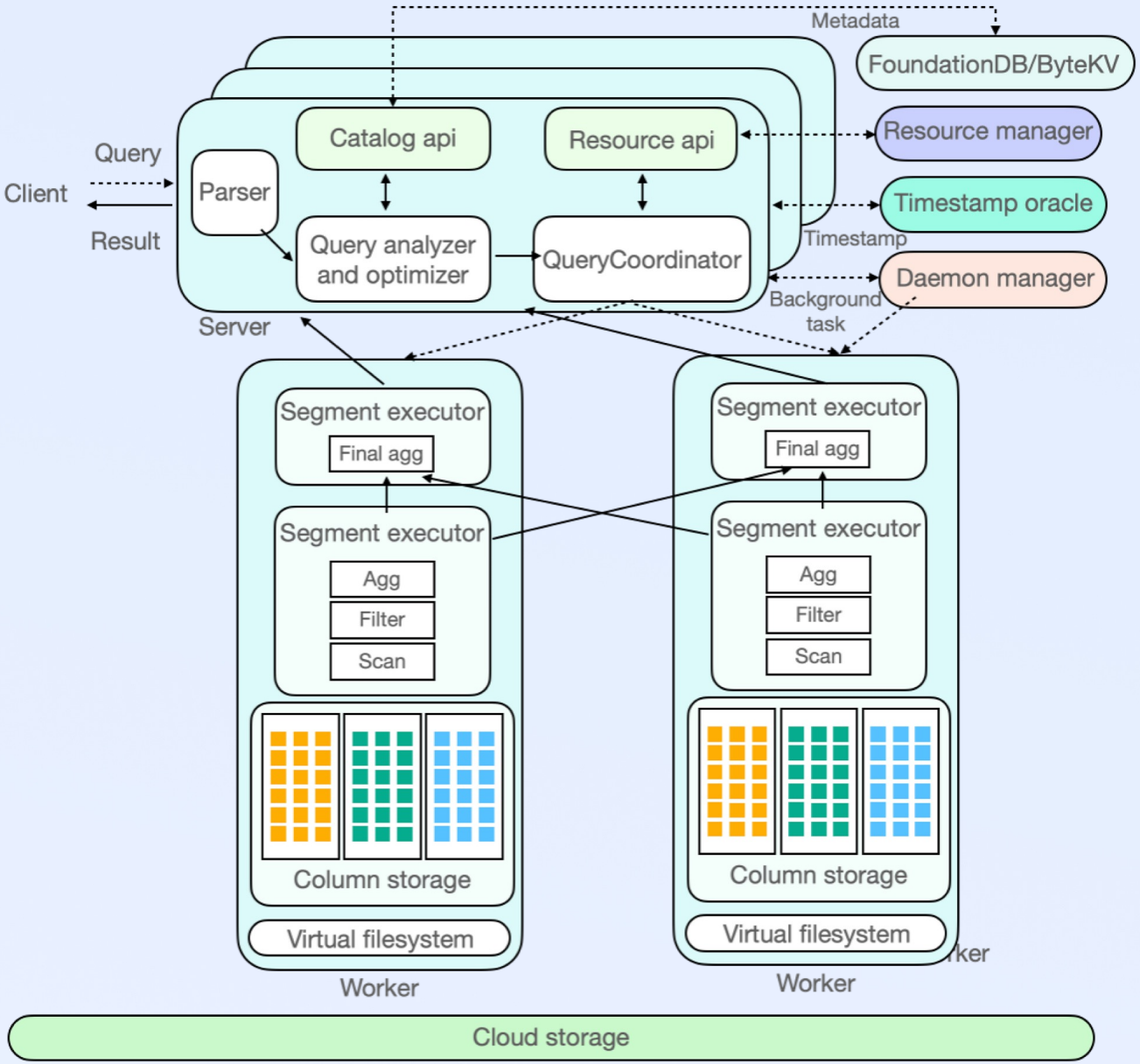
存储层（Cloud Storage)

- 支持HDFS、S3



ByteHouse 架构图

架构设计

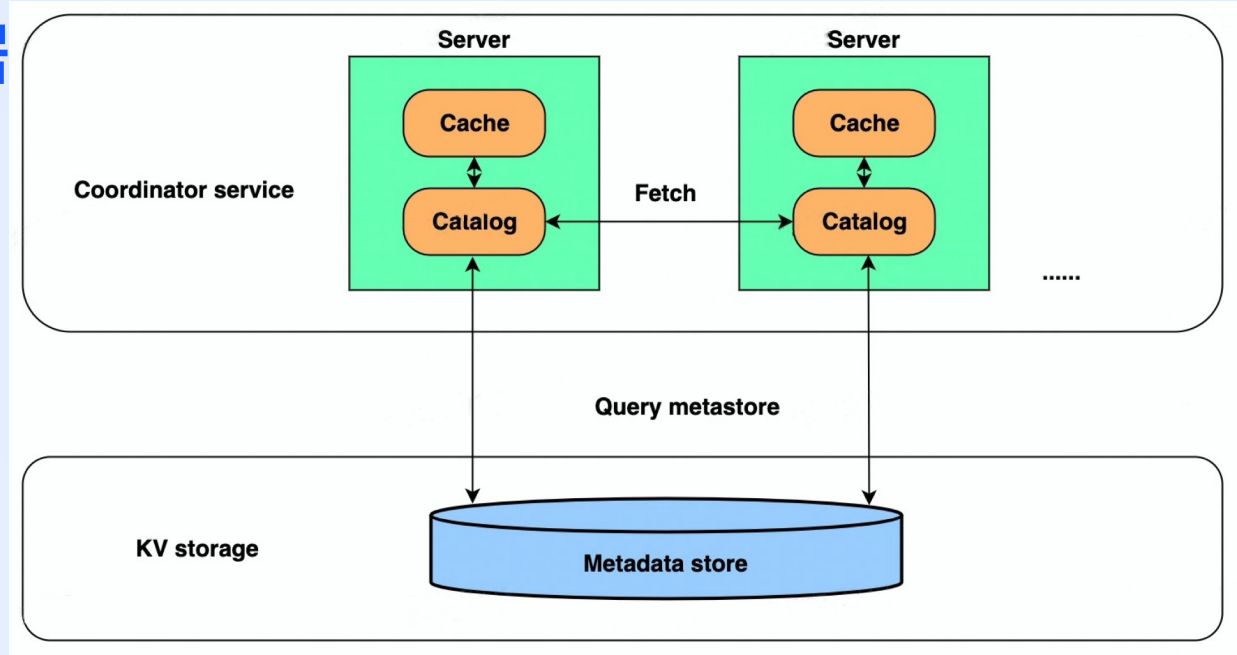

Metadata
FoundationDB/ByteKV
Catalog api
Resource api
Resource manager
Query
Client
Parser
Timestamp oracle
Result
Query analyzer and optimizer
QueryCoordinator
Timestamp
Daemon manager
Background task
Server
Segment executor
Final agg
Segment executor
Agg
Filter
Scan
Column storage
Virtual filesystem
Worker
Segment executor
Final agg
Segment executor
Agg
Filter
Scan
Column storage
Virtual filesystem
Worker
Cloud storage

存算分离的关键技术——元数据

- 需要统一的元数据管理系统
- 分布式文件系统在数据量大的情况下容易存在元数据管理压力
- 分布式统一存储系统大多不支持 rewrite，一些对象存储系统甚至不支持 append
- 分布式对象存储系统通常 move 代价比较高
- io latency 通常情况对比本地文件系统下都存在增加的情况

存算分离的关键技术——元数据

- 管理 Table/Part/Transcation 等数据
- 高效的 Part 元数据缓存管理
- 拓扑管理



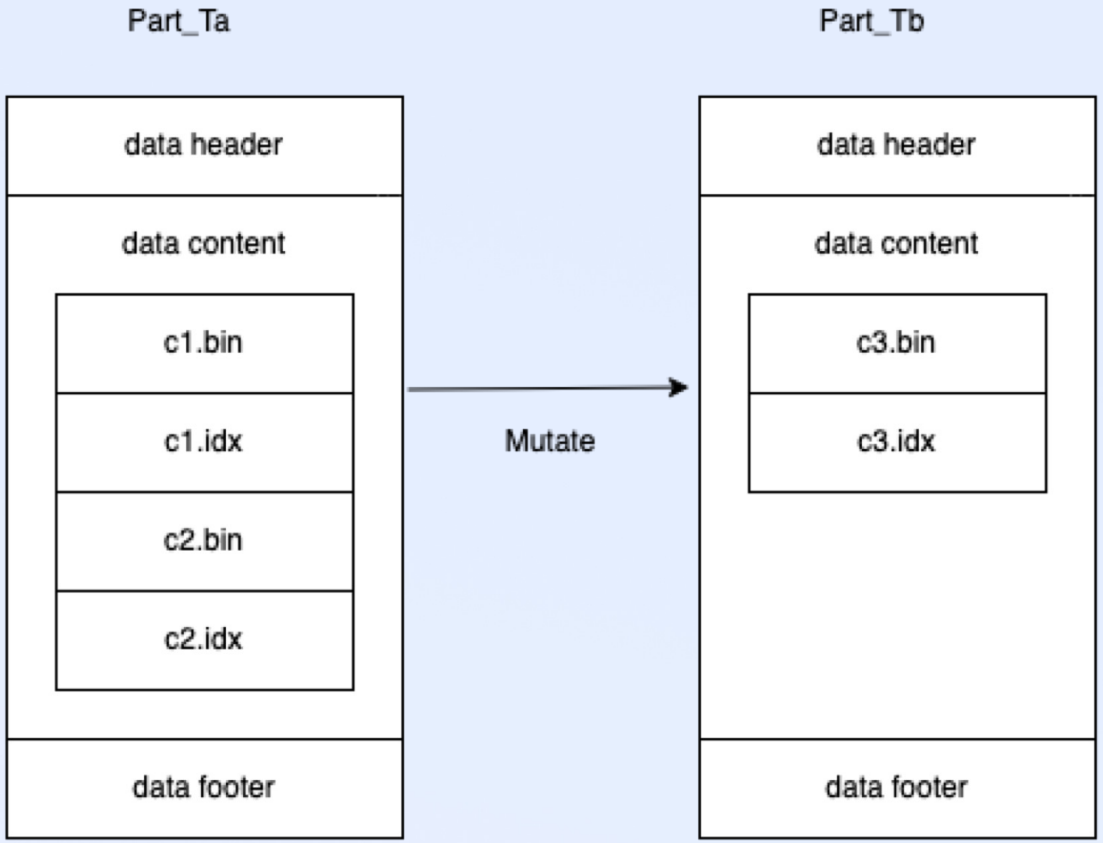
存算分离的关键技术——数据存储

- 每个 part 所有列数据存储在一个文件中，避免产生太多小文件
- footer 建立索引，快速定位数据位置

```
* Data information will be stored in cloud storage(e.g. s3 hdfs) as one data file,
* Which will not be updated once generated, only can be rewrited to new data file or be deleted.
*
* ------------data header---------
* magic_code(4 bytes)
* version(8 bytes)
* deleted(1 bytes)
* reserved size(256 - 12 bytes)
* -----------data content---------
* columns data files & idx files
* primary_index
* checksums
* metainfo
* -----------data footer----------
* primary_index offset(8 bytes)
* primary_index size(8 bytes)
* primary_index checksum(16 bytes)
* checksums offset(8 bytes)
* checksums size(8 bytes)
* checksums checksum(16 bytes)
* metainfo offset(8 bytes)
* metainfo size(8 bytes)
* metainfo checksum(16 bytes)
* unique_key_index offset(8 bytes)
* unique_key_index size(8 bytes)
* unique_key_index checksum(16 bytes)
* metainfo key (32 bytes)
* ------------------------------
*/
```

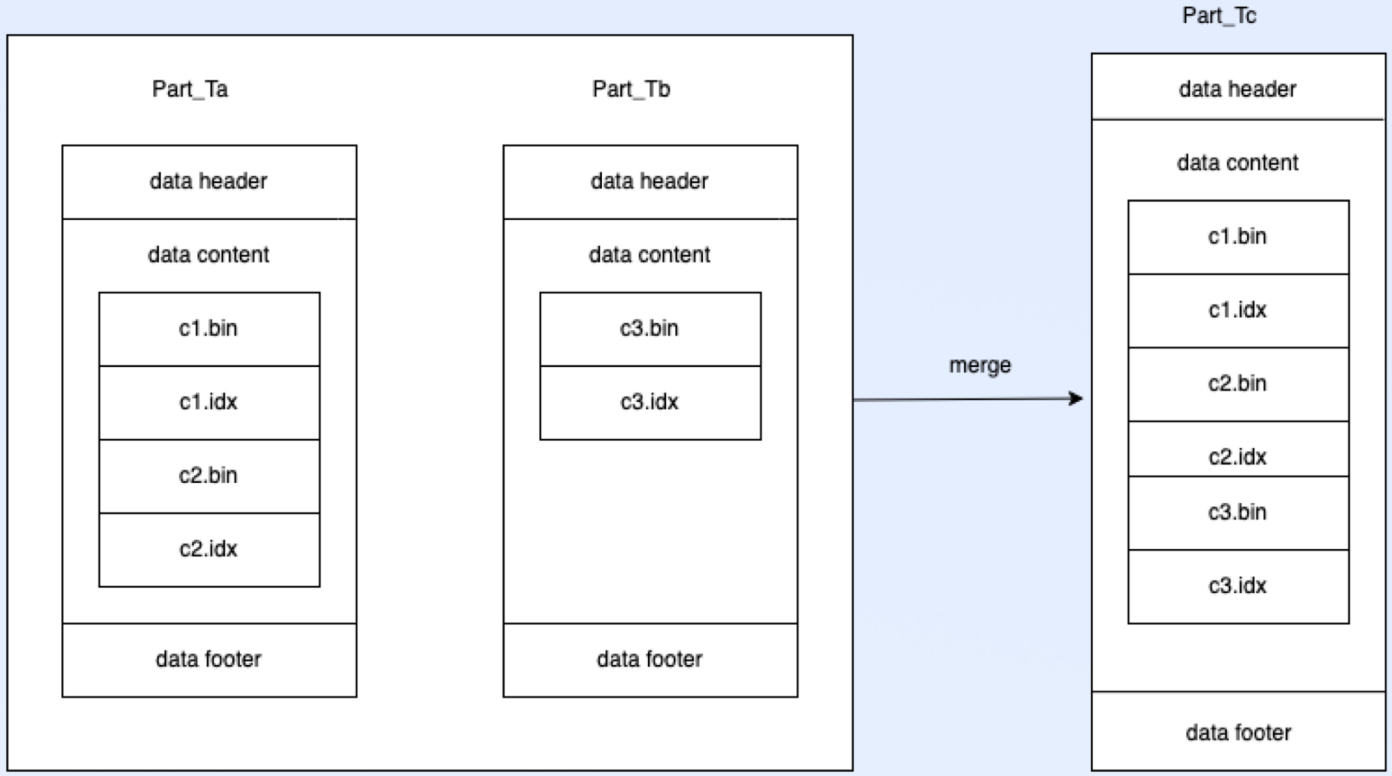
存算分离的关键技术——数据存储

- Part 数据文件生成后不可变
- delta part => part chain



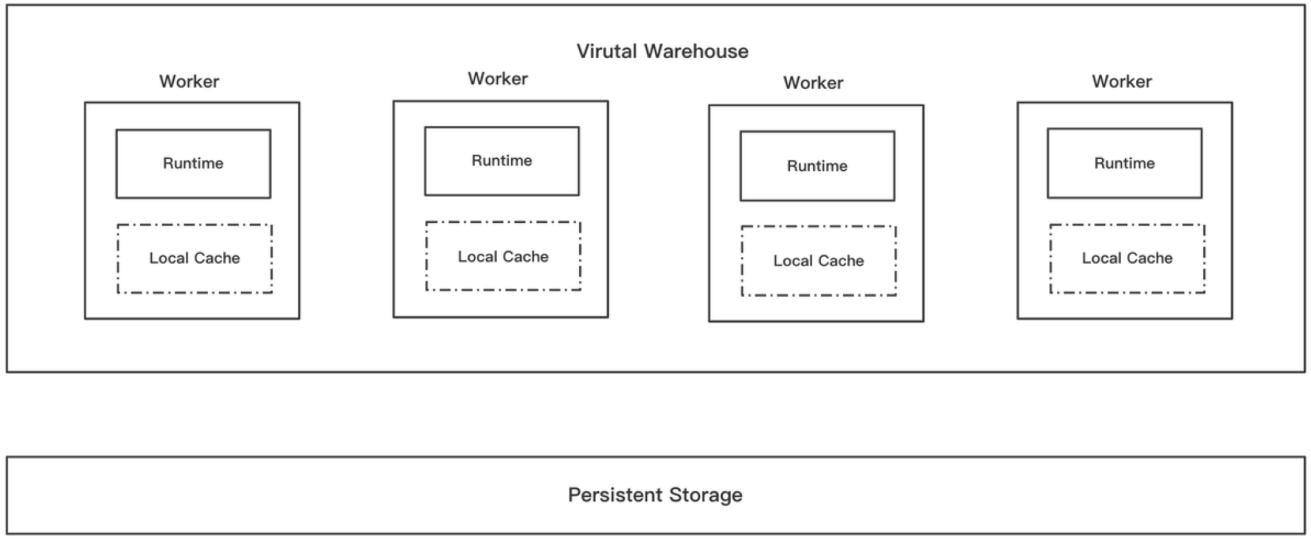
存算分离的关键技术——数据合并

- 异步 merge 合并数据
- 过期数据通过 gc 异步清理



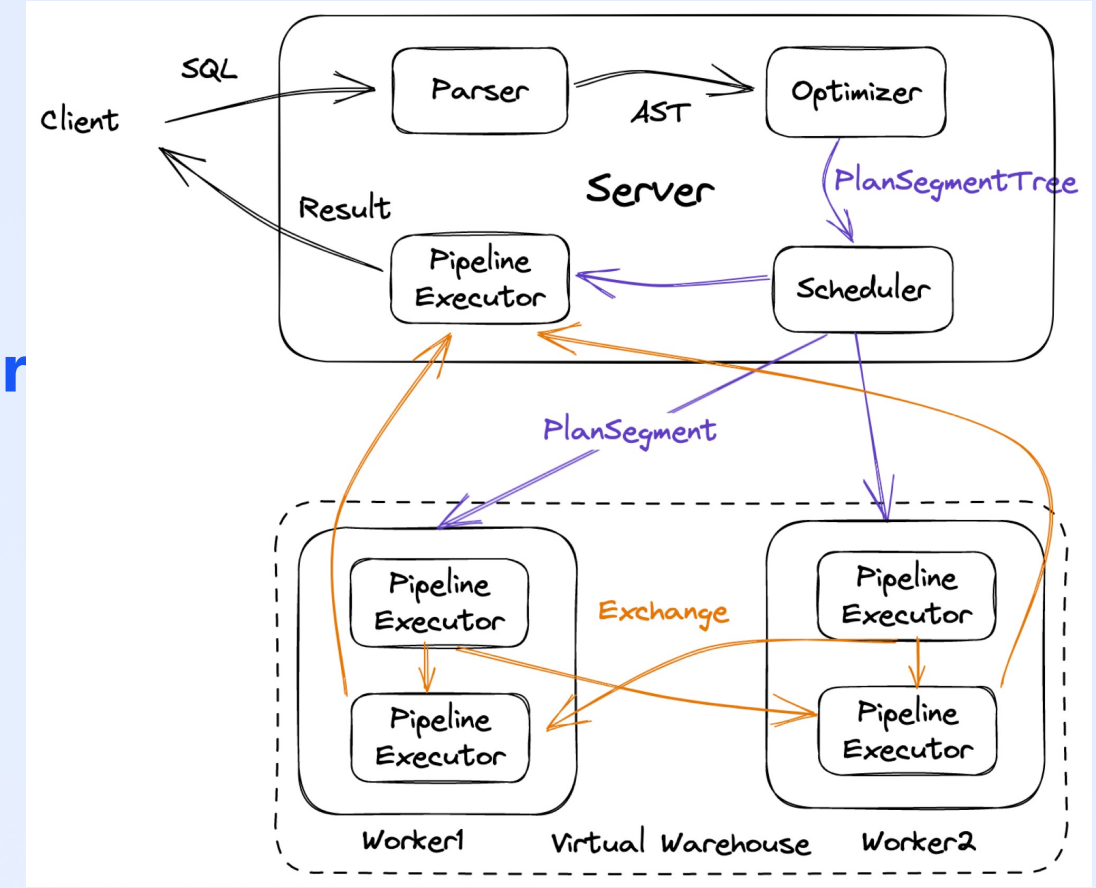
存算分离的关键技术——数据缓存

- 数据分配算法：一致性 hash
- 热数据 worker 节点自动缓存
- 优化缓存粒度和算法
- 避免数据 reshuffle



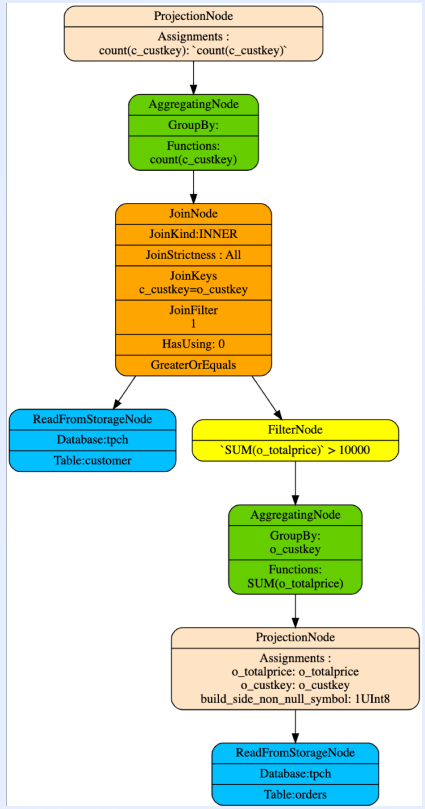
性能优化—分布式计划

- PlanSegment：分布式执行计划逻辑单元
- Optimizer: 生成 PlanSegmentTree
- Scheduler: 发送 PlanSegment 到 Worker
- Exchange: 在Pipeline之间传输数据



性能优化—优化器 RBO

```sql
SELECT count(c_custkey)
FROM customer
WHERE 10000 < (
    SELECT SUM(o_totalprice)
    FROM orders
    WHERE o_custkey = c_custkey);
```



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

- 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（如右图）。
- 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

- 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。
- 相对于通用场景中将非等值Join直接转成类似 Outer/Inner Join + Filter 算子组合，有显著的性能提升。

Multiple-Distinct优化

- 基于复制的方式提升多个不同列计算distinct的并行度。

性能优化—优化器 CBO

基于 Cascade 搜索框架，利用 Graph Partition 技术实现了高效的 Join Order 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
-

CTE

- 在 INLINE，SHARED 和 Partial INLINE 之间做权衡，选出最优的计划。

Magic Set Placement

- 对Aggregate/Join Reorder实现了相关能力

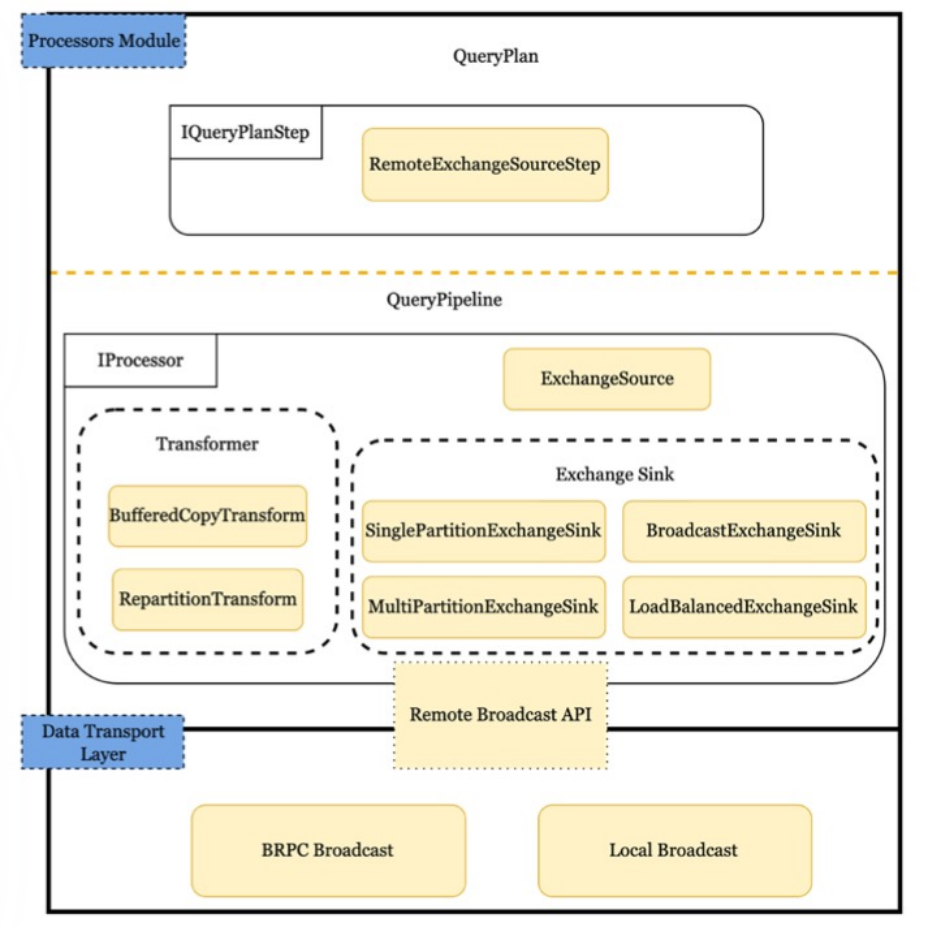
性能优化—Exchange

数据传输层

- 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。

算子层

- 支持 一对多的 Broadcast、多对多的 Repartition、多对一的 Gather、及进程内部 Round-Robin。
- 其他的优化项：避免 Broadcast 重复序列化、提升 Repartition 性能以及 Sink 攒批。



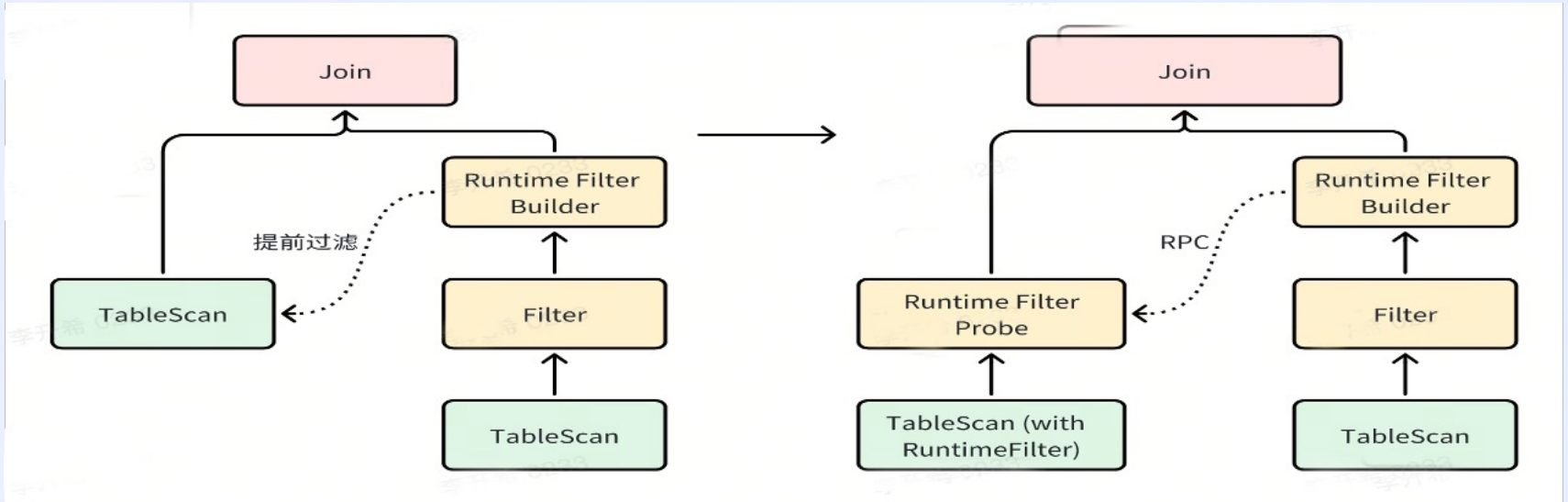
性能优化—Runtime Filter

功能描述

- Runtime Filter是在执行引擎中动态构建Filter 。如在 Hash Join 的 Probe 阶段前，提前过滤掉大部分不会参与 Join 的数据，减少数据传输和计算的开销，提升性能。

更多优化

- 自动生成最优的 RuntimeFilter，优化了 Runtime Filter 的生成和执行的流程。
- 支持 Distributed 和 Local RuntimeFilter，及自适应的 Shuffle-Aware 能力。



性能优化—编码和 Cache 优化

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，将字符串类型的计算改变为基于整型编码值的计算

Zero copy

- 在数据传输过程中减少数据的拷贝次数，提高数据传输的效率和性能。 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。

Uncompressed Cache 优化

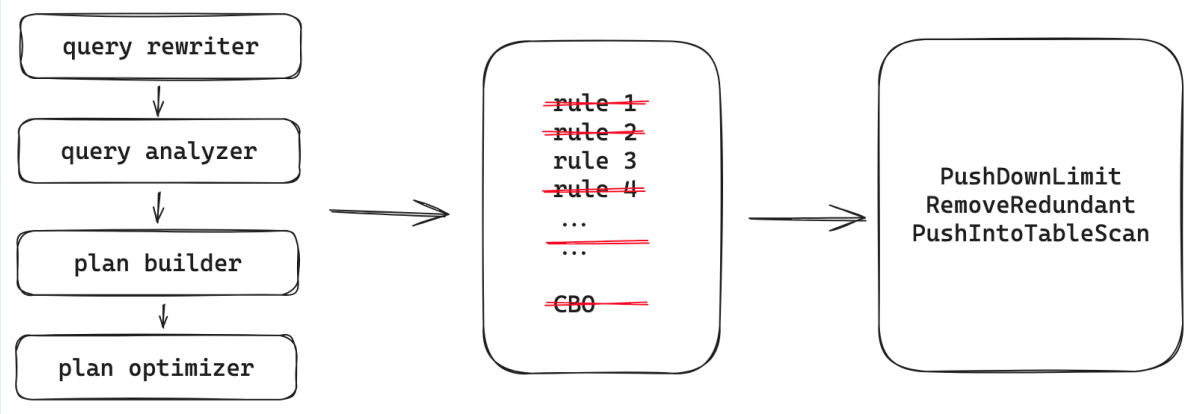
- 提高了多线程下数据 cache 访问的效率。CK 社区 OLAP 多个 cache 模块使用 LRU cache，在多线程并发场景中存在锁竞争激烈。

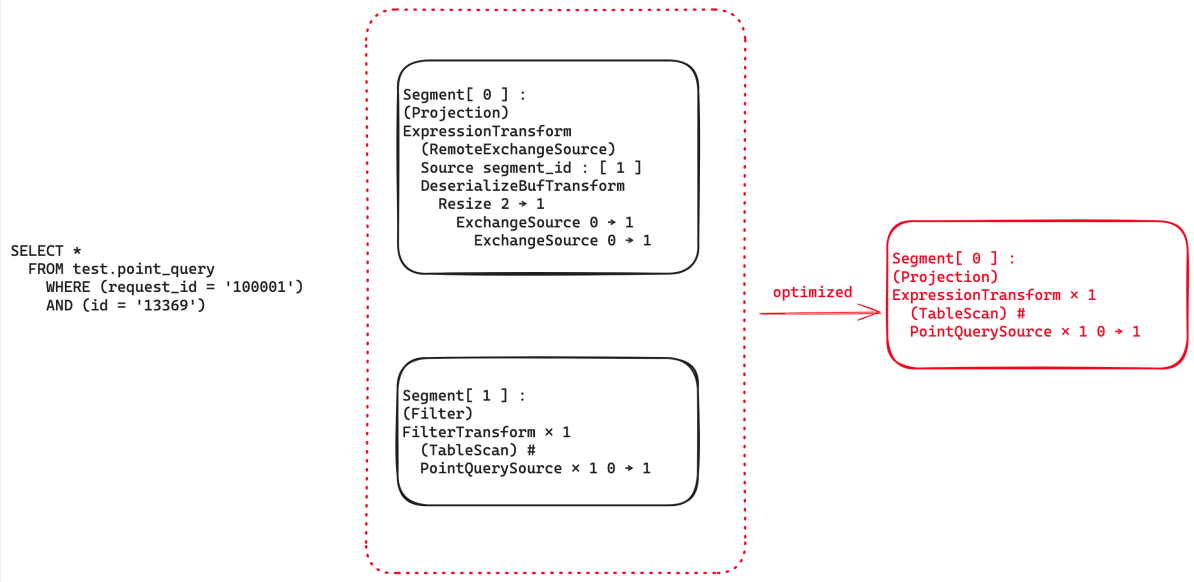
性能优化—高并发

核心优化点

简洁的短路执行计划

- 裁剪不必要的优化
- 简化 plansegment
- 裁剪冗余节点



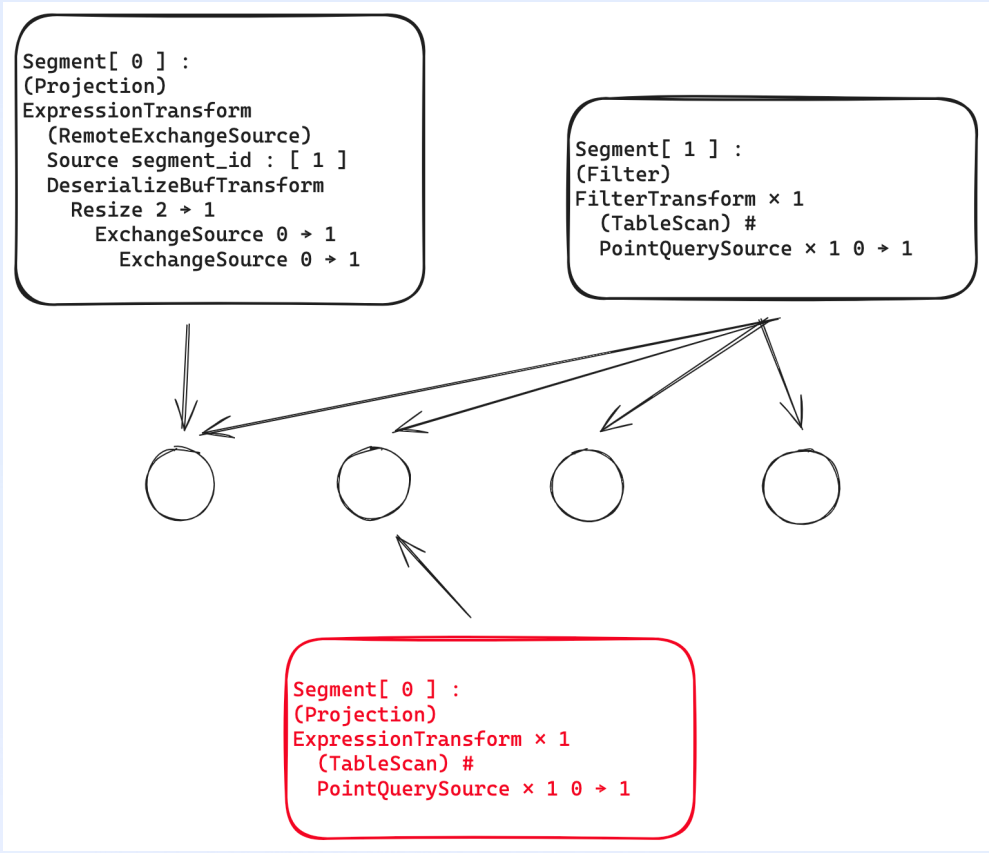


性能优化—高并发

核心优化点

简洁的短路执行计划

- 裁剪不必要的优化
- 简化 plansegment
- 裁剪冗余节点

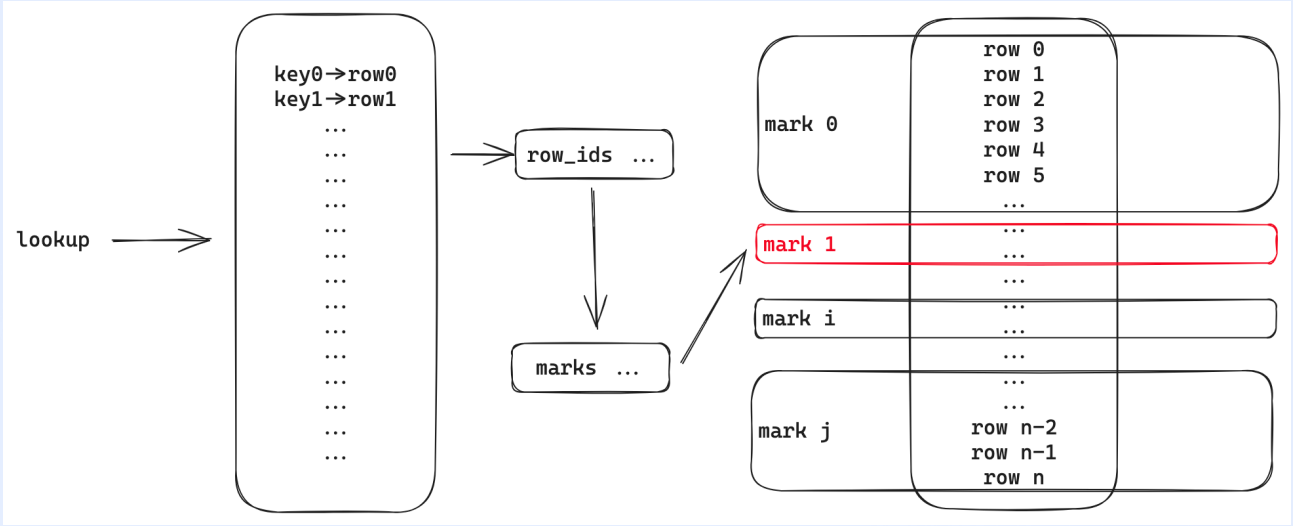


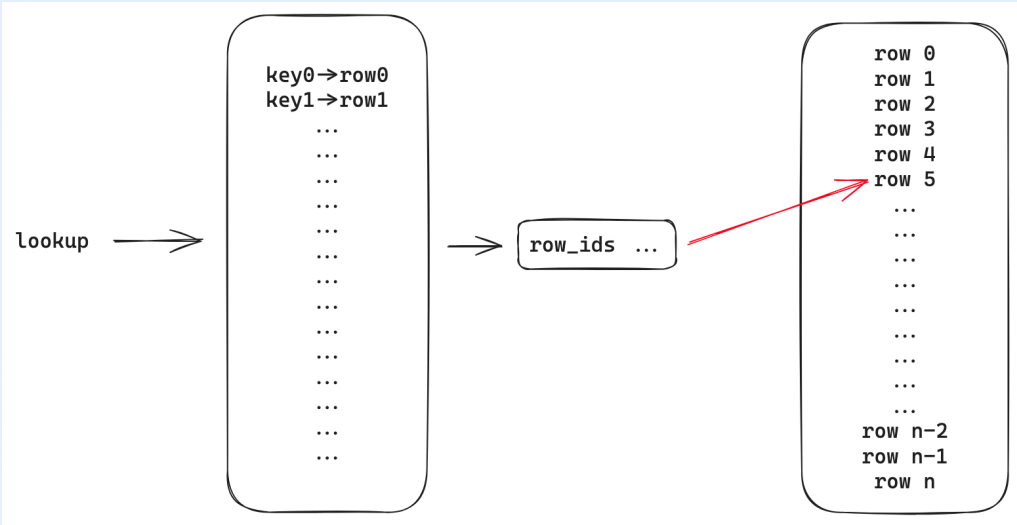
性能优化—高并发

核心优化点

基于索引快速定位数据范围

- 通过点查索引得到行号，找到读取的列存 mark
- 通过点查索引直接访问行存



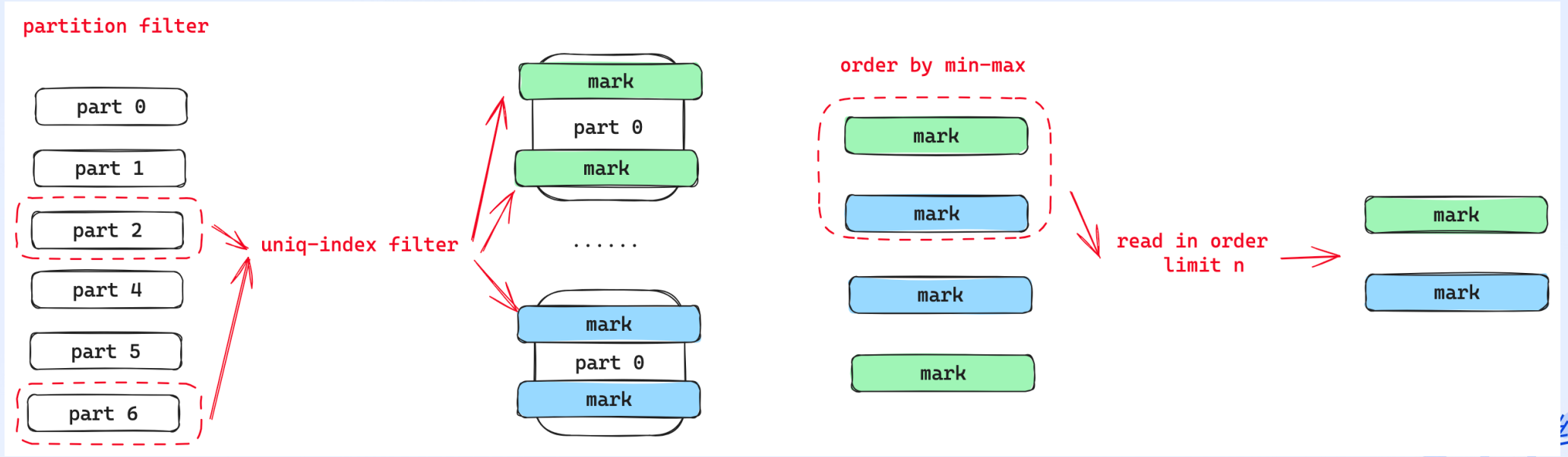


性能优化—高并发

核心优化点

高效的读链路优化

- 轻量的分区裁剪
- topn 场景的提前过滤

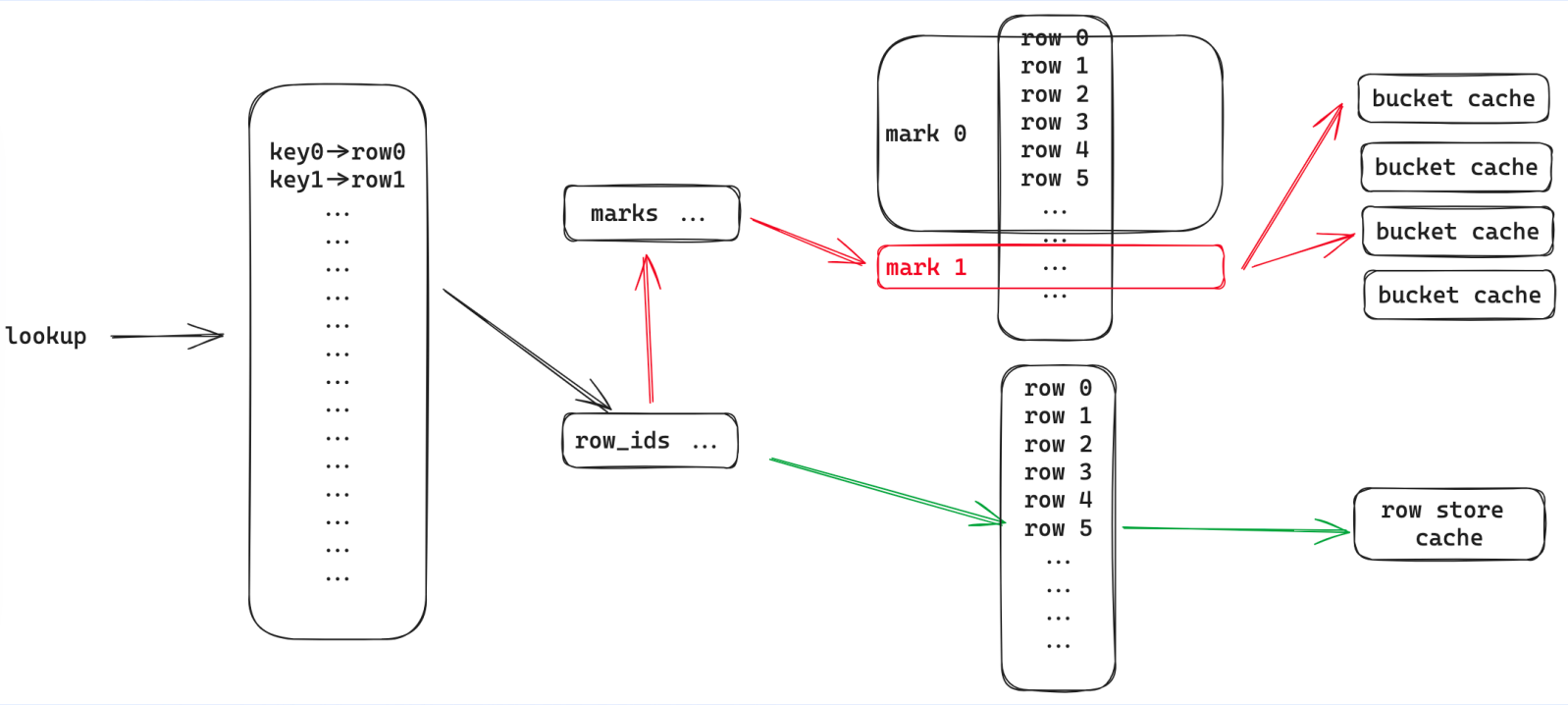


性能优化—高并发

核心优化点

高效的读链路优化

- 列存数据做分桶缓存
- 行存数据做行级别缓存



性能优化—高并发

核心优化点

Prepared Statement

- 优化parser 解析的时间
- 优化queryplan 生成的时间

```
CREATE [PERMANENT] PREPARED STATEMENT [IF NOT EXISTS | OR REPLACE] name [ON CLUSTER cluster_name]
    AS
select query with prepared_parameters;
```

```
CREATE PREPARED STATEMENT prep1 AS
SELECT count()
FROM (SELECT number FROM system.numbers LIMIT 10)
WHERE number < [i: UInt32];
```

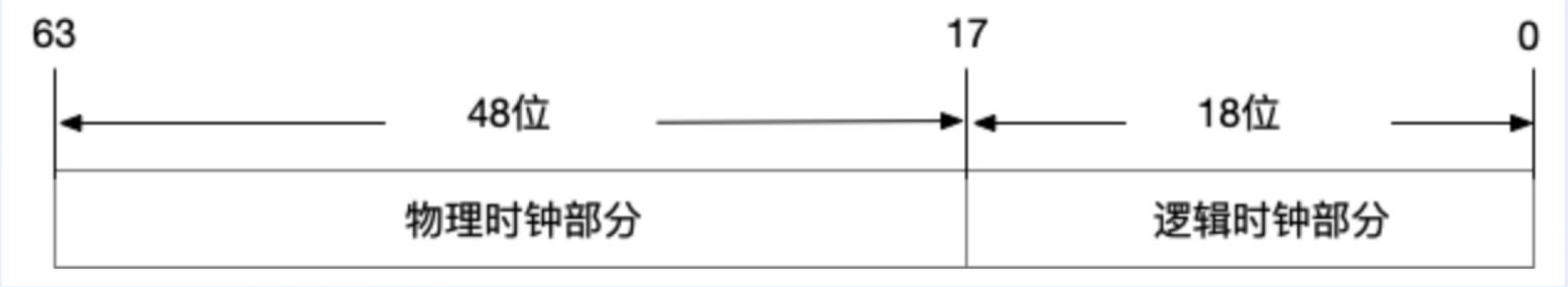
```
EXECUTE PREPARED STATEMENT prep1 USING i = 1;
```

性能优化—事务

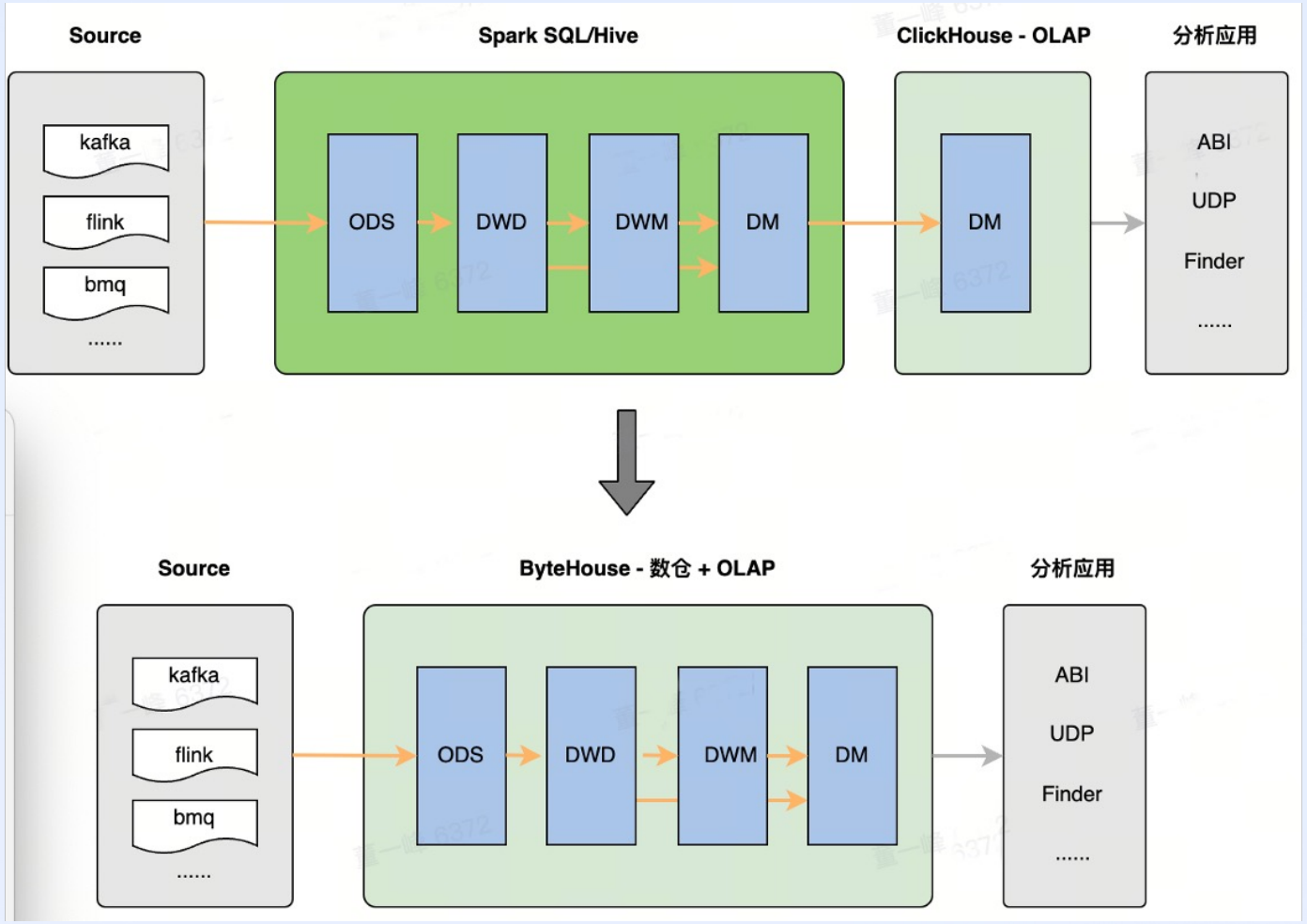
- 隐式和显式事务
- Read Committed 隔离级别，写不阻塞读
- 两阶段提交实现，支持海量数据的原子写入
- 具备灵活可控的并发控制的功能

性能优化—事务

- 中心授时服务 TSO (TimeStamp Oracle)
- Timestamp ordering
- 创建事务：为事务分配开始时间t_s
- 提交事务：为事务分配提交时间t_c
- 可见性判断：对t_s为TS的事务，能读到所有已提交且t_c < TS的事务数据



最新功能—ETL


Source
Spark SQL/Hive
ClickHouse - OLAP
分析应用
kafka
flink
bmq
.....
ODS
DWD
DWM
DM
DM
ABI
UDP
Finder
.....
Source
ByteHouse - 数仓 + OLAP
分析应用
kafka
flink
bmq
.....
ODS
DWD
DWM
DM
ABI
UDP
Finder
.....

最新功能—ETL

- 简化架构，减少冗余和复杂性(一体化方案，减少多模块、多存储格式)
- 提高稳定性，任务级容错(BSP模式，通过Task级别容错)
- 优化资源利用(增加并行度和智能调度，优化资源利用)
- 提高实时分析和流处理能力（实时数据的快速加载和转换）

BSP 执行、失败重试、查询队列、算子下推、Autospill、异步执行、智能调度、ELT日志 等

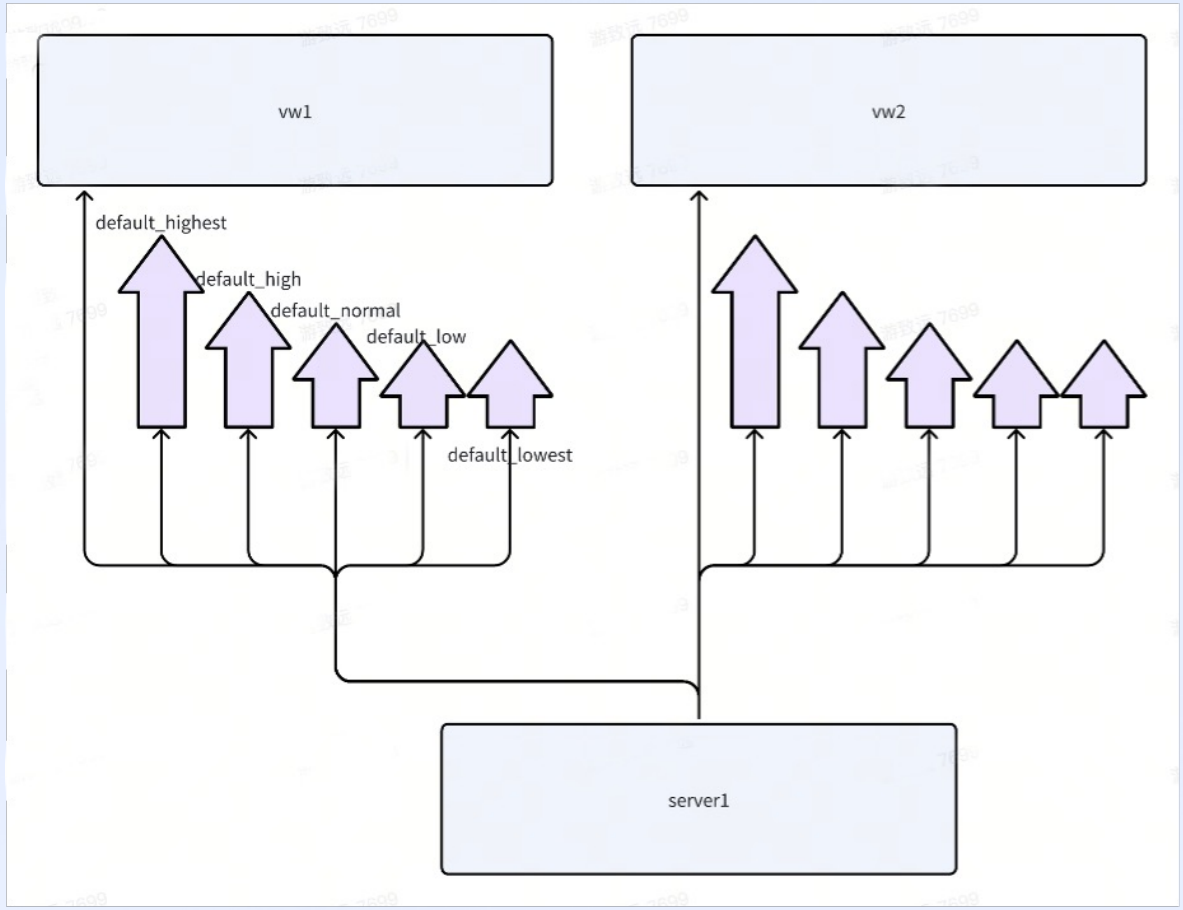
最新功能—ETL BSP

- Exchange 数据落盘
- Stage By Stage 调度执行
- Stage 实例失败重试
- Stage 实例并发优化

bsp模式的优势

- 通过增加并行度，能够显著降低内存及cpu的峰值；
- task failover: 在查询中发生错误时（特别是总内存超限制），能够原地重试，减少用户感知和手动重试成本。

最新功能—计算组队列




```
alter warehouse vw_default add rule set tables =
['test.orders'] , rule_name = 'table_rule_name1'
where queue_name = 'default_high';

alter warehouse vw_default add rule set query_id =
'.*2.*' , rule_name = 'query_id_rule1', user =
'default' where queue_name = 'default_high';

alter warehouse vw_default modify rule set
max_concurrency = 10 , query_queue_size = 20 where
queue_name = 'default_high'
```

构建多任务队列，对应不同优先级查询

最新功能—异步物化视图

```
CREATE MATERIALIZED VIEW customer_order_mv
REFRESH ASYNC START('2024-01-01 10:00:00') EVERY (INTERVAL 1 DAY)
AS SELECT
    order_list.customer_id,
    sum(goods.price * order_list.quantity) as total_spent
FROM order_list
INNER JOIN goods ON goods.item_id = order_list.item_id
GROUP BY order_list.customer_id;
```

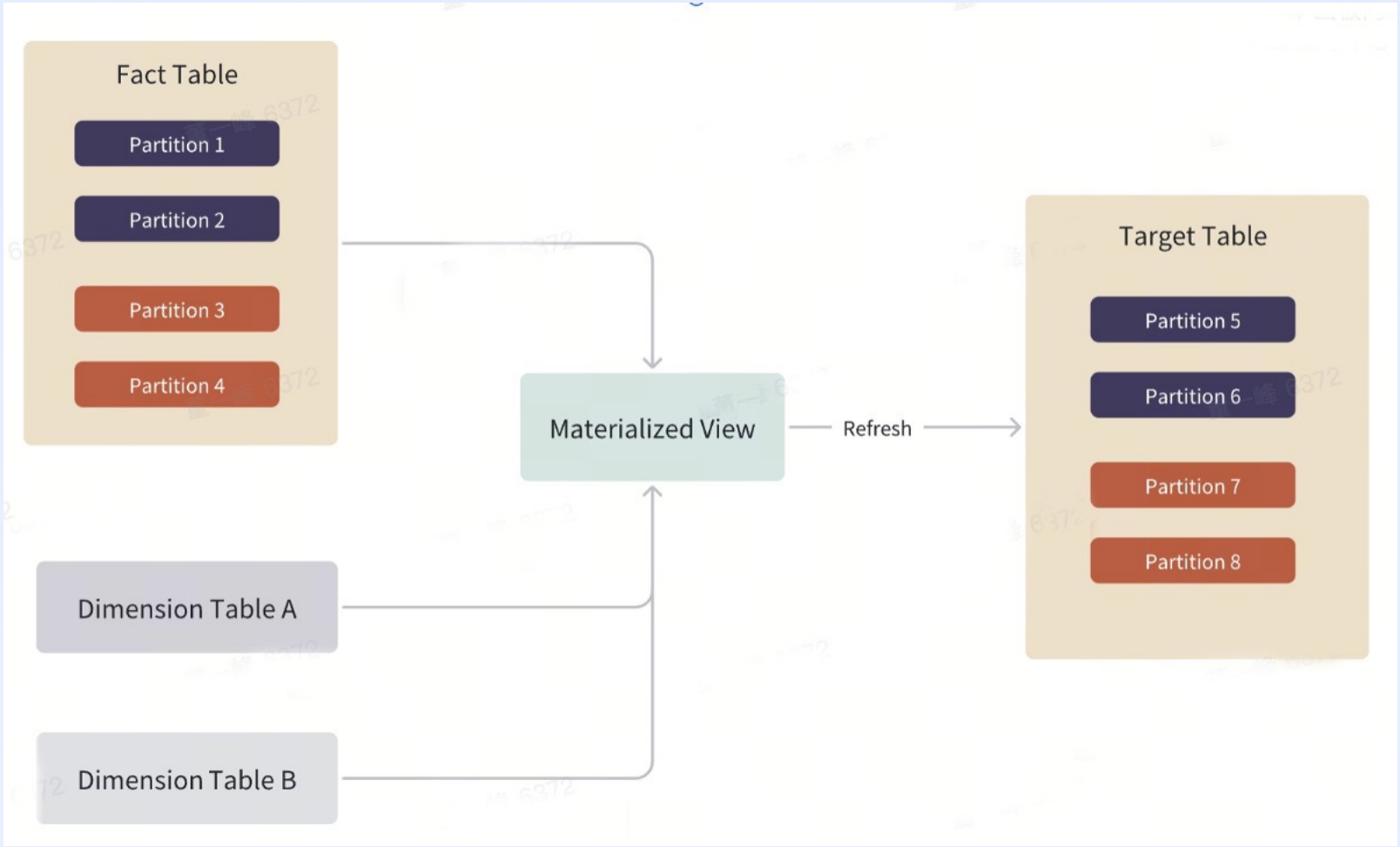
```
SYSTEM [START VIEW | STOP VIEW] [database.]<mv_name>;

ALTER TABLE [database.]<mv_name> MODIFY REFRESH EVERY INTERVAL ... ;

REFRESH MATERIALIZED VIEW [database.]<mv_name>;
```

物化视图加速查询执行

最新功能—异步物化视图



多表物化视图异步刷新原因

最新功能—向量检索

- 建表与索引定义

```
CREATE TABLE test_ann
(
    `id` UInt64,
    `label` String,
    `time` DateTime,
    `vector` Array(Float32),
    INDEX v1 vector TYPE HNSW('DIM=960, METRIC=COSINE'),
    CONSTRAINT cons_vec_len CHECK length(vector) = 960
)
ENGINE = MergeTree
ORDER BY time
PARTITION BY toYYYYMMDD(time)
SETTINGS
index_granularity = 128
```

- 查询：基本查询

```
select
    id,
    dist
from test_ann
order by cosineDistance(vector, [query_vector]) as dist
limit 100
settings hnsw_ef_s=200
```

- 查询：混合查询

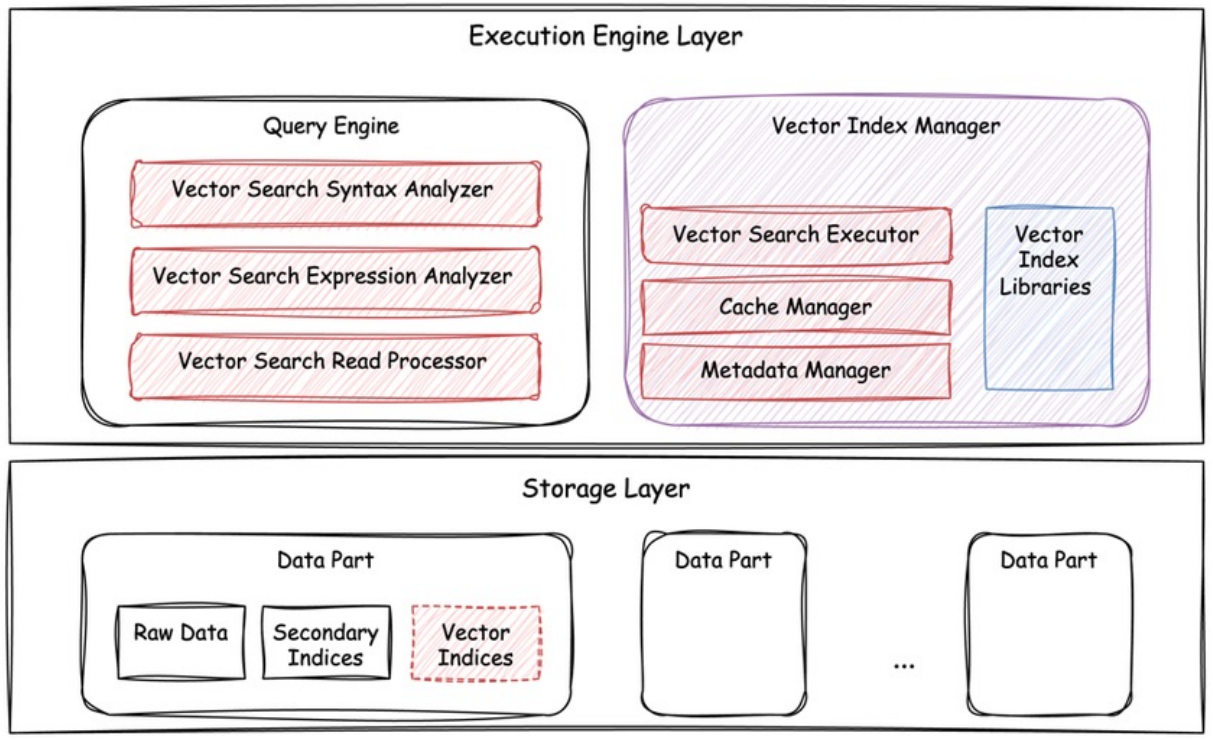
```
select
    id,
    dist
from test_ann
where label = 'coat'
    and time >= '2023-11-22 11:00:00'
    and time <= '2023-11-23 18:00:00'
order by cosineDistance(vector, [query_vector]) as dist
limit 100
settings hnsw_ef_s=200
```

- 查询：Distance Range Filter

```
select
    id,
    dist
from test_ann
where dist < 0.3
order by cosineDistance(vector, [query_vector]) as dist
limit 100
settings hnsw_ef_s=200
```

最新功能—向量检索

主要组件



ByteHouse 支持了80%的字节数据分析应用

节点总数	最大集群
18,000	2,400

数据量
700PB

截至 2022 年 2 月底

- 行为分析，用户分群
- 推荐业务
- A/B测试对比查询
- 敏捷BI，数据可视化
- 智能归因分析
  波动分析
- 指标计算、比对
- APP使用分析
  运维分析
- 微服务日志搜索

谢 谢 观 看

THANKS

进入ByteHouse官方交流群，获取更多技术干货、产品资讯