云数据库ClickHouse分析业务最佳实践

ClickHouse表结构设计、关联查询优化

演讲人

阿里云 OLAP产品部 仁劼

2020/08/20

ClickHouse 表结构设计

- MergeTree 原理
- Update、Delete处理
- 建表优化

ClickHouse 表结构设计

MergeTree 原理

阿里云

订单业务分析 Case：

MYSQL 数据源

```
CREATE TABLE order_info
(
    `oid`   VARCHAR, --订单ID
    `buyer_nick`    VARCHAR, --顾客ID
    `seller_nick`   VARCHAR, --售货员ID
    `payment`   Integer, --订单金额
    `order_status`  VARCHAR, --订单状态
    ....
    `gmt_order_create`  DateTime, --下单时间
    `gmt_order_pay` DateTime, --付款时间
    `gmt_update_time` DateTime –记录边更实际
    PRIMARY KEY ( `oid` )
);
```

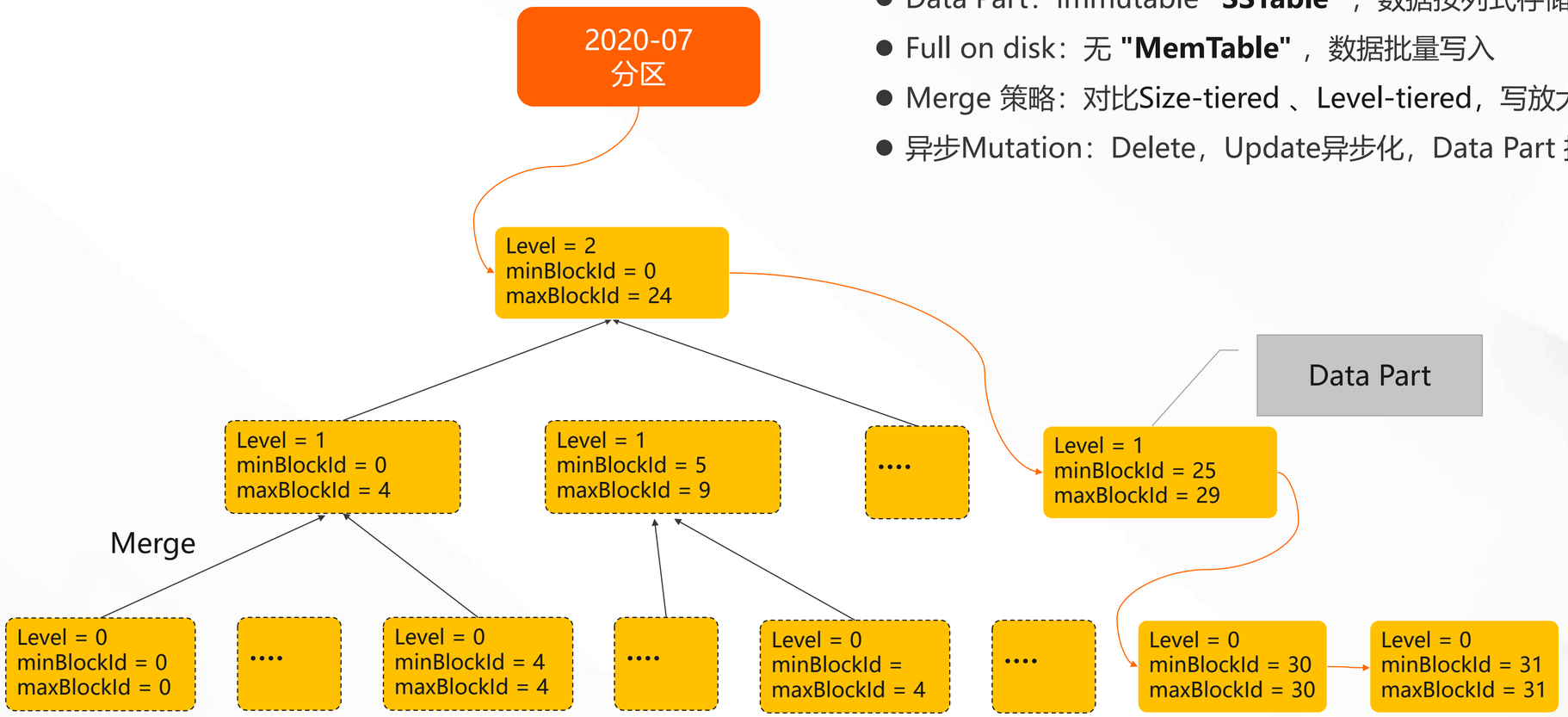
ClickHouse 数据表

```
CREATE TABLE order_info
(
    `oid`   String, --订单ID
    `buyer_nick`    String, --顾客ID
    `seller_nick`   String, --售货员ID
    `payment`   Int16, --订单金额
    `order_status`  String, --订单状态
    ....
    `gmt_order_create`  DateTime, --下单时间
    `gmt_order_pay` DateTime, --付款时间
    `gmt_update_time` DateTime –记录边更实际
)
ENGINE = ReplacingMergeTree(gmt_update_time)
PARTITION BY toYYYYMM(gmt_order_create) -- 以月为单位分区
ORDER BY (seller_nick, gmt_order_create, oid) --订单主键
PRIMARY KEY (seller_nick, gmt_order_create) --正向索引列
SETTINGS index_granularity = 8192;
```

ClickHouse 表结构设计

MergeTree 结构

- Data Part：immutable **"SSTable"**，数据按列式存储，有序（order by key）
- Full on disk：无 **"MemTable"**，数据批量写入
- Merge 策略：对比Size-tiered 、Level-tiered，写放大表现好，主键读放大表现差
- 异步Mutation：Delete，Update异步化，Data Part 按写入顺序组织

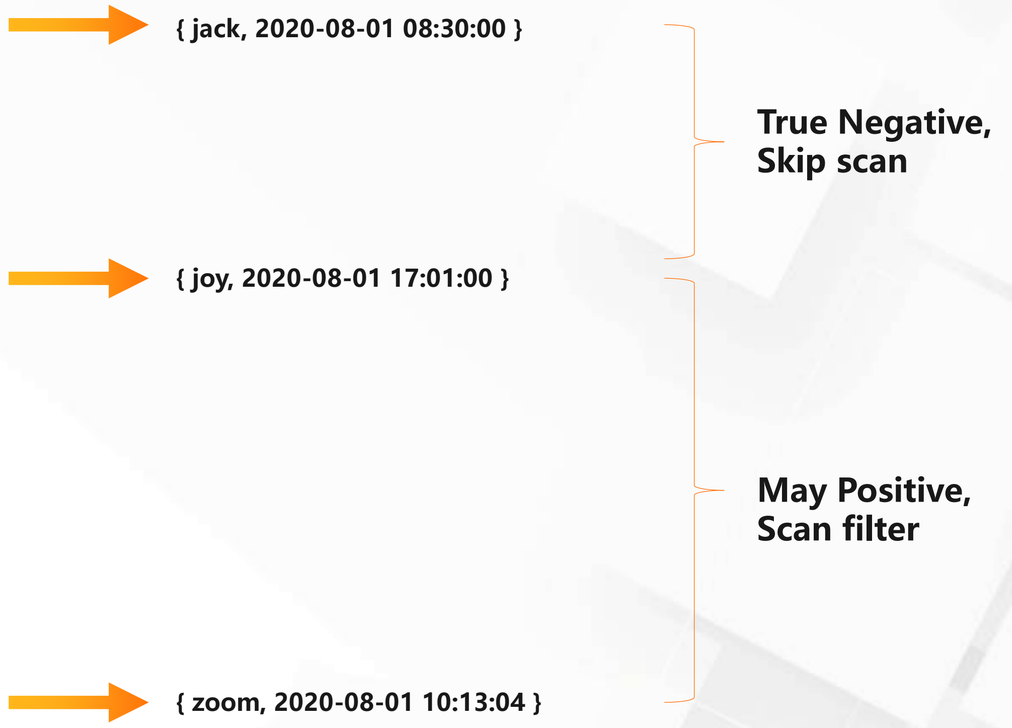


ClickHouse 表结构设计

Data Part 列存

```
select count(*) from order_info
where seller_nick = 'pick'
and gmt_order_create > '2020-08-02 00:00:00'
and gmt_order_create < '2020-08-03 00:00:00';
```

seller_nick	gmt_order_create	oid	buyer_nick	payment	...	行号
...	...	...	...	...		
jack	2020-08-01 08:30:00	94853	rock	13.3$		81920
jack	2020-08-01 09:01:00	98422	tiny	24.9$		81921
...	...					
joy	2020-08-01 17:01:00	68395	ray	45.9$		90112
...	...	...	...	...		
sonny	2020-08-01 08:23:00	89183	jay	95.0$		90592
sonny	2020-08-01 10:13:04	94734	juggle	7.4$		90593
...	...	...	...	...		
zoom	2020-08-01 10:13:04	136295	nick	6.5$		98304



磁盘列存文件

内存排序索引：正向，粗糙

ClickHouse 表结构设计

Update、Delete 处理

ClickHouse Delete + Update能力：

- 异步线程处理
- 写放大严重，引入额外IO

业务中的Mutation场景：

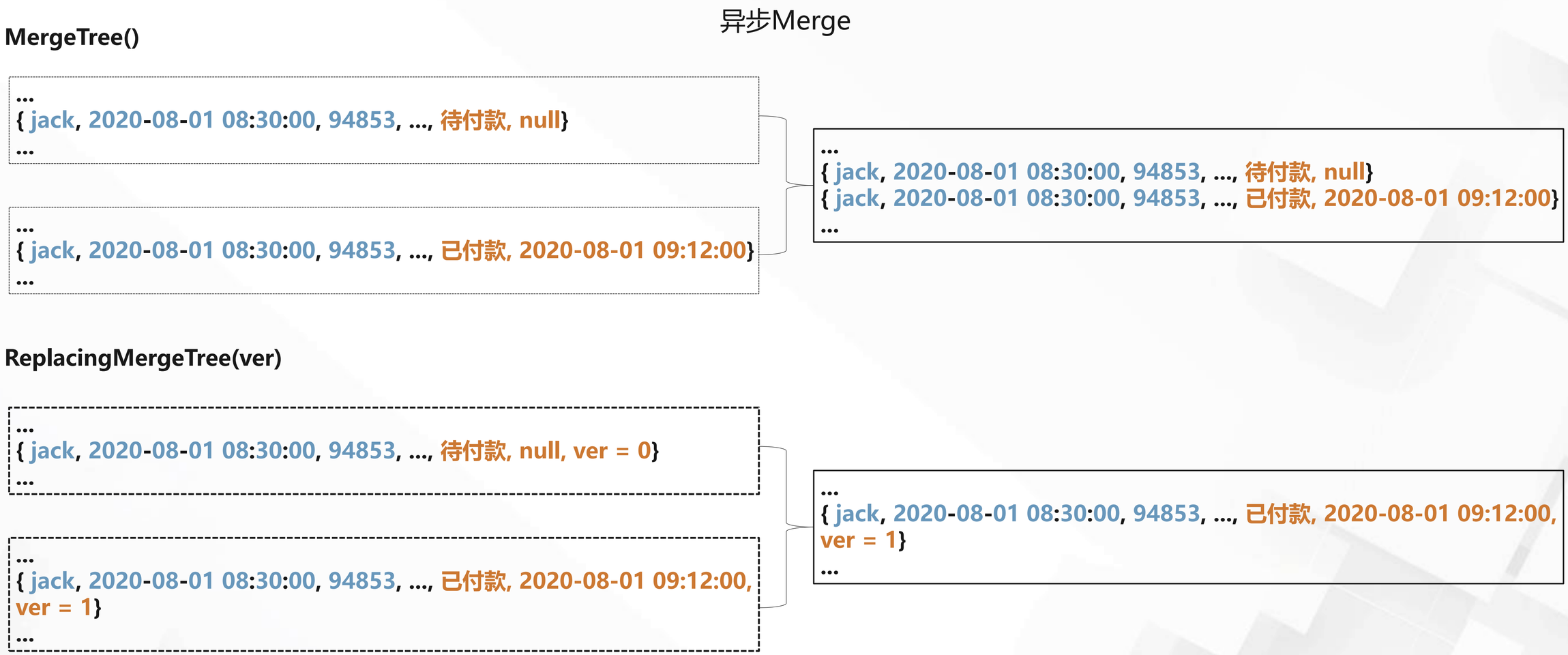
Primary Update 、Delete：主键条件单条记录更新删除，高频

Secondary Update、Delete：非主键条件数据批量订正，低频

```
--下单
insert into order_info(oid, buyer_nick, seller_nick, payment, order_status, gmt_order_create) values(....);

--付款
update order_info set order_status = '', gmt_order_pay = now() where oid = '';
```

ClickHouse 表结构设计

MergeTree Family



ClickHouse 表结构设计

MergeTree Family

异步Merge

VersionedCollapsingMergeTree(sign, ver)

```
...
{ jack, 2020-08-01 08:30:00, 94853, ..., 待付款, null, ver = 0, sign = +1}
...
```

```
...
{ jack, 2020-08-01 08:30:00, 94853, ..., 待付款, null, ver = 0, sign = -1}
{ jack, 2020-08-01 08:30:00, 94853, ..., 已付款, 2020-08-01 09:12:00,
ver = 1, sign = +1}
...
```

```
...
{ jack, 2020-08-01 08:30:00, 94853, ..., 已付款, 2020-08-01 09:12:00 ,
ver = 1, sign = +1}
...
```

AggregatingMergeTree()

```
`order_status`  SimpleAggregateFunction(anyLast, String),
`gmt_order_pay`   SimpleAggregateFunction(anyLast, Nullable(DateTime))
```

minBlock = 1024,
maxBlock = 2093,

```
...
{ jack, 2020-08-01 08:30:00, 94853, ..., 待付款, null}
...
```

minBlock = 2094,
maxBlock = 3242,

```
...
{ jack, 2020-08-01 08:30:00, 94853, ..., 已付款, 2020-
08-01 09:12:00}
...
```

```
...
{ jack, 2020-08-01 08:30:00, 94853, ..., 已付款, 2020-08-01 09:12:00}
...
```

ClickHouse 表结构设计

如何保证查询结果一致性

异步Merge不可控：

- 查询结果突然跳变

Merge策略不考虑主键合并问题：

- 查询没有准确结果

解决方案：

T+1 数据执行强制 optimize partition

T+0 数据查询时主键合并

查询时主键合并策略：

final 修饰符 （merge sort aggregation），异步merge逻辑保持一致
占用内存少，效率较低

```
select * from order_info final
where gmt_order_create > '2020-08-01
00:00:00');
```

聚合视图 (hash aggregation)，group by primary key创建主键去重视图
内存占用大，OOM风险

```
select argmax(order_status, gmt_update_time), ...
from order_info
where gmt_order_create > '2020-08-01
00:00:00'
group by seller_nick, gmt_order_create, oid;
```

ClickHouse 表结构设计

建表优化

三要素：
- 数据分区粒度
- 排序键 & 主键序列
- 索引粒度（index_granularity）

原则：
- 排序键和分区键减少重合
- 主键序列适配业务场景
- 索引粒度适配主键序列

多维度查询分析：
- 按seller_nick筛选
- 按下单时间筛选

Bad Case：
单个数据分区下的，每个seller记录太少，一个粗糙索引区间覆盖多个seller，下单时间值域不能确定

解决方案：
1）调小index_granularity，调大数据分区粒度

2）调小数据分区粒度，主键索引去掉下单时间

ClickHouse 查询优化

- MPP计算模型
- 关联查询优化

ClickHouse 查询优化

分布式MPP计算模型

```
CREATE TABLE lineorder ON CLUSTER default
(
    LO_ORDERKEY       UInt32,
    LO_LINENUMBER        UInt8,
    LO_CUSTKEY      UInt32,
    LO_PARTKEY      UInt32,
    LO_SUPPKEY      UInt32,
    LO_ORDERDATE      Date,
    ...
)
ENGINE = MergeTree
PARTITION BY toYear(LO_ORDERDATE)
ORDER BY (LO_ORDERDATE, LO_ORDERKEY);

CREATE TABLE lineorder_dist ON CLUSTER default as lineorder
ENGINE = Distributed(default, default, lineorder, LO_ORDERKEY);
```

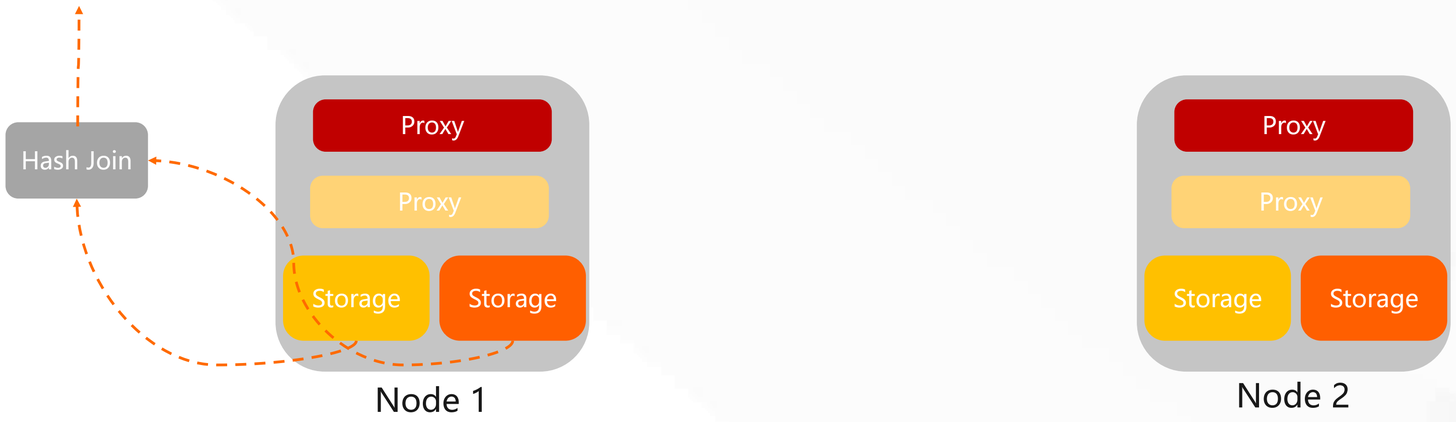
```
CREATE TABLE customer ON CLUSTER default
(
        C_CUSTKEY    UInt32,
        C_NAME        String,
        C_ADDRESS        String,
        C_CITY      LowCardinality(String),
        C_NATION      LowCardinality(String),
        C_REGION      LowCardinality(String),
        C_PHONE       String,
        C_MKTSEGMENT      LowCardinality(String)
)
ENGINE = MergeTree ORDER BY (C_CUSTKEY);

CREATE TABLE customer_dist ON CLUSTER default as customer
ENGINE = Distributed(default, default, customer, C_CUSTKEY);
```

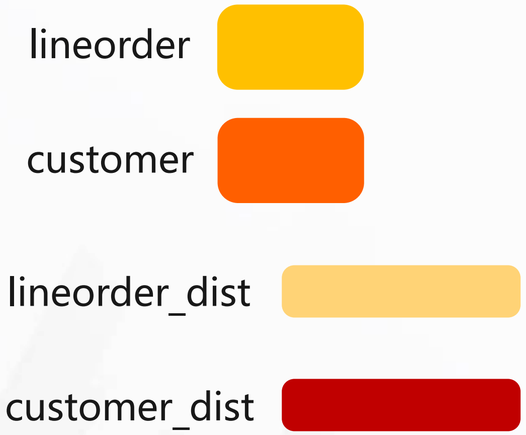
- 存储表：lineorder、customer
- Proxy 表（查询、写入分发）：lineorder_dist、customer_dist

ClickHouse 查询优化

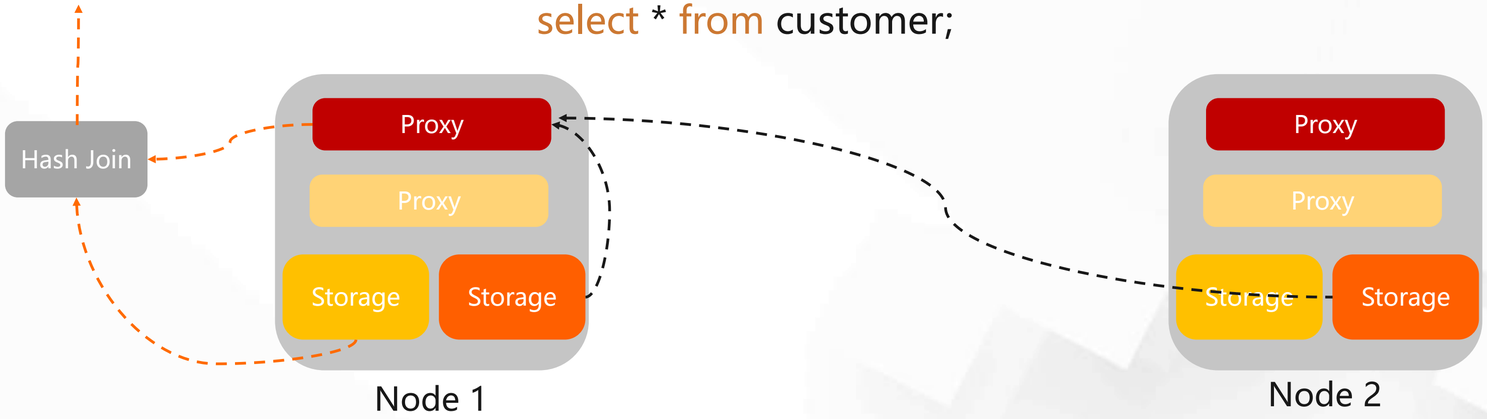
分布式MPP计算模型

```
select * from lineorder as l join customer as c on l.LO_CUSTKEY = c.C_CUSTKEY;
```



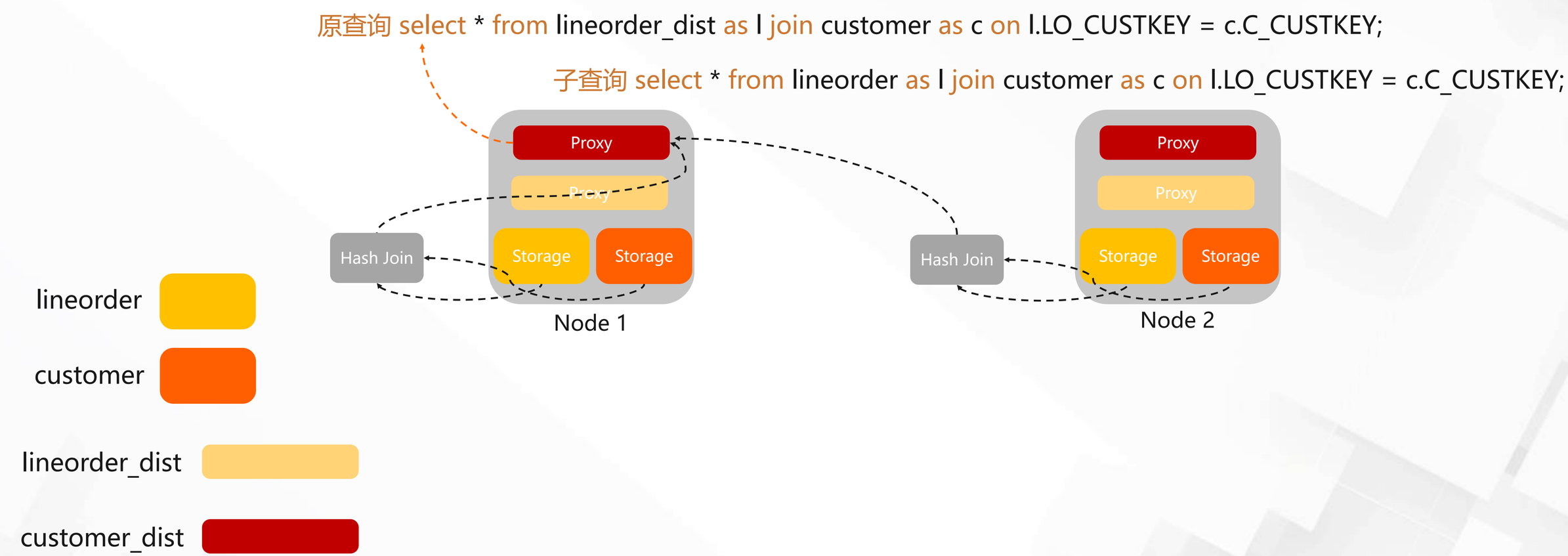



```
select * from lineorder as l join customer_dist as c on l.LO_CUSTKEY = c.C_CUSTKEY;
select * from customer;
```



ClickHouse 查询优化

分布式MPP计算模型



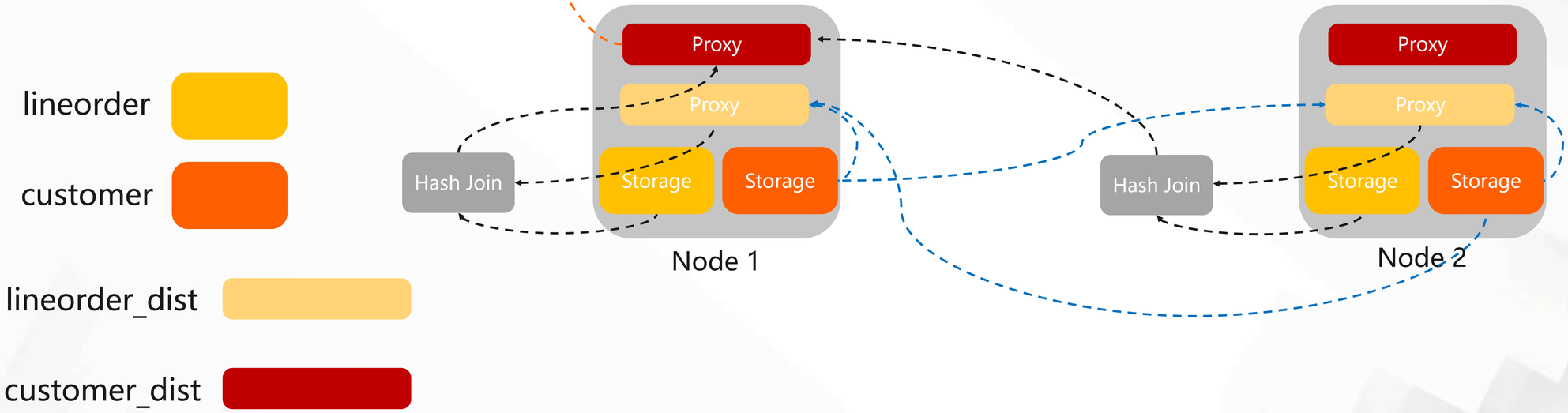
ClickHouse 查询优化

分布式MPP计算模型

原查询 select * from lineorder_dist as l join customer_dist as c on l.LO_CUSTKEY = c.C_CUSTKEY;

子查询1 select * from lineorder as l join customer_dist as c on l.LO_CUSTKEY = c.C_CUSTKEY;

子查询2 select * from customer;



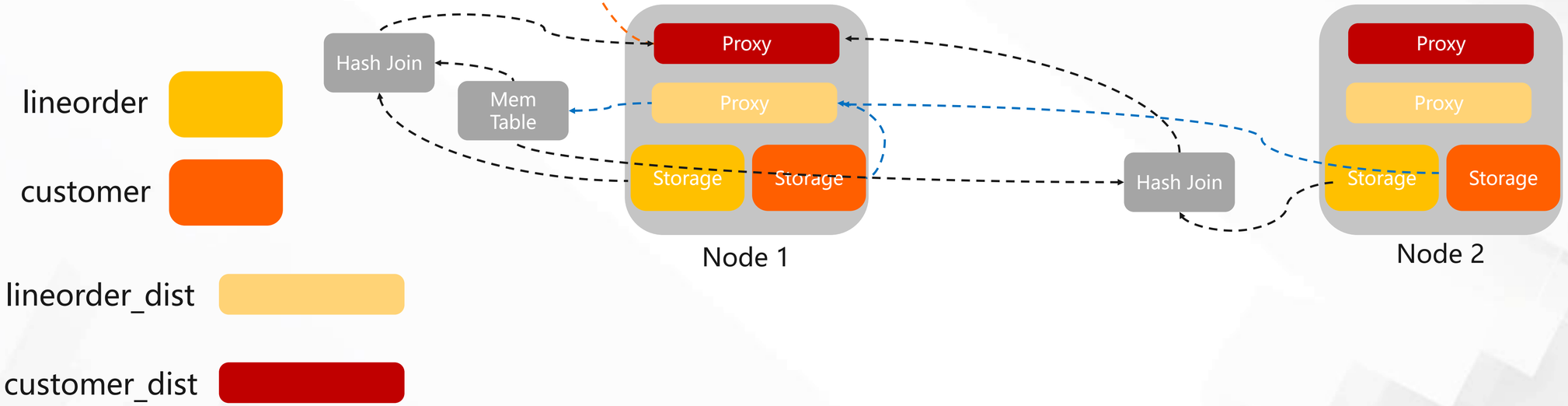
ClickHouse 查询优化

分布式MPP计算模型

原查询 select * from lineorder_dist as l global join customer_dist as c on l.LO_CUSTKEY = c.C_CUSTKEY;

子查询1 select * from lineorder as l join c on l.LO_CUSTKEY = c.C_CUSTKEY;

子查询2 select * from customer;



ClickHouse 查询优化

分布式Join 优化

Join 执行规则：

- 从最后一个Join递归改写子查询，每个子查询只处理一个Join
- 左表是Proxy表时才可以分布式执行Join
- Join Order不调整
- Global Join构建内存临时表

Join 优化原则：

- set distributed_product_mode = allow
- 多表Join改写成嵌套子查询
- 分区对齐，使用local Join
- 字典表，取消Join 操作

Q&A