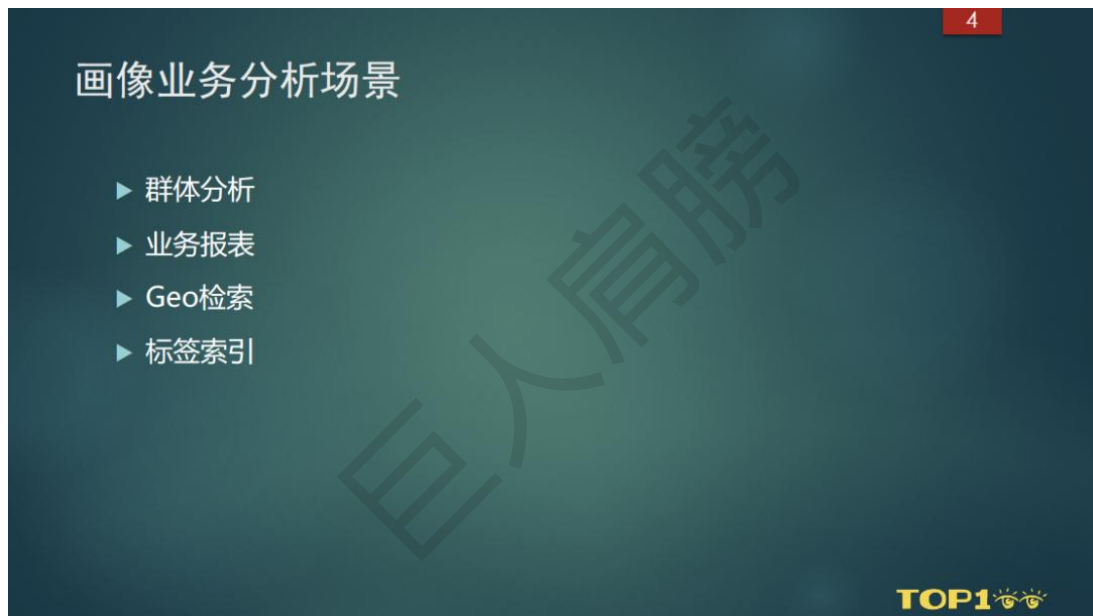


## 1.1 画像业务分析场景

百度的用户画像是面向百度全产品线的基础数据和服务平台。

我们会为包括百度凤巢、搜索、Feed 等百度各条产品线提供服务，每天有着千亿级的离/在线的数据调用规模。

随着业务的发展，我们逐步形成了用户理解全流程的数据和服务，覆盖从多元数据的采集、大规模的数据挖掘、高性能的数据服务以及面向业务场景的解决方案。



在我们的工作中，**用户群体分析**和**人群圈选**是比较常见的需求，包括群体分析、业务报表、Geo 检索、圈选等具体的场景。由于整个数据的规模比较大，同时业务对于时效性的要求比较高，在不少的业务中我们采用基于 Doris 的方法来支持和实现。群体分析大家可能会比较熟悉，也就是对一个给定人群做人群报告和多维分析。业务报表也是常见的 OLAP 场景，对很多业务数据进行统计计算，Doris 的基础能力能够很好的满足这些业务场景。

Geo 检索大家可能不是那么熟悉，Doris 在 2019 年前后集成了 Google Geo 的索引，使得对于基于地理位置的数据能够快速地进行查询、聚合操作。在一部分对于区域位置相关检索的领域，Doris 能够发挥它分布式的优势，达到不错的应用效果。

我们自己的应用测试是通过构建覆盖全北京的 400 万细粒度多层网格数据，去完成一个具体的随机的  $0.1\text{km}^2$  以上的区域的群体计算。Doris 处理 1 平方公里区域的群体数据的耗时在 1.5 秒左右，这对于很多的区域报表，像城市大屏、城市综合管理等场景，它是有非常大的应用空间的。

## 1.2 超大数据规模下的标签索引问题

基于用户标签的索引，是各大互联网公司应用最广泛的业务，也是我们今天重点讨论的一个场景。

5

### 超大数据规模下的标签索引问题

- ▶ 用户画像数据沉淀，支撑应用
- ▶ 基于标签索引的人群定向在多个关键行业、场景有广泛应用
- ▶ 计算效率对于人群圈选应用至关重要

TOP1

无论是基础数据团队还是业务团队都会通过标签 tag 的挖掘来更好表达对用户的理解。

我们画像团队系统地构建了用户标签体系，但我们的数据规模相对来说更大一些。

**这里有几个原因：**一个是整个产品线覆盖和流量规模，此外还有一些特殊情况，比如我们的 id 规模远大于自然人的 id 规模，是一个数百亿级别的数据。

另外一个方面，我们从数据挖掘的层面建立了一个比较强、比较全面的画像标签体系，它的整个规模会比较大，人均的标签覆盖比较广。它的好处是可以灵活支撑应用，但是问题是在应用过程中会产生一些规模上的问题。

基于这些标签，可以条件筛选的去构建人群，进而在用户推荐、广告定向、消息推送、用户增长等领域应用。

一般来说这种业务有两个特点，一个是客户对标签的选择范围非常广，**条件组合很复杂，业务灵活度非常高**；另一个是对计算效率，特别是对于人群圈选的数量，人群分析计算的**时效要求非常高**。计算越快，使用越灵活，越能够帮助客户找到他的目标人群。

## 2 技术问题、思路

### 2.1 早期基于离线计算的方法

早期我们采用的是基于离线的计算方法，也就是用 MapReduce 来解决问题。这个方法的问题非常显而易见，灵活性差、计算成本非常高、时效是业务团队几乎不能忍受的，早期基本上是天级，最少也是小时级才能产出结果。

### 2.2 技术问题

我们对问题进行了一个简单的分析，问题的核心还是前面提到的——被计算的数据规模。

## 技术问题

### ► 影响计算复杂度的几个关键问题

#### ► IO规模

- 百亿级数据规模
- 千级别标签、百万+ tag
- 平均数百标签

#### ► 计算

- 对全量数据进行规则过滤

TOP1 

我们目前是 300 亿——600 亿的 IO 规模，标签数量 2000+，由于标签下面还可能会有枚举值，所以最终会有大概 300 万左右的 tag。对全部数据进行一个扫表操作就要花非常多的时间。

另一个核心问题是，我们之前的业务场景过于复杂，为了配合业务场景在选择技术方案是做了很多的让步——更多的考虑满足业务需求而非性能。所以早期在计算逻辑层面没有处理的特别好，整个计算效率是比较低的。

但是这一类需求在我们的业务当中却展现的尤为强烈，我们非常迫切的想解决性能和功能的问题。所以在调研了非常多类似业务的方案后，**我们提出了一种高性能标签索引的解决思路**，并且考虑开发一套专用的系统来实现和解决类似的问题。

## 2.3 技术思路

## 技术思路

- ▶ 倒排索引解决IO规模问题
  - ▶ 逻辑转换
    - ▶ 标签 => 二值TAG
    - ▶ 条件 => 交并集运算
  - ▶ 存储优化 Bitmap
- ▶ 分布式计算加速计算过程
  - ▶ tag的并行
  - ▶ 分桶

### (1) 解决 IO 规模问题

原先的方法无论是基于 MapReduce 还是其他类似的逻辑，核心的问题在于我们要对全局数据进行遍历扫描。因为我们是 `uid` 为 `key` 的一个正向的数据结构，对标签进行扫描必须要扫描全量。

这里我们的解决办法是做**倒排索引**，以标签为 `key` 把 `uid` 作为 `value`。这样构建一个反向索引之后，原来我们是全表扫描，现在变成只处理关注的标签，这样整体的 IO 规模会瞬间下降好几个数量级。

其次我们要做一个计算加速的优化，这里面主要是逻辑的变化。把原来**圈选的逻辑变成交集并集的处理**。

在细节上需要做考虑两个问题：

一个是把原来标签的枚举变成二值化的 `tag`。

比如：标签球类运动，它的每一个枚举值，就需要拆解为具体的 `tag`，如篮球、足球等。基于这个转化，可以把条件圈选变成交集、并集计算的组合。例如，选

择喜欢篮球运动、不喜欢足球运动，就可以变成 tag-篮球 1，tag-足球 0 的交集。

另一个是我们有超过 300 万个 tag，在对 tag 做倒排索引时存储空间会成为一个非常大的问题，所以需要进一步降低存储，提升计算效率。

我们选择采用 Bitmap 来优化标签索引，用一个 bit 来标记一个 value，将用户作为整个 Bitmap 里的一个位，这样可以实现在存储上的空间节省。同时由于位运算在交并集上天然的优势，在计算上也能带来性能提升。

## （2）加速计算过程

在解决计算规模问题时，核心的逻辑是[用并行计算来加速过程](#)。由于数据的 tag 是 key 构建的，尤其一些基础的标签的数据覆盖率非常高，有些可以达到 90% 以上，整个 Bitmap 会非常长。

这个情况下，对于一个百亿级的 uid 范围，bitmap 的 size 将会非常大，这些 bitmap 会成为计算瓶颈，需要进一步对 bitmap 进行纵向的分桶，以加速计算，减少长尾。考虑 tag 数乘以分桶的情况，这是一个数量可观的分布式并行的储存与计算过程，对于分布式系统有着很高的要求，也是一种典型的 MPP 场景。

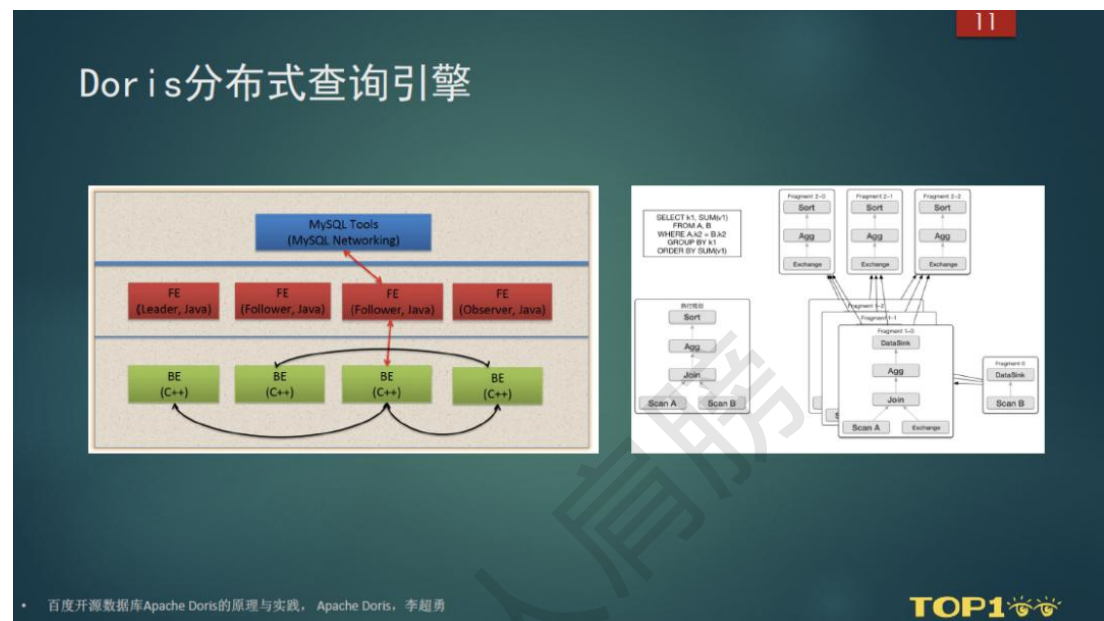
比较巧的是，在和 Doris 同学的交流过程当中，我们得知 Doris 正在做相似的工作。Doris 的 MPP 架构和正在进行的 Bitmap 集成，刚好是我们业务需要的能力。可以说是不谋而合。

我们也调研了其他开源解决方案比如说 Kylin 和 Druid，Kylin 在这个场景下有一定的局限性，它需要预计算，这就带来了维度和空间的爆炸，并且不能满足我们对细粒度数据的需求。Druid 在这方面可以满足我们的需求，但是在一些特定使用场景下我们是依赖 Doris 的，[所以我们最终选择了 Doris](#)。

### 3 基于 Doris 的技术实现

#### 3.1 Doris 分布式查询引擎

结合业务场景，我们将方案调整为基于 Doris 实现全套的标签索引服务。

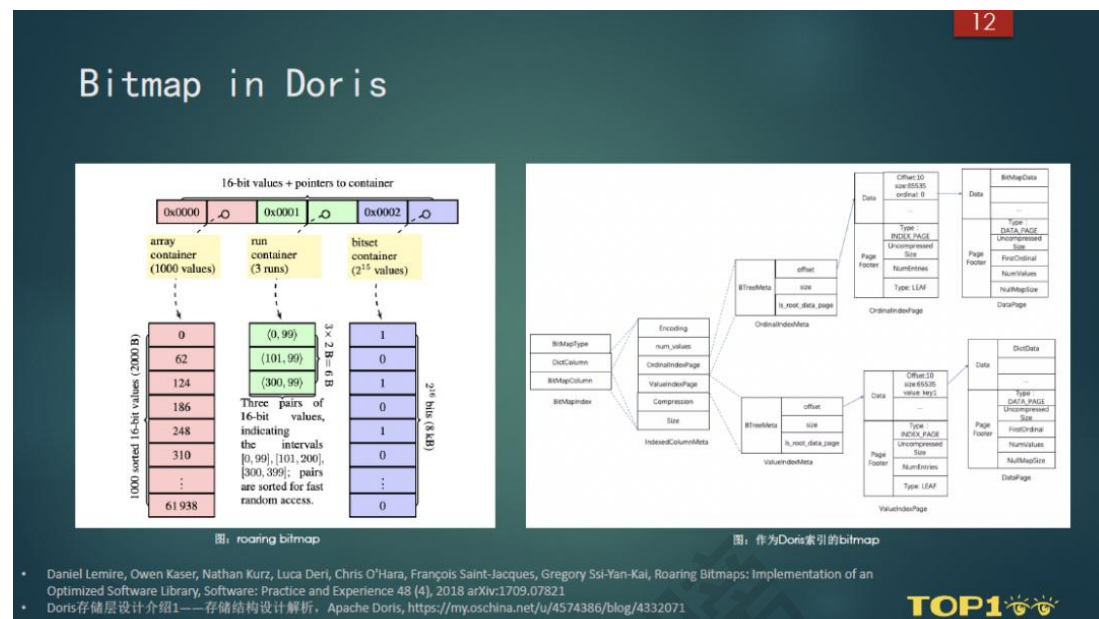


Doris 采用的 MPP 架构非常简洁，由 FE 和 BE 完成从服务接入到数据存储、管理、计算的分布式服务。其中，FE 负责存储以及维护集群数据、接收、解析、查询、设计规划整体查询流程，BE 负责数据存储和具体的实施过程。FE 会根据用户的查询去生成一个完整的逻辑规划，进一步构建分布式的逻辑发给整个集群去执行。

在右边规划图中，由一个 BE 去执行的时候，需要通过 RPC 进行数据交换，不同的计算方法和内容，交换的数据也不同。

#### 3.2 Bitmap 在 Doris 的应用

Bitmap 通常作为 OLAP 系统和存储系统的索引, Doris 很早就集成了 Bitmap 来加速数据查询的过程。



如上图所示, 右边是 Doris Bitmap 索引的数据结构。



和索引应用不同, Bitmap 作为数据应用, 可以解决明细数据的查询、交并集问题。我们在使用的時候将 Bitmap 作为数据结构, 直接使用 Bitmap 作为实际存储数据来解决明细数据查询和交并集计算的问题。

## Doris based solution

- ▶ 标签-->Tag二值化, uid倒排
- ▶ Tag bitmap建库, doris load
- ▶ intersect\_count / union\_count
- ▶ 分布式查询执行

TOP1 

基于 Doris, 首先我们通过离线的 MR 形式完成了标签 tag 的二值化和 uid 倒排 (这里还包括对 ID 的顺序处理)。

然后我们用 Bitmap 作为数据结构, Bitmap Union 为聚合函数, 采用 Doris Load 的方法完成了数据建库和数据 Load, 这个过程可能比较慢但很可行。

接下来, 我们需要把条件查询转变成交并集计算, 对单个节点来说, intersect\_count / union\_count 的方法 Doris 可以自动完成分布式的计算过程, 实现已经 Bitmap 化的标签数据的聚合计算。

基于以上的逻辑, 我们只用了两周就完成了测试过程。

### 3.3 标签索引应用在 Doris 基础实现的问题

以上的方案在比较小的计算规模上可以得到计算结果, 对于稍微复杂的计算场景, 就出现了一些问题。

## 标签索引应用在Doris基础实现的问题

### ► Doris原生bitmap实现的性能测试结果

| 测试类型    | 是否包含稠密 bitmap | 标签个数 | doris-base     |
|---------|---------------|------|----------------|
| 交集count | 否             | 6    | 3.76s          |
| 并集count | 否             | 6    | 1.32s          |
| 交集count | 是             | 14   | 超时报错: 299960ms |
| 并集count | 是             | 14   | 超时报错           |
| 交集count | 是             | 100  | 超时报错           |
| 并集count | 是             | 100  | 超时报错           |

测试数据: x00亿id, x000+标签 (平均标签数x00), x00w+ tag  
 测试环境: 物理机 (24core / 128G mem / 1T SSD \* 5)

### ► Doris 不支持批量id数据导出

TOP1 

我们做了一个测试, 在 300 亿的 uid, 平均标签数为 300 的情况下, 能够完成不包含稠密 Bitmap 的 tag 进行 6 个标签的交集和并集的计算, 但当我们把条件 tag 换成稠密的 Bitmap 之后, 用 14 个标签就已经超时了。(这里稠密的 Bitmap 指 tag 数据在全量用户 id 的覆盖率非常高的情况, 而这种 tag 恰恰是我们业务中最常用到的。)

除此之外还有一个功能问题, 当时 Doris 还不支持批量 ID 导出。

### 3.4 标签索引应用在 Doris 基础实现的性能问题

结合 Doris 的计算过程, 我们分析认为问题的核心主要有两点。第一点在于 Bitmap 本身实现的逻辑, 第二点在于在 Doris 中 Bitmap 分布式实现的问题。

## 标签索引应用在Doris基础实现的性能问题

### ► Doris的bitmap聚合函数逻辑

#### ► Roaring Bitmap 实现

- Roaring bitmap原生支持32位整型的集合运算，并引入了大量指令级别优化，因此压缩比和运算效率都比较高
- Roaring64Map将bitmap扩展到支持64位整型，高32位采用红黑树的方式存储，集合计算效率极差
- 画像规模场景下，uid的范围通常需要64位整型表示，计算效率不高

#### ► IO成本与单点计算压力

- 当bitmap基数较大时，如bitmap大小超过1g，网络/磁盘IO处理时间比较长
- 后端be实例在scan数据后全部传输到顶层节点进行求交和并运算，给顶层单节点带来压力，成为处理瓶颈

TOP1

一般来说，32 位 Bitmap 在压缩比和计算效率上有很好的平衡，但是 64 位的 Bitmap 的计算效率就相对较差。画像的数据在百亿级别，所以需要用 64 位的 Bitmap ，那么计算效率就比较低。

第二个，当 Bitmap 基数比较大时，数据规模也比较大，网络磁盘和网络 IO 处理时间比较长。Doris 在计算过程中需要 scan 数据后全部传输到顶层节点进行求交和求并运算，但本质上交并集计算是在单节点进行的，同时要经过网络 IO 之后才能进行处理，这都成为了影响性能的关键点。

## 3.5 性能解决方案

### (1) Bitmap 纵向切分建库

针对以上的问题，我们提出了一种正交的 Bitmap 计算的 UDAF 解决办法。

## 性能解决方案

### ► Bitmap 纵向切分建库

#### ► 逻辑

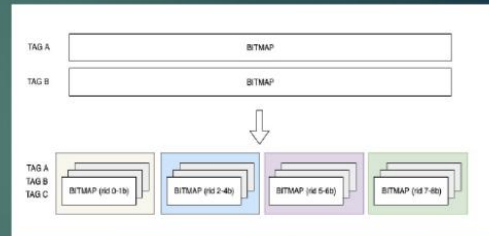
- 减少或避免高32位的计算
- 进一步提升并发能力

#### ► 解决方案

- uid维度纵向切分
- Rid 范围分片
- 降低单个bitmap的size

#### ► 场景特点

- 满足正交
- uid有序



图：bitmap纵向切分

我们对 Bitmap 进行纵向的切分建库，如右图所示，我们的出发点是减少或避免高 32 位的计算，并且进一步提升并发能力。

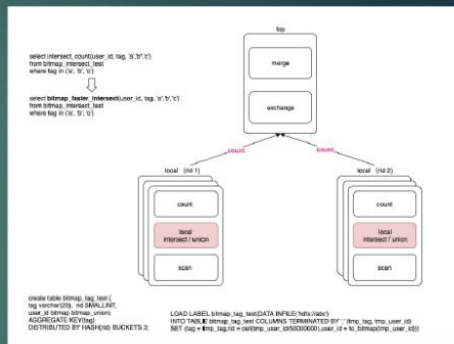
具体来讲，首先我们按照 uid 进行数据纵向切分，以 rid 范围进行分片，降低单个 Bitmap 的 size。这同时带来了一个好处，这种切分方式满足正交方式，无论在 tag 维度层面还是在 id 范围层面，同一个分片内是可以相互独立的。基于这种正交关系，我们进一步优化分布式计算的实现。

### （2）分布式计算优化

## 性能解决方案

## ► 分布式计算优化

- ▶ 相同的id范围，不同tag数据，同实例部署
- ▶ 各实例完成本地count，避免merge节点的单点计算
- ▶ 只做节点计算结果的传输，无需原始bitmap全量数据，减少网络IO



图：以count计算为例的分布式优化

**TOP1**👁👁

首先在数据层面，我们让相同 id 范围的数据（即使不同 tag）也在相同的实例部署。在数据的交并集计算上，我们让各个分片分别在各自的节点上完成计算，merge 的是计算结果而不是原始数据。这样可以让计算的节点数变多、实现并发，还可以在数据传输过程中不进行全量传输，网络通信的成本也降到最低。

右图所示是进行 count 计算时，在本地完成了 local 的交并集计算之后只需要把 count 值给出去，最后将不同的 count 值 merge 就完成了整个计算过程。

### 3.5 性能测试

经过以上的处理，我们的整体计算效率得到了很大的提升，小数据量的情况下得到了**两个数量级的提升**，在大数据量下我们**由不可能变成了可能**。

## 性能测试

| 测试类型    | 是否包含稠密bitmap | 检索标签数 | Doris basic solution | doris rid 外部并行计算 | 标签索引方案 |
|---------|--------------|-------|----------------------|------------------|--------|
| 交集count | 否            | 6     | 3.76s                | -                | 0.06s  |
| 并集count | 否            | 6     | 1.32s                | -                | 0.06s  |
| 交集count | 是            | 14    | 超时报错: 299960ms       | 5s               | 2.6s   |
| 并集count | 是            | 14    | 超时报错                 | 7s               | 2.52s  |
| 交集count | 是            | 100   | 超时报错                 | 9s               | 6.74s  |
| 并集count | 是            | 100   | 超时报错                 | 12s              | 7.60s  |

测试数据: x00亿id, x000+标签 (平均标签数x00), x00w+ tag  
测试环境: 物理机 (24core / 128G mem / 1T SSD \* 5)

TOP1

从上图中可以看出，性能变得更加可用，可以在秒级完成我们的业务需求。

### 3.6 标签索引应用在 Doris 基础实现的功能问题

## 标签索引应用在Doris基础实现的功能问题

- ▶ 数据灌库、建库
  - ▶ 问题：原生索引创建耗时冗长
  - ▶ 解决：Enhanced Spark-based load
- ▶ 批量ID导出
  - ▶ 问题：Doris sql 接口支持亿级数据导出存在困难
  - ▶ 解决：批量导出的方法，解决规模输出问题
    - ▶ select outfile : MysqlWriter --> FileWriter
    - ▶ 通过Broker把结果数据写到远端存储 ( hdfs/bos )

TOP1

#### (1) 数据灌库、建库

目前还是基于之前的离线计算方式，这种方式效率不够高，但是引入新的解决方法——Enhanced Spark-based Load，这种方式现在还在测试，预计性能会有非常大的提升。

## (2) 批量 ID 导出

Doris 自身的 SQL API 没有支持亿级数据导出。结合业务场景 Doris 的同学帮我们想出了一种解决方法——通过修改现有的 `select outfile` 将 `MysqlWriter` 改写成 `FileWriter`，并且通过 `Broker` 把结果数据写到远端存储。这样就可以实现批量原始 id 的数据导出，满足我们对细粒度用户数据的需求。

## 4 业务效果

标签索引满足主要人群圈选业务场景

人群圈选时效从天/小时级提升到秒级响应

业务应用效率大幅度提升，支持更加灵活的业务应用

在广告、增长等方向取得良好的业务效果

## 5 应用指南

我们在今年 8 月已经将相关的代码和使用指南已经提交到 Apache Doris 的代码库，供

大家分享使用：

<http://doris.incubator.apache.org/master/zh-CN/extending-doris/udf/contrib/udaf-orthogonal-bitmap-manual.html>

## 应用指南（正交的BITMAP计算UDAF）

### ► 代码合入&官方指南

- <http://doris.incubator.apache.org/master/zh-CN/extending-doris/udf/contrib/udaf-orthogonal-bitmap-manual.html#%E8%83%8C%E6%99%AF>

```
CREATE TABLE `user_tag_bitmap` (
  `tag` bigint(20) NULL COMMENT "用户标签",
  `hid` smallint(6) NULL COMMENT "分桶id",
  `user_id` bitmap BITMAP_UNION NULL COMMENT ""
) ENGINE=OLAP
  AGGREGATE KEY(`tag`, `hid`)
  COMMENT "OLAP"
  DISTRIBUTED BY HASH(`hid`) BUCKETS 3
```

```
LOAD LABEL user_tag_bitmap_test
(
  DATA INFILE('hdfs://abc')
  INTO TABLE user_tag_bitmap
  COLUMNS TERMINATED BY ','
  (tmp_tag, tmp_user_id)
  SET (
    tag = tmp_tag,
    hid = ceil(tmp_user_id/5000000),
    user_id = to_bitmap(tmp_user_id)
  )
)
```

TOP1

这里需要有一个额外的编译过程来满足 UDAF 的应用：

## 应用指南（正交BITMAP计算UDAF）

```
drop FUNCTION orthogonal_bitmap_intersect(BITMAP,BIGINT,BIGINT, ...);
CREATE AGGREGATE FUNCTION orthogonal_bitmap_intersect(BITMAP,BIGINT,BIGINT, ...) R
PROPERTIES (
  "init_fn"="_ZN9doris_udf250orthogonalBitmapFunctions21bitmap_intersect_initILNS_9Bi
  "update_fn"="_ZN9doris_udf250orthogonalBitmapFunctions23bitmap_intersect_updateILNS
  "serialize_fn"="_ZN9doris_udf250orthogonalBitmapFunctions30bitmap_intersect_and_ser
  "merge_fn"="_ZN9doris_udf250orthogonalBitmapFunctions12bitmap_unionEPNS_15FunctionC
  "finalize_fn"="_ZN9doris_udf250orthogonalBitmapFunctions16bitmap_serializeEPNS_15Fu
  "object_file"="http://ip:port/libudaf_orthogonal_bitmap.so" );
```

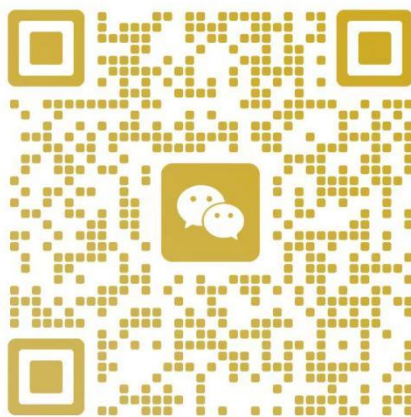
```
select BITMAP_COUNT(orthogonal_bitmap_intersect(user_id, tag, 13080800, 11110200))
from user_tag_bitmap where tag in (13080800, 11110200);
```

TOP1

最终使用的时候只需要写一个 select BITMAP\_COUNT 就可以快速得到需要的数据结果。

## 6 交流讨论

欢迎加入 Doris 分享、交流、互助大家庭：



扫一扫上面的二维码图案，加我为朋友。

欢迎访问巨人肩膀：[www.atbigapp.com](http://www.atbigapp.com)。一个致力于大数据技术和应用发展与推广，通过环境及基础能力的提供降低开发者门槛、提高开发者效率的互助互利的大数据开放平台。