

基于 Apache Doris 的货拉拉 用户画像数据模型设计与实践

高级大数据工程师 李纬航



目录

01 货拉拉画像服务背景与工程架构

02 应用层的数据模型设计与异构查询实现

03 货拉拉画像工程端的查询优化实践

04 后续规划

货拉拉介绍

国内货运开城数量

363座

月活用户数

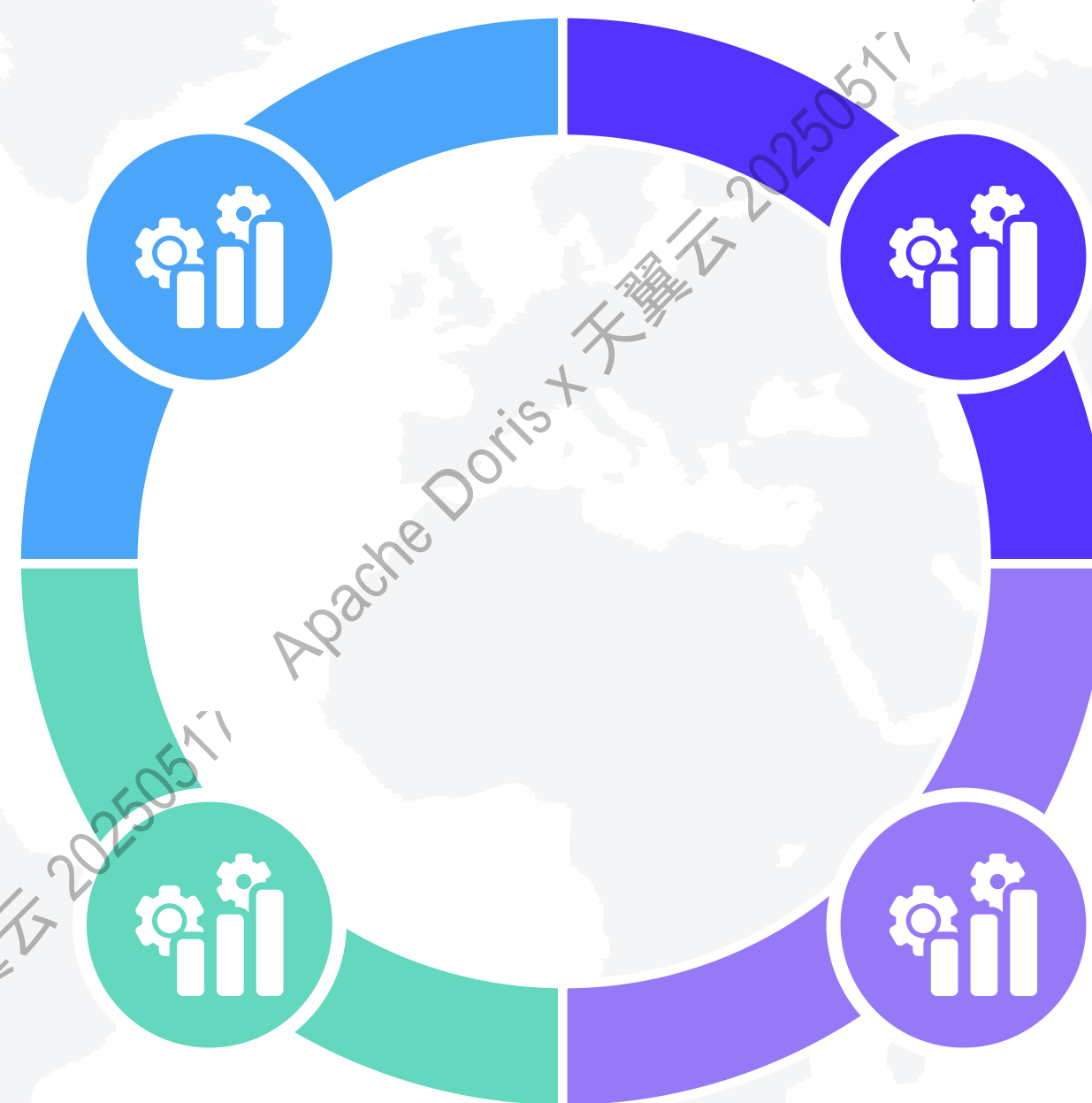
1000万+

月活司机数

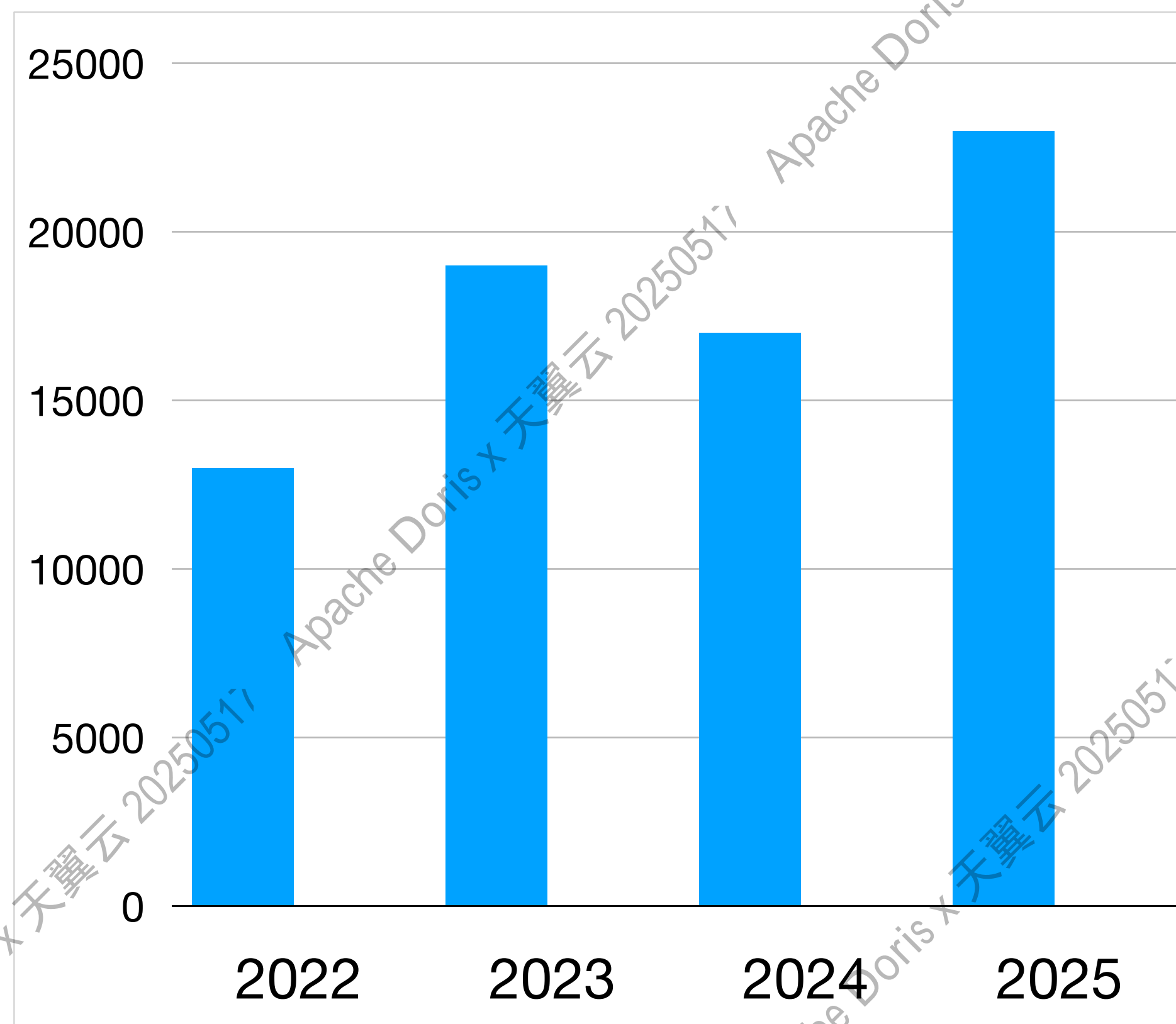
100万+

全球性覆盖

11个国家



画像平台价值 & 人群数量



画像应用场景

支撑货拉拉司机派券/邀约、用户满意度、估价等业务

画像人群增长情况

用户圈选人群的量级以及画像的标签总量逐年增加

画像平台应用规模



接入业务方数量

300+

标签数

3000+

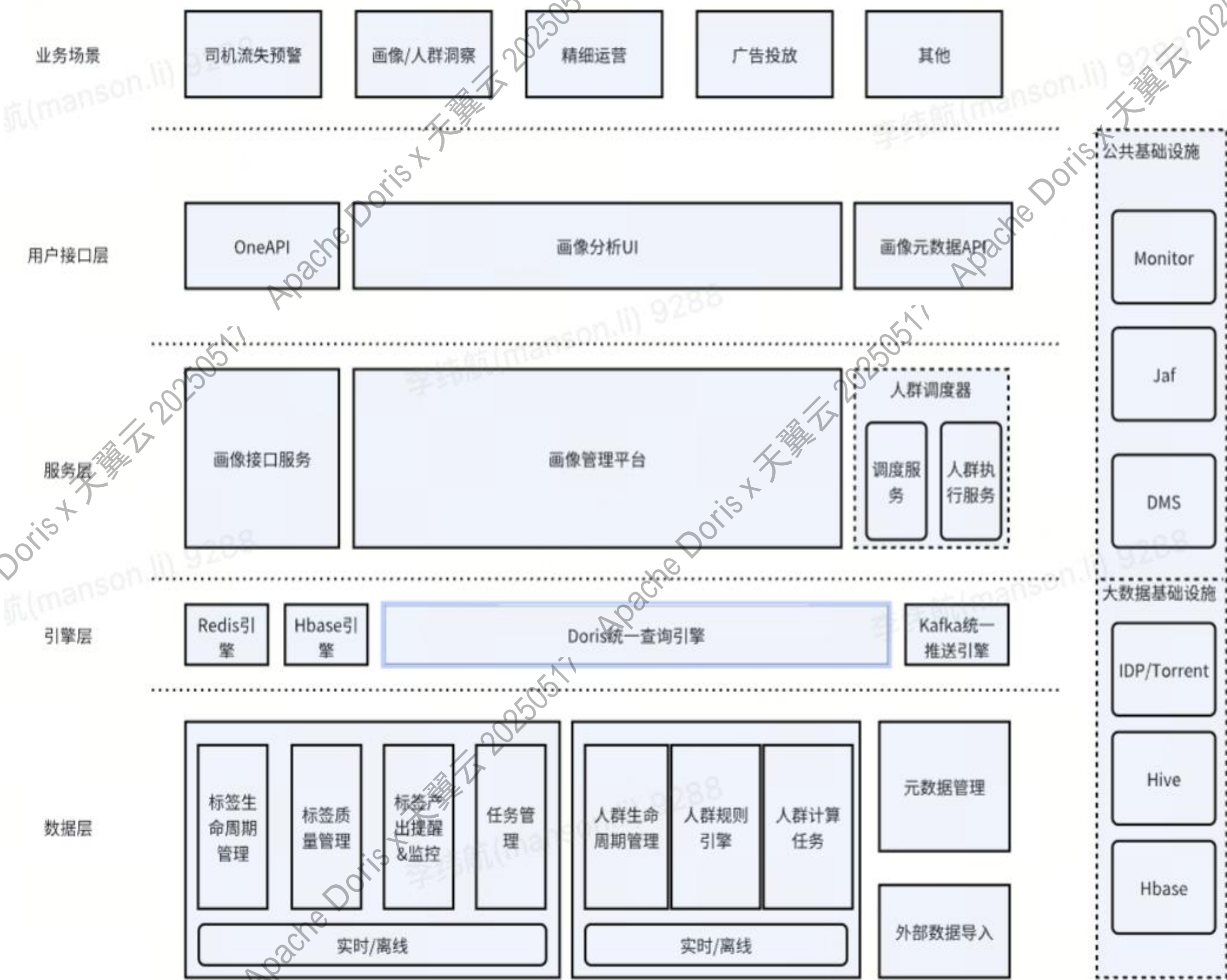
人群数

50000+

画像应用架构

Persona-BE（画像管理平台）：建造基于大数据体系的用户标签链路及系统，用于在特定业务形态下描述业务主体；

Persona-API（画像接口服务）：基于重点使用场景/对接系统搭建画像服务，从而精准地生成用户画像，给到对接业务方使用。



画像计算引擎演进

IMPALA + KUDU

- 早期出于成本原因，使用三方服务 + Impala + KUDU 架构作为画像人群计算的引擎。
- 人群计算耗时长，计算时长 10+min, 高峰期 30+min。
- 数仓导入 Impala 耗时长，经常超时不能及时产出人群数据，使用 Impala 进行数据导入 90+min。
- KUDU 存在动态列扩展困难，复杂查询效率一般，运维复杂度高等问题。

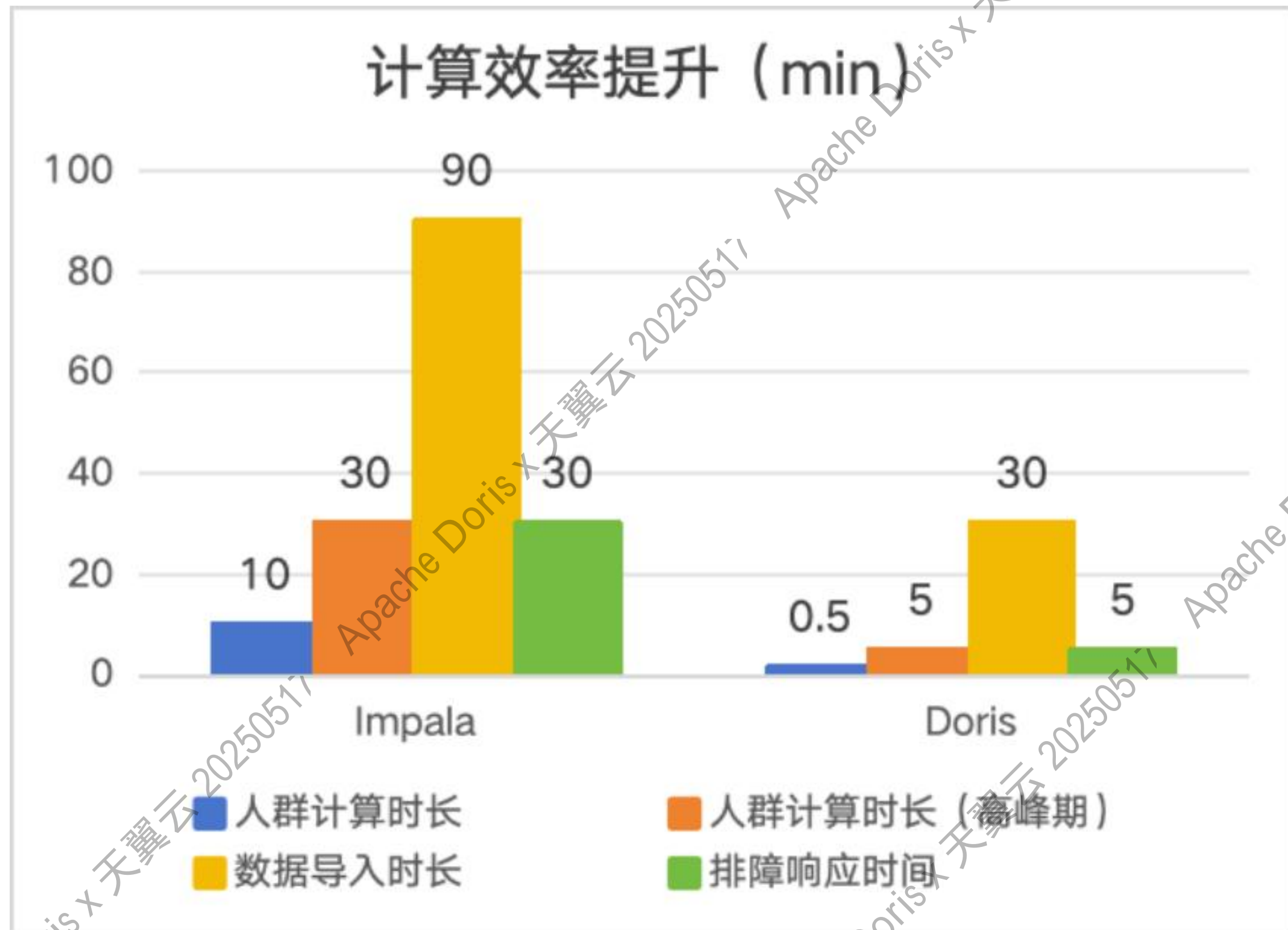
ES

- 开发成本高，多个场景需要适配。
- 多维分析能力不足。
- 仍然存在动态列扩展效率低，复杂查询效率等问题。
- 无法很好的解决货拉拉人群互相依赖计算的场景。

Doris

- 社区活跃度高。
- 拥有多种数据类型，可以使用 BITMAP 进行快速的交并计算。
- 具备向量化引擎，MPP 架构等，查询性能优秀。
- 可以使用不同的数据模型支撑各种标签类型与人群业务。

效果 & 收益



1. 画像服务

稳定性：接入Doris至今没有出现稳定性问题

2. 数据链路

时效性：接入Doris后数据链路相对之前Impala更加稳定；4亿行+200+列单表Doris的导入耗时30min内，使用Impala进行数据导入90+min

人群计算：接入Doris后单人群计算链路秒级别完成（高峰期1+min），使用Impala进行人群计算链路时长10+min（高峰期30+min）

目录

01 货拉拉画像服务背景与工程架构

02 应用层的数据模型设计与异构查询实现

03 货拉拉画像工程端的查询优化实践

04 后续规划

画像标签存储

关键挑战：

货拉拉（3000+）标签因业务属性（行为/属性/地理等）、聚合粒度（明细/聚合/人群）、更新时效（离线/近实时/实时）分裂为差异显著的数据体，不同业务属性的数据在更新频率、查询模式、存储密度上差异巨大。

用户使用画像标签的时候，对人群标签/明细标签/聚合标签等概念不清晰，只会进行简单的托拉拽拼接规则。

存储模型的设计选择？

- 宽表？挑战：动态更新、列拓展
- 高表？挑战：复杂查询、嵌套逻辑



数据模型设计

标签宽表模型

用户ID	完单量	点击次数
123	1	100
456	2	50

标签高表模型

标签名	标签值	标签版本	用户ID集合
城市	东莞	11111	Bitmap(123,810,111)
城市	深圳	11112	Bitmap(456,323,12344)

人群位图表模型

分群名	数据时间	用户ID集合
分群A	2023-01-01 00:00:11	Bitmap(123,810,111)
分群B	2023-01-02 00:00:11	Bitmap(456,323,12344)

标签宽表

- 存储低频更新的标签。
- 存储无法用高表存储的密集标签。
- 利用 Doris 的列式存储技术，利用索引、物化视图等优化手段，支持高效的多维分析。

标签高表

- 存储高频更新的稀疏标签。
- 支持秒级/分钟级数据更新，标签新增的场景下，可以规避宽表频繁 ALTER TABLE 导致的锁表问题。
- 多个标签的位图交并计算支持毫秒级响应。

人群位图表

- 存储标签规则圈选出来的人群结果。
- 用户 ID 集合使用 RoaringBitmap 压缩。
- 支持人群依赖计算，支持人群营销实验业务。

Create SQL

//标签宽表

```
CREATE TABLE wide_table (  
  user_id varchar(1000) NULL COMMENT "",  
  age bigint(20) REPLACE_IF_NOT_NULL NULL COMMENT "",  
  height bigint(20) REPLACE_IF_NOT_NULL NULL COMMENT ""  
ENGINE=OLAP AGGREGATE KEY( user_id ) COMMENT "OLAP"  
DISTRIBUTED BY HASH( user_id ) BUCKETS 40  
PROPERTIES ( ... )
```

//标签高表

```
CREATE TABLE high_table (  
  tag varchar(45) NULL COMMENT "标签名",  
  tag_value varchar(45) NULL COMMENT "标签值",  
  time datetime NOT NULL COMMENT "数据灌入时间",  
  user_ids bitmap BITMAP_UNION NULL COMMENT "用户集"  
) ENGINE=OLAP AGGREGATE KEY( tag , tag_value , time ) COMMENT  
"OLAP"  
DISTRIBUTED BY HASH( tag ) BUCKETS 128  
PROPERTIES ( ... )
```

//人群位图表

```
CREATE TABLE routine_segmentation_bitmap (  
  time datetime NOT NULL COMMENT "数据灌入时间",  
  seg_name varchar(45) NULL COMMENT "标签值",  
  user_ids bitmap BITMAP_UNION NULL COMMENT "人群ID集合"  
) ENGINE=OLAP AGGREGATE KEY( time , seg_name )  
COMMENT "OLAP" PARTITION BY RANGE( `time` ) (...)  
DISTRIBUTED BY HASH( seg_name )  
BUCKETS 128  
PROPERTIES (...,  
  "dynamic_partition.enable" = "true", ...);
```

人群圈选：异构组合查询

异构数据组合查询：

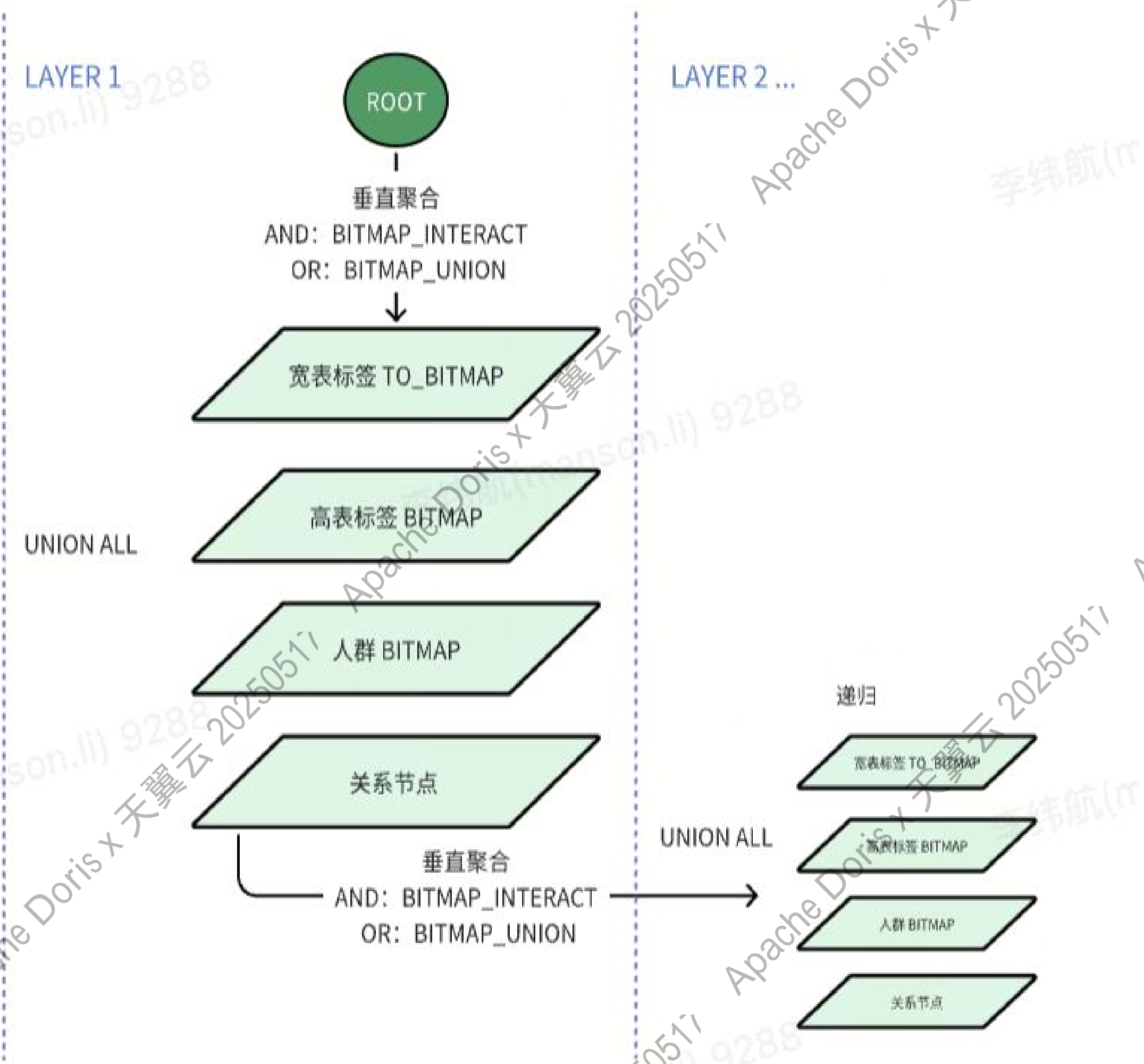
- 宽表：使用 TO_BITMAP 聚合多列过滤结果（如年龄+地域）。
- 高表：直接提取预存的行为标签位图（如点击商品X的用户ID集合）。
- 人群表：复用预计算位图（如AAA用户群），加速逻辑复用。

位图运算：

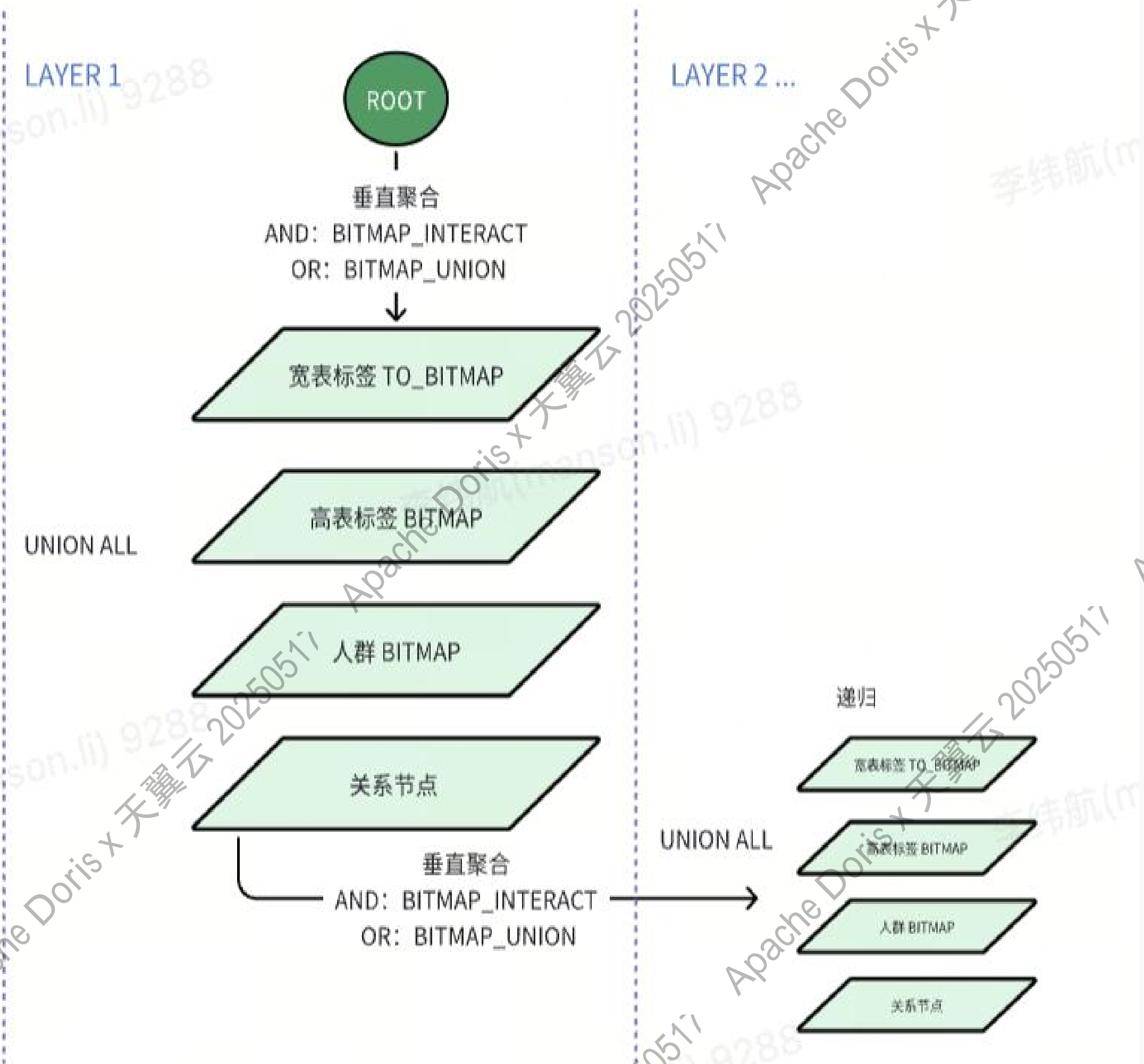
- 垂直聚合：UNION ALL 合并各层结果，BITMAP_INTERSECT/UNION 取交集（最终命中所有条件的用户）。
- 最终输出全局交集的用户ID位图，支持直接导出或对接营销系统。

核心优势：

- 灵活拓展：动态嵌套子查询支持无限层级规则（如“A且(B或C)且(D且(E或F))”），静态属性+动态行为+预计算人群无缝联动。
- 资源节省：复用预计算人群（人群表）减少计算压力；人群计算结果使用RoaringBitmap存储，不需要额外新增表。
- 业务友好：支持人群+标签的复杂混合嵌套查询，用户无需技术背景，拖拽组合标签条件即可配置生成精准用户群。

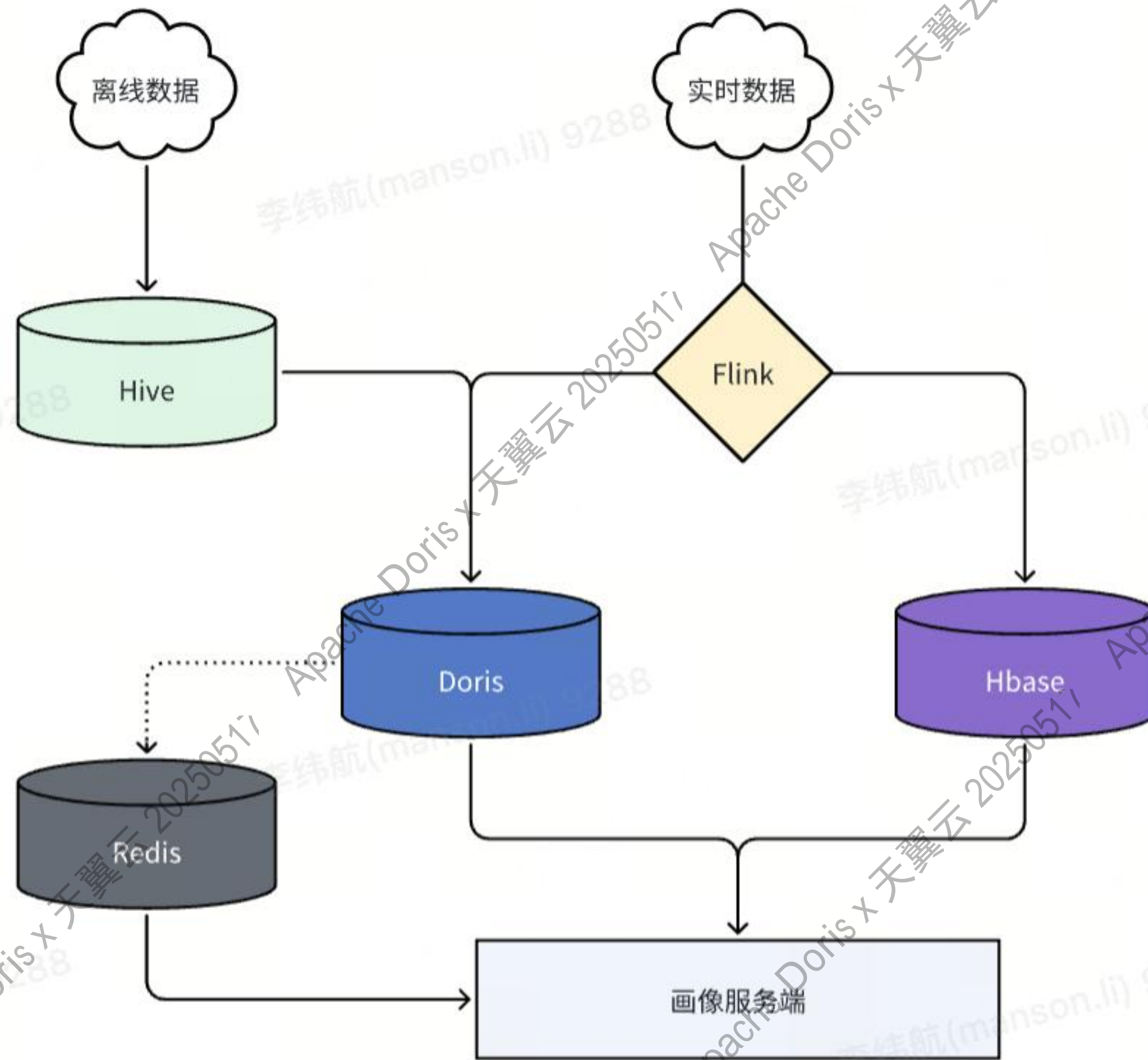


人群圈选：异构组合查询



```
SELECT BITMAP_INTERSECT(b) FROM (  
  -- Layer1: 宽表条件A  
  SELECT TO_BITMAP(...) AS b FROM 标签宽表 WHERE 条件A  
  UNION ALL  
  -- Layer1: 高表条件B  
  SELECT user_ids AS b FROM 标签高表 WHERE 条件B  
  UNION ALL  
  -- Layer1: 人群条件C  
  SELECT user_ids AS b FROM 人群表 WHERE 条件C  
  UNION ALL  
  -- Layer2: 嵌套子查询（新条件D/E/F）  
  SELECT BITMAP_INTERSECT(b) FROM (  
    SELECT TO_BITMAP(...) AS b FROM 标签宽表 WHERE 条件D -- 新宽表条件  
    UNION ALL  
    SELECT user_ids AS b FROM 标签高表 WHERE 条件E -- 新高表条件  
    UNION ALL  
    SELECT BITMAP_INTERSECT(...) AS b FROM 人群表 WHERE 条件F -- 新人群条件  
  )  
) t;
```

数据导入实践



实时/近实时标签

- 定义：秒级/小时级更新的动态数据（如点击、登录事件）。
- 数据源：Kafka 日志、API 埋点、云文件存储。
- 处理方式：
 - 秒级/分钟级标签：Flink -> Doris / Hbase。
 - 小时级标签：云文件存储 -> BrokerLoad。
- 场景：用户行为分析，用户实时人群。

离线标签

- 定义：T+1 更新的历史数据（如年龄、历史订单）。
- 数据源：Hive。
- 处理方式：
 - 数据量：3000+ 标签，4 亿+用户总量。
 - 定时调度：BrokerLoad。
 - 调优手段：宽表导入拆分多表和多个 BrokerLoad 任务。
数据量少的稀疏标签使用高表导入。
 - 导入效率：200+ 标签宽表 4 亿行+导入 30min 以内，高表标签导入 5min 以内。
- 场景：用户人群圈选、用户画像分析。

目录

01 货拉拉画像服务背景与工程架构

02 应用层的数据模型设计与异构查询实现

03 货拉拉画像工程端的查询优化实践

04 后续规划

离线人群包圈选SQL构建流程

且

且

且

用户属性满足 ...

user_nickname

等于

test ×

1↑

+

✎

?

×

expired_days

大于

10

?

×

user_sleep

等于

11 ×

+

✎

?

×

画布用户画像ljm

为真

+

?

×

添加

查询量级

保存用户群



规则转化为DSL

DSL优化

优化后的DSL
直译为SQL

```
{
  "content": {
    "type": "rules_relation",
    "relation": "and",
    "rules": [
      {
        "type": "rules_relation",
        "relation": "and",
        "rules": [
          {
            "type": "rules_relation",
            "relation": "and",
            "rules": [
              {
                "type": "profile_rule",
                "field": "user.user_nickname",
                "function": "equal",
                "params": [
                  "test"
                ]
              }
            ]
          }
        ]
      }
    ]
  },
  "Object{...},
```

DSL 优化 --> 生成 SQL

1. 为什么需要优化？

当前异构查询SQL痛点：

- 部分标签逻辑可合并，属于业务范围；
- 冗余扫描：同一表被多次扫描。
- 多层聚合：复杂嵌套的场景下，嵌套的 UNION ALL 和 BITMAP_INTERSECT 导致执行计划层级膨胀。
- 1.2.x 版本优化器策略未覆盖：升级到 1.2.x 后，出现了 bitmap 向量化读未下推的情况。
- 稳定性：高峰期人群计算时，内存占用高、网络传输量大，高内存开销影响集群稳定性。

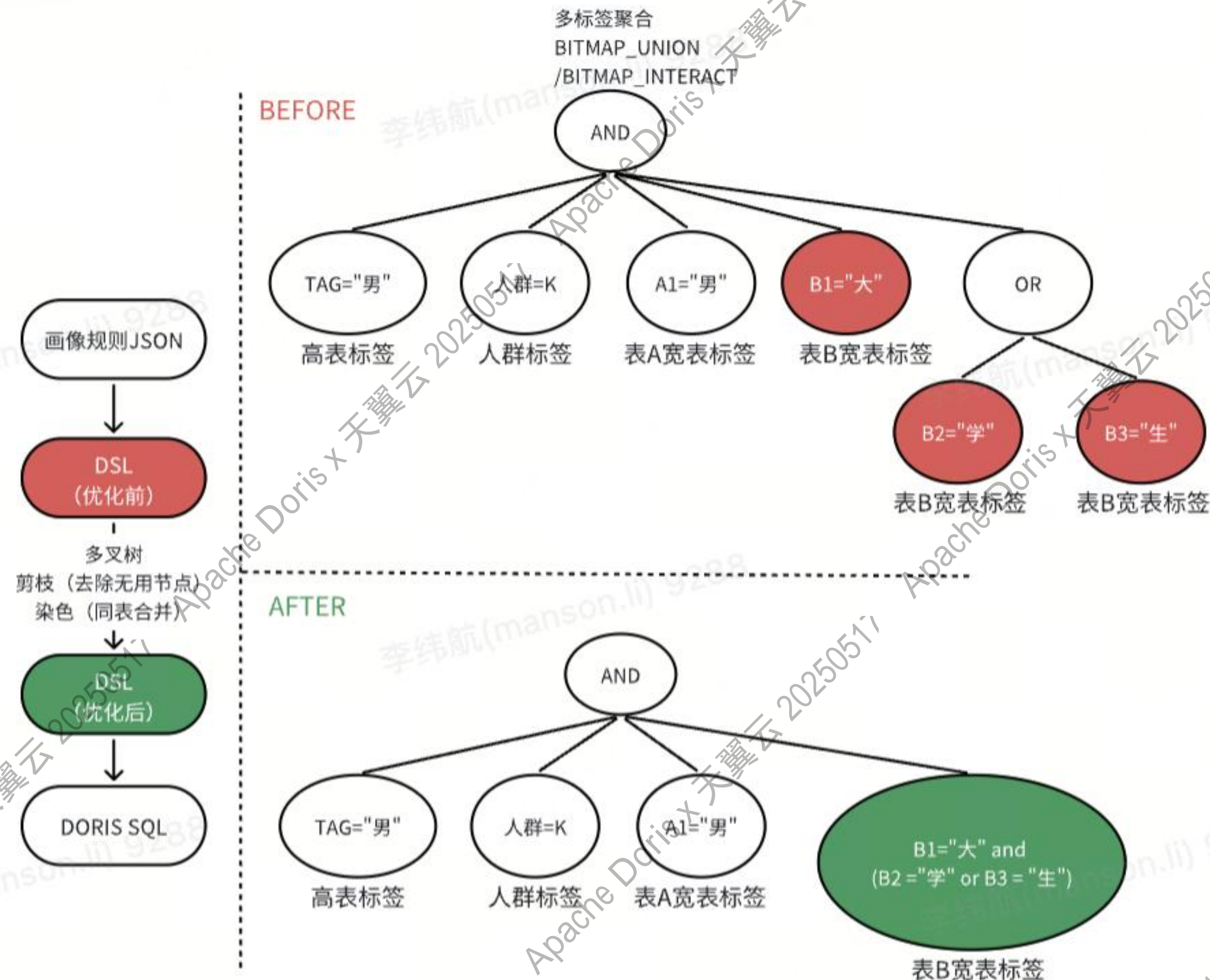
2. 如何实现优化？

DSL优化：

- 条件合并（染色）：将同类标记的标签条件合并为同个子查询。
- 结构扁平化（剪枝）：去除冗余的 AND/OR 逻辑节点

DSL -> SQL：

优化后的 DSL，直接翻译为 SQL，图内的每一个圆圈对应一个 BITMAP 子查询。



DSL 优化 --> 生成 SQL

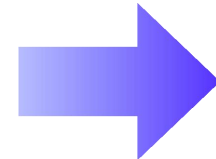
步骤

0. 遍历所有标签，对可合并的同类型的标签进行**类型标记**。
1. 剪枝阶段：
 - 后序遍历规则树。
 - 对每个【子查询节点】：
 - 若其子节点是另一个子查询节点，且该子节点仅有一个子节点，则剪枝。
 - 递归处理新插入的节点。
2. 染色阶段：
 - 后序遍历规则树。
 - 对每个【子查询节点】：
 - 收集所有【子节点类型】。
 - 若所有【子节点类型】相同，则父节点继承该类型。
3. 返回优化后的DSL规则树。
4. DSL 翻译为 SQL



输入

优化前 DSL 规则树



输出

Doris SQL

原始规则树 → [剪枝] → 扁平化层级 → [染色] → 类型合并 → 优化后规则树 → SQL

优化效果

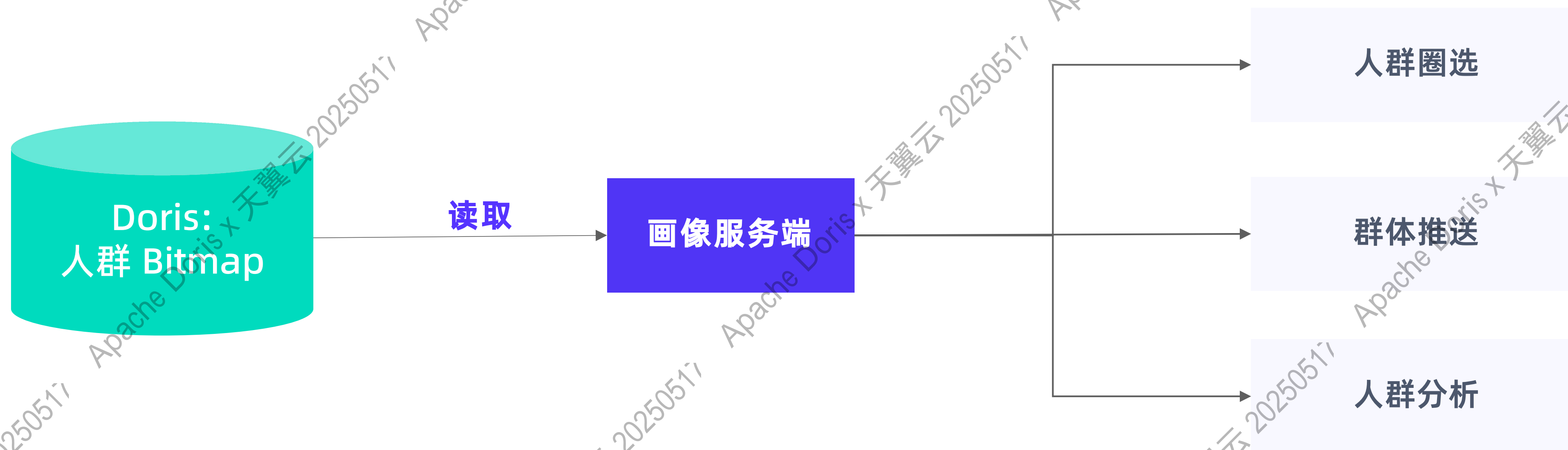
```
//优化前
SELECT BITMAP_INTERSECT(b) AS result
FROM (
  SELECT BITMAP_INTERSECT(b) AS b
  FROM (
    SELECT user_bitmap as b FROM user_bitmap WHERE group = 'A'
    UNION ALL
    SELECT TO_BITMAP(id) AS b FROM wide_table WHERE city = '东莞'
    UNION ALL
    SELECT TO_BITMAP(id) AS b FROM wide_table WHERE sex = '男'
  ) t1 ) t2;

//优化后
SELECT BITMAP_INTERSECT(b) AS result
FROM (
  SELECT user_bitmap as b FROM user_bitmap WHERE group = 'A'
  UNION ALL
  SELECT TO_BITMAP(id) AS b FROM wide_table WHERE city = '东莞' and
sex = '男'
) t1 ) t2;
```

EXPLAIN指标	优化前	优化后	变化
BITMAP_INTERSECT层级	2层	1层	减少1层
VEXCHANGE节点数	6个	4个	减少2个
聚合节点（VAGGREGATE）	5个	3个	减少2个

- **条件合并**：将优化前 BITMAP 子查询 3 的合并到子查询 2 的 WHERE 条件中，避免重复扫描表。
- **结构扁平化**：
 - BITMAP 子查询减少，减少数据分片合并次数。
 - 在业务圈选逻辑不变的情况下，聚合层级减少，降低人群计算时的内存峰值。
- **资源节省**：高峰期大规模人群计算减少 30%~50% 的内存开销。1.2.x 版本中，优化后 Jvm Heap Size 的峰值由 60% 降低到 20%。

人群位图读取实践



人群位图读取优化实践

	bitmap_to_string方案	EXPLODE+LATERAL VIEW流读	二进制直读+反序列化
实现方式	使用 bitmap_to_string 将位图转为逗号分隔的字符串（如 1,2,3），客户端按字符串解析用户 ID 列表。	通过 EXPLODE 函数展开位图为多行用户 ID，逐行返回明细数据流。	Doris 服务端设置 <code>return_object_data_as_binary=true</code> ，直接读取二进制数据，画像服务端基于 Doris 源码中的 Bitmap 协议反序列化。
网络与内存开销	体积膨胀 5~100 倍，内存占用高。	Doris 执行 EXPLODE 时需将位图全量展开为用户 ID 列表，对 Doris 服务端存在压力。	仅传输位图原始二进制（1 bit/用户），内存以及网络开销低。
开发与维护成本	简单易用，但需处理字符串解析异常（如分隔符冲突、类型转换错误、单个画像字符串过大）。	需维护画像服务端流读逻辑，开发维护成本高。	需深度定制解析逻辑（参考 Doris 源码的 BitmapValue 类），初期开发成本高，但长期维护稳定。
适用场景	测试、快速导出	人群多表关联分析	人群圈选后的用户 ID 全量读取

调优经验分享

问题1: 查询 bitmap/hll 类型的数据返回 NULL 的问题

解决方案:

在 1.1.x 版本中, 在开启向量化的情况下, 执行查询数据表中 bitmap 类型字段返回结果为 NULL 的情况下,

- 首先: set return_object_data_as_binary=true;
- 关闭向量化 set enable_vectorized_engine=false;
- 关闭 SQL 缓存 set [global] enable_sql_cache = false;

这里是因为 bitmap / hll 类型在向量化执行引擎中: 输入均为 NULL, 则输出的结果也是 NULL 而不是 0。

调优经验分享

```
6 left join [shuffle] (  
7 select  
8 id from (  
9 select  
10 bitmap_union(user_ids) as sss  
11 from  
12 user_profile.segmentation_bitmap  
13 where  
14 (  
15 seg_name = 'defaultd54aa49b3'  
16 and time = '2022-08-15 17:36:51'  
17 )  
18 or (seg_name = 'default')  
19 ) t1 lateral view explode_bitmap(sss) tmp as id  
20 ) t2 on t2.id = t3.u_lala_id;
```

Execute

Results

Execution failed: Error Failed to execute sql: java.sql.SQLException: (conn=697671) errCode = 2, detailMessage = Memory exceed limit. TableFunctionNode, while getting next batch. Backend: 172.18.65.100, fragment: 211974aac61244a5-a779a1b78735537c Used: 8615333888, Limit: 8589934592. You can change the limit by session variable exec_mem_limit.

```
select  
t2.id,  
user_ban  
from  
user_base_table_wide t3  
left join[shuffle] (  
select
```



问题2: 1.x 版本, BITMAP EXPLODE+LATERAL VIEW 后与大数据量的宽表 JOIN, 偶发内存异常导致的查询异常。

解决方案:

在 1.x 版本中避免在大规模数据查询场景 (结果行 2kw+) 使用类似的查询,

可以使用临时表, 或者分批查询的方案解决。

在 2.1.x 版本中, Doris 对 join 顺序进行了优化, 该错误不再复现。

目录

01 货拉拉画像服务背景与工程架构

02 应用层的数据模型设计与异构查询实现

03 货拉拉画像工程端的查询优化实践

04 后续规划

后续规划

Doris 接入画像所有实时业务

当前货拉拉画像服务的实时标签/人群点查主要使用 Hbase 和 Redis，基于稳定性和迁移升级成本的考虑，Doris 主要承担人群圈选、人群洞察、行为分析等作业。未来规划使用 **Doris 高版本**的架构，承担大部分的高并发的实时点查流量。

基础架构：Doris + 数据湖

在存储架构方面，货拉拉开始落地数据湖解决方案，画像平台将打通其他数据应用平台、实现超大规模数据的分析。

Thanks !

