阿里云实时推荐系统数据库设计

digoal

您也许感兴趣 | 新品发布 | 发烧好评 | 狠惠配

亚马逊 kindle 新入门款升级版6英寸电子墨水触控显示

¥539.00

SONY 索尼 DSC-RX100 M3 数码相机 黑色 F1.8-2.8大光

¥4599.00

亚马逊 kindle 新入门款升级版6英寸电子墨水触控显示

¥539.00

亚马逊 全 新Kindle Paperwhite 3 电子书阅读器 6英寸 黑

¥958.00

Microsoft 微软 Surface Pro 4 - Intel Core i5 128G存储4G

¥6599.00

业务背景

- ToB的用户推荐系统
- 业务数据包括USERID，APPID，TAGs。
  - 根据APPID+TAG组合，搜索复合条件的USERID
  - 查询单个用户的TAG
- 数据总量约10亿
- 单个APPID用户数设计量级1亿
- TAG设计量级1万
- 规模：10万亿user_tags

贴标签 - 业务指标

- **分钟级延迟**
  - 写入标签数据
  - 删除用户标签
  - 更新用户标签

圈人 - 业务指标

- 并发指标
  - 约**200~300**个ToB用户根据对应的APPID，根据TAG组合筛选需要的用户群体。
- 查询需求: 按TAG组合
  - 包含，不包含，或，与
  - and, or, not
- 响应时间指标
  - 毫秒级

开脑洞

- 表结构设计
  - appid, userid, tag1, tag2, ..., tag10000
- 查询写法
  - select appid,userid from tbl where tag1 **and** tag2 **and** ... **or** (not tagn) **or** (tagx **and** tagxx **or** tagxxx) ......
- 问题
  - 全字段索引？
  - 超宽表？
  - 存储空间？
  - 插入效率、查询效率？

用户目前的方案

- 目前没有产品可以支撑**1万列**的宽表
- 拆成N张表，通过PK关联
- 目前规模1亿，使用了8台主机（ADS）
- TAG更新延迟1天
- 查询响应 - 分钟级

方案1

- 数据结构
  - APPID， USERID， **TAG[]数组**
  - 单个数组最大长度1GB(约支持2.6亿个TAG(int4))
- 按APPID分区，随机分片
- query语法
  - 包含array2指定的所有TAG
    - 数组1包含数组2的所有元素
    - array1 @> array2
    - **支持索引检索**
  - 包含array2指定的TAG之一
    - 数组1与数组2有重叠元素
    - array1 && array2
    - **支持索引检索**
  - 不包含array2指定的所有tag
    - 数组1与数组2没有重叠元素
    - not array1 && array2
    - **不支持索引检索**

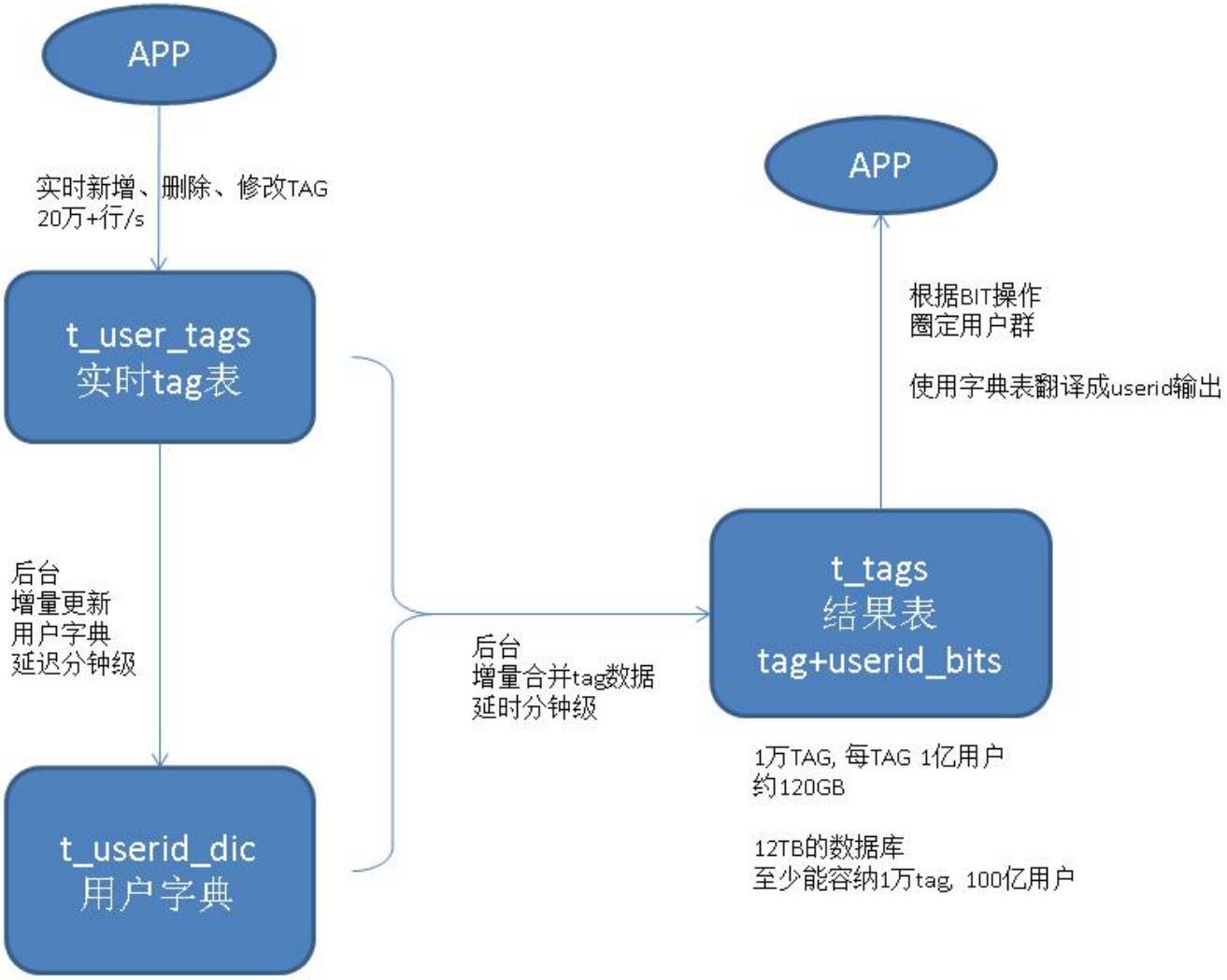
方案2

- 数据结构
  - APPID， USERID， TAG比特流
  - 单个BIT字段最大支持1GB长度BIT流（支持85亿个TAG）
  - 每个BIT代表一个TAG
- 按APPID分区，随机分片
- query语法
  - 都不支持索引
  - 包含bit2指定的所有TAG(需要包含的TAG对应的BIT设置为1，其他为0)
    - bitand(tag_bit,bit2) = bit2
  - 包含bit2指定的TAG之一(需要包含的TAG对应的BIT设置为1，其他为0)
    - bitand(tag_bit,bit2) > 0
  - 不包含bit2指定的所有tag (需要包含的TAG对应的BIT设置为1，其他为0)
    - bitand(tag_bit,bit2) = zerobit(10000) -- '0000...0000000'

方案3

- 结构设计
  - appid, tagid, userid比特流(1亿用户12MB)
- 延迟合并 + query时合并
  - data -> 明细表 -> 增量聚合 -> appid,tagid,userid
- query
  - 都支持索引
  - 包含这些tags的用户
    - userids (bitand) userids .....
    - 结果为bit位为1的用户
  - 不包含这些tags的用户
    - userids (bitor) userids .....
    - 结果为bit位为0的用户
  - 包含这些tags之一的用户
    - userids (bitor) userids .....
    - 结果为bit位为1的用户

方案3 - 增量合并


APP
实时新增、删除、修改TAG
20万+行/s
t_user_tags
实时tag表
后台
增量更新
用户字典
延迟分钟级
t_userid_dic
用户字典
后台
增量合并tag数据
延时分钟级
t_tags
结果表
tag+userid_bits
1万TAG, 每TAG 1亿用户
约120GB
12TB的数据库
至少能容纳1万tag, 100亿用户
APP
根据BIT操作
圈定用户群
使用字典表翻译成userid输出

TAG类型 数组 vs bit

- 空间占用
  - 数组，与用户平均TAG数有关
    - **1万个元素约80KB**
      - select pg_column_size(id) from (select (select array_agg(10000*random()) from generate_series(1,10000)) id from generate_series(1,1)) t;
      - 80024
  - BIT，固定长度
    - **1万个bit约1.25K**
      - select pg_column_size(id) from (select (select (string_agg(mod((2*random())::int,2)::text,''))::varbit from generate_series(1,10000)) id from generate_series(1,1)) t;
      - 1258
- 索引支持
  - 数组支持(包含，包含任意)的索引查询，不支持(不包含)的索引查询
  - bit不支持本CASE的索引（使用方案3无需bit索引）
- 性能

空间评估

- 1亿用户，1万标签
- 方案1
  - 裸数据约8TB
  - 索引约8TB
- 方案2
  - 裸数据约120GB
  - 无索引，全量计算
- 方案3
  - 裸数据约120GB
  - 索引很小，MB级(仅仅1万条记录)

PostgreSQL性能指标

- 方案1
  - 如果使用方案1，建议按USERID拆库(使用plproxy并行)
- 性能
  - 插入 速度(10w/s)
  - 更新 tag速度(10w/s)
  - 删除 tag速度(10w/s)
  - 查询速度(返回结果集10万条,1秒左右)
  - 并发指标(500+)

PostgreSQL性能指标

- 方案2
- BIT
  - 插入速度
    - **(批量)每秒插入24.5万, 326MB/s**
  - 更新、删除 tag速度
    - create or replace function randbit(int) returns varbit as $$
    - select (string_agg(mod((2*random())::int,2)::text,''))::varbit from generate_series(1,$1);
    - $$ language sql strict volatile;
    - create or replace function zerobit(int) returns varbit as $$
    - select (string_agg('0',''))::varbit from generate_series(1,$1);
    - $$ language sql strict immutable;
    - update t_bit set tags=randbit(10000) where userid=:id;
    - **每秒更新、删除1万 记录，响应时间约4毫秒**
  - 查询速度
    - do language plpgsql $$
    - declare
    - sql text;
    - bit1 varbit := randbit(10000);
    - bit2 varbit := randbit(10000);
    - bit3 varbit := randbit(10000);
    - zbit varbit := zerobit(10000);
    - begin
    - set max_parallel_workers_per_gather =27;
    - sql := 'select * from t_bit where bitand(tags,'''||bit1::text||''')='''||bit1::text||''' and bitand(tags,'''||bit2::text||''')>bit''0'' and bitand(tags,'''||bit3::text||''')='''||zbit::text||'''';
    - raise notice '%', sql;
    - -- execute sql;
    - end;
    - $$;
    - **开27个并行, 17秒。**
  - 并发指标

PostgreSQL性能指标

- 方案3
- BIT
  - 前端写入速度
    - **每秒插入24.5万, 326MB/s，满足实时标签修正**
  - 合并速度
    - (批量)10亿用户bit/秒, 100TAG/s。满足分钟级延迟。
  - 更新、删除 tag速度
    - 每个TAG每次变更1万个用户。
    - 100 tag/s, 100毫秒/tag，支持tag并行更新。
    - 换算成user tag绑定/删除 ops ~= 100万 users/s。
  - 查询速度
    - 任意条件组合
    - **毫秒级**返回用户数据
    - 支持流式返回用户集合
  - 并发指标
    - 使用游标返回数据
    - **支持500+并发**

前后对比

- 优化前
  - ADS(8台物理机)
  - 1亿用户
  - 标签更新，延迟1天
  - 查询并发，200
  - 查询响应，分钟
- 优化后
  - PostgreSQL(1台RDS PG)
  - 100亿用户,1万tag
  - 标签更新，延迟分钟级
  - 查询并发，500+
  - 查询响应，毫秒

方案3 - 后期优化

- 多行优化（同一个tag占多行，每行存储部分USERID BITS），减少垃圾版本大小(改少量BIT)，减少锁冲突(不同分段可以同时更新,行锁原因)
- BIT字段块级元数据(比如每个块有多少0,1？)