MongoDB + PostgreSQL
中文社区南京大会

苏宁大规模标签场景应用实践

陈华军
微信：chenhj_07
苏宁易购 数据库研发中心

目录

如何快速找到目标营销人群(圈人)?

- **精准**
  - ✓基于准确的用户画像
- **实时**
  - ✓实时查询满足条件的目标人群
  - ✓灵活多变的查询条件组合
  - ✓用户画像的实时更新
- **可扩展**
  - ✓支撑数亿甚至数十亿的用户规模
  - ✓支撑百万，亿，甚至百亿/千亿规模的标签

常规技术方案的短板

曾经用过的方案

方案	问题
Hive	由于每次查询都需要对亿级的用户表做全量扫描，资源消耗极大，响应时间很长
Spark+ElasticSearch	利用ES的索引技术结合并行处理，查询性能比Hive有几十倍的提升； 业务上经常需要新增标签(字段)，导致必须重新灌全量ES数据，非常耗时。

参考：https://www.infoq.cn/article/9*dSWwPT6UqSH1LSeNzq

目录

- 海量用户下精准营销的挑战
- roaringbitmap在圈人场景中的作用
- PostgreSQL + roaringbitmap的最佳实践
- 如何用分布式PG支撑百亿标签实时查询

从 搜索“人”到搜索“标签(人群)”

用户ID	性别	城市	……
1	男	南京	
2	女	北京	
…			

标签类型	标签值	用户ID集合
性别	男	{1,3,5,6,…}
性别	女	{2,4,7,8…}
城市	南京	{1,4,17,…}
城市	北京	{2, 17,98…}
…		

两种处理方式的对比

	搜”人”	搜”标签”
记录数	十亿级	百万级
索引数	几十，上百	一个
新增标签	修改全量记录	仅插入新标签记录
计算方式	组合条件过滤	“人群”集合的交并差运算

如何存储“人群”集合?

- Bitmap适合大集合的交并差运算

- roaringbitmap是一种已被业界广泛使用的高效的bitmap压缩算法，使用者包括Elasticsearch，Durid，Hive，Spack，InfluxDB等，详见 http://roaringbitmap.org/

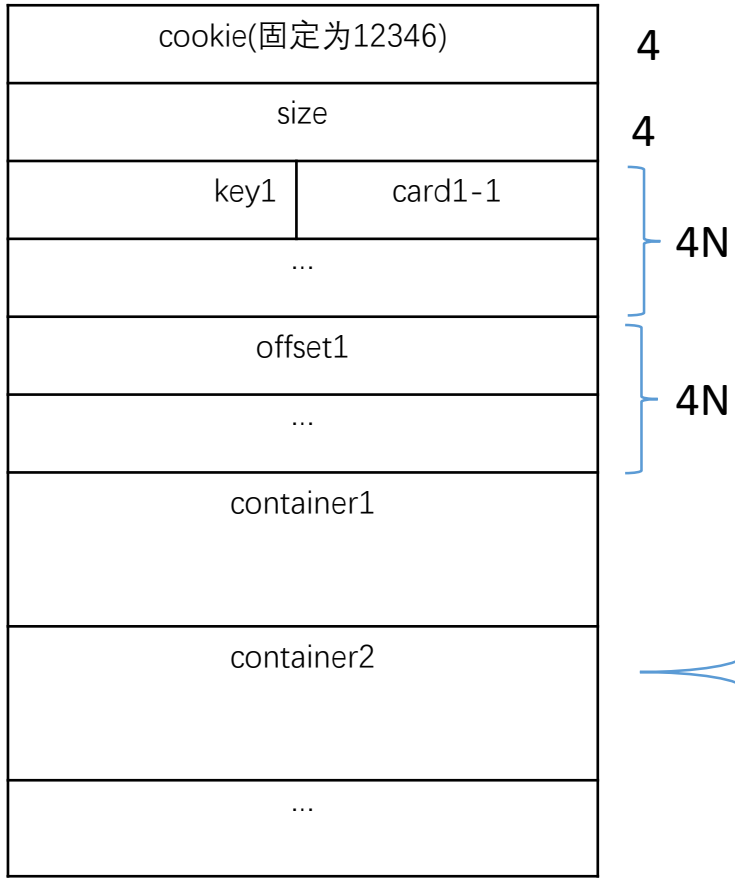
注： Elasticsearch在查询时内部用roaringbitmap临时存储不同倒排索引的结果集做交并运算。

Roaringbitmap的存储格式

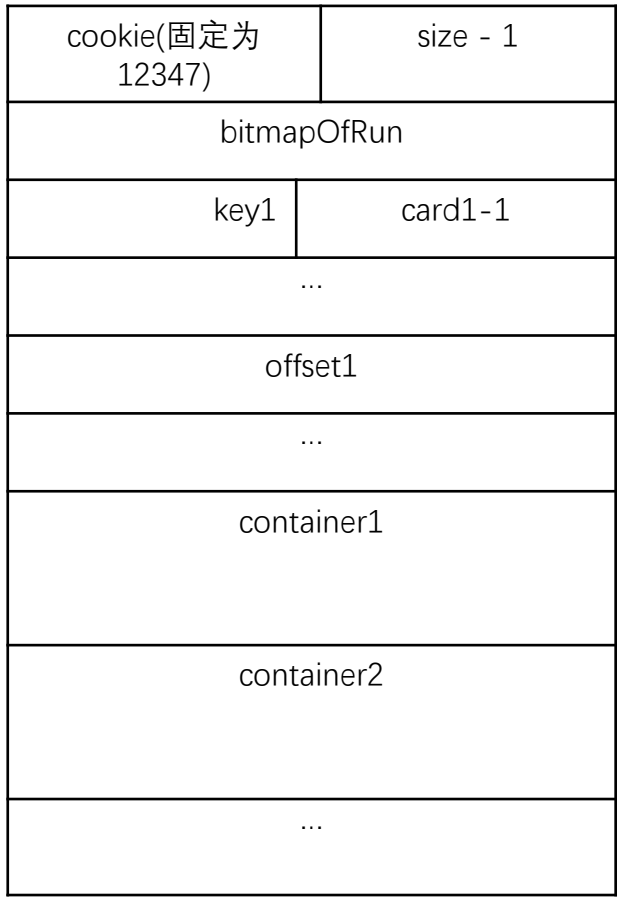
Roaringbitmap 将32位的整形拆分成高16位和低16位，高16位作为容器的key，低16位作为value存储在3种不同的容器中。每个容器最多存储65536个值，最多有65536个容器。每种容器存储方法不同适用于不同场景

容器类型	存储形式	容器大小	容量	容器转换
Array	有序的short数组	基数*2Byte	4096	基数超过4096时自动转换为Bitset容器
Bitset	8KB大小的bitset,每个值对应一个特定的bit位	8KB	65535	基数低于4096时自动转换为Array容器
run	RLE(Run Length Encoding行程长度编码)**格式，**由成对的short型value+length组成。 比如10,11,12,13压缩为10,3	4B~128KB	65535	调用run优化且run格式存储占用空间更小时实施转换

■ 不包含RUN容器时的序列化格式



■ 包含RUN容器时的序列化格式



4

(N+7)/8（所有run容器对应的bit位设置为1，其它为0）

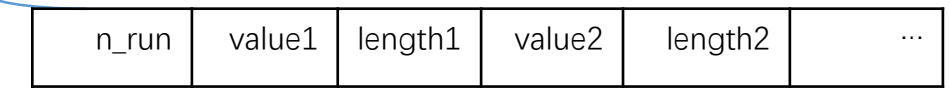
4N

4N（可选，N<4时没有）

Array容器:short数组

BitSet容器:8KB的bitset

Run容器: 2+ 4*run个数



参考： https://github.com/RoaringBitmap/RoaringFormatSpec/

目录

pg_roaringbitmap插件

- **基于gpdb-roaringbitmap并进行了功能增强(操作符，函数，并行聚合)**
- **支持PostgreSQL 10，11**

pg_roaringbitmap安装

- 从github下载pg_roaringbitmap
- 编译并安装插件

```
su – postgres
make
sudo make install
```

- 登录到目标数据库安装扩展

```
create extension roaringbitmap
```

pg_roaringbitmap使用示例(部分功能)

■Bitmap Calculation (OR, AND, XOR, ANDNOT)

```
SELECT roaringbitmap('{1,2,3}') | roaringbitmap('{3,4,5}');
SELECT roaringbitmap('{1,2,3}') & roaringbitmap('{3,4,5}');
SELECT roaringbitmap('{1,2,3}') # roaringbitmap('{3,4,5}');
SELECT roaringbitmap('{1,2,3}') - roaringbitmap('{3,4,5}');
```

■Bitmap Aggregate (OR, AND, XOR, BUILD)

```
SELECT rb_or_agg(bitmap) FROM t1;
SELECT rb_and_agg(bitmap) FROM t1;
SELECT rb_xor_agg(bitmap) FROM t1;
SELECT rb_build_agg(e) FROM generate_series(1,100) e;
```

■Cardinality

```
SELECT rb_cardinality(rb_build('{1,2,3}'));
```

pg_roaringbitmap性能参考

■测试SQL： select rb_cardinality(rb_or_agg(bitmap)) from bitmaptb

数据量	bitmap基数（注1）	整形范围	表大小(字节)	查询时间(ms)	平均记录大小(字节)	平均每整形占用空间(字节)	并行度	计算速度(次/core/秒)
10000000	1	1w	521789440	1450.932	52	52	2	3446061
10000000	1	10w	521789440	1442.005	52	52	2	3467394
10000000	1	1000w	521789440	1439.639	52	52	2	3473093
10000000	1	10亿	521789440	1463.23	52	**52**	2	**3417098**
10000000	10	1w	682672128	1407.813	68	6.8	2	3551608
10000000	10	10w	765607936	1868.86	77	7.7	2	2675428
10000000	10	1000w	1342955520	4377.34	134	13.4	2	1142246
10000000	10	10亿	1412415488	4696.178	141	**14.1**	2	**1064696**
10000000	1000	1w	20480000000	10334.641	2048	2.0	2	483810
10000000	1000	10w	28277710848	21462.611	2828	2.8	2	232963
10000000	1000	1000w	41931038720	106924.884	4193	4.2	2	46762
10000000	1000	10亿	1.04045E+11	531798.695	10404	**10.4**	2	**9402**
1000	1000000	1000w	1302405120	734.808	1302405	1.3	1	1361
1000	1000000	10亿	2200985600	6086.907	2200986	**2.2**	1	**164**
1000	10000000	1000w(注2)	1302405120	714.316	1302405	0.13	1	1400
1000	10000000	10亿	20766810112	41784.536	20766810	**2.1**	1	**24**

注1）该基数值只是插入到bitmap中的随机数，实际基数在测试数据去重后小于该数
注2）实际基数在测试数据去重后大概是500w，实际每整数占用大小应该大约是0.26
注3）测试环境:16C/128G/3000G SSD物理机+ CentOS 7.3 + PostgreSQL 10.2 + pg_roaringbitmap 0.3

超大基数bitmap的存取

- 如果使用unnest(rb_to_array()) 获取大结果集，不仅速度慢，而且受array类型最大1GB的限制，在bit位超过7000万时发生错误。

```
postgres=# select count(*) from unnest(rb_to_array(rb_fill('{1,10,100}',1,70000000)));
ERROR: invalid memory alloc request size 1073741824
```

基于Roaringbitmap二进制格式的存取

采用基于Roaringbitmap二进制格式的传输，不仅降低资源消耗，性能也有几十倍以上的提升。示例如下:

- 表定义:

```
create table testtb(id int, bitmap roaringbitmap);
```

- 读取bitmap数据:

```
String sql = "select bitmap::bytea from testtb where id = ?";
PreparedStatement stmt = conn.prepareStatement(sql);
stmt.setInt(1, 1);
ResultSet rs = stmt.executeQuery();
while(rs.next()){
    RoaringBitmap rb = new RoaringBitmap();
    DataInputStream is = new DataInputStream(rs.getBinaryStream(1));
    rb.deserialize(is);
    is.close();
}
rs.close();
stmt.close();
```

■ 写入bitmap数据：

```
String sql = "INSERT INTO testtb(id, bitmap) VALUES (?,?::bytea::roaringbitmap)";
PreparedStatement stmt = conn.prepareStatement(sql);
RoaringBitmap rb = RoaringBitmap.bitmapOf();
for(int i = 0; i < 10000000; i ++)
   rb.add((int)(1+Math.random()*100000000));
rb.runOptimize(); // run存储格式优化
byte[] array = new byte[rb.serializedSizeInBytes()];
 try {
    rb.serialize(new java.io.DataOutputStream(new java.io.OutputStream() {
    int c = 0;
    public void flush() {
    }
    public void close() {
    }
    public void write(int b) {
      array[c++] = (byte) b;
    }
    public void write(byte[] b) {
      write(b, 0, b.length);
    }
    public void write(byte[] b, int off, int l) {
      System.arraycopy(b, off, array, c, l);
      c += l;
    }
  }));
 } catch (IOException ioe) {
   throw new RuntimeException("unexpected error while serializing to a byte array");
 }
stmt.setInt(1, 1);
ByteArrayInputStream byteArrayInputStream = new ByteArrayInputStream(array);
stmt.setBinaryStream(2, byteArrayInputStream, array.length);
stmt.executeUpdate();
stmt.close();
```

注：需要pg_ roaringbitmap 0.4以上版本，之前版本必须通过text类型中转(bitmap::text::bytea和?::bytea::roaringbitmap)

目录

某买家分析场景

- ✓ 按照不同的维度组合(品类、品牌、大区、门店...)分析会员的新老买家类、留存率类、复购类以及沉睡类指标等
- ✓ 每个维度组合的维度值数量多的有几十万
- ✓ 每个维度值涉及的用户数多的有数百万
- ✓ 用户总数数亿
- ✓ 计算纯新买家，次新买家需要和所有历史买家(亿级别)做集合运算
- ✓ 所有维度+日期的组合(每个组合相当于一个标签)有**几百亿**

当前生产最大单表已达到200多亿。超出了正常的数据库单机容量。怎么办？

简单的查询示例：
20190520日当天，新买家数排名Top100的品牌

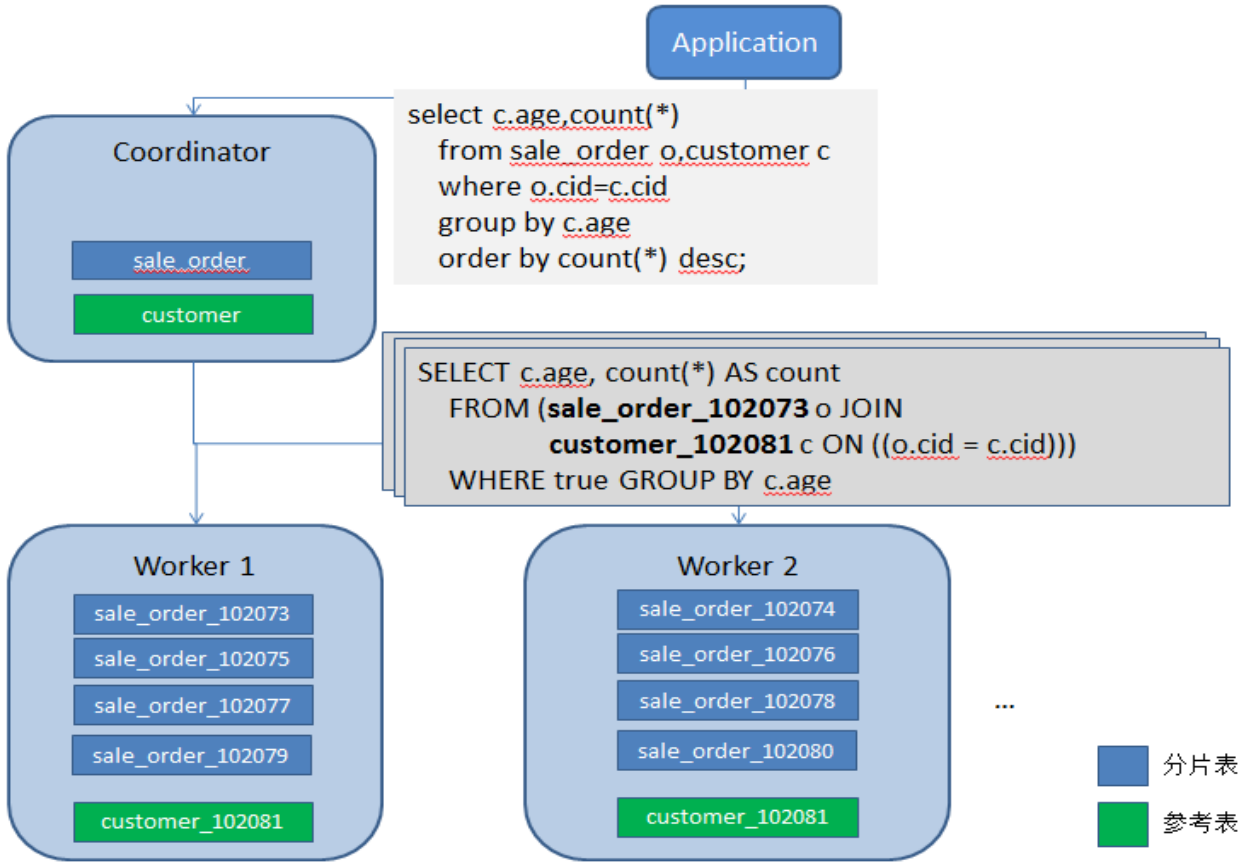
```
SELECT brand_cd,
     rb_and_cardinality( bitmap_cur, bitmap_sum ) AS oldusercount,       ---老买家数
     rb_andnot_cardinality( bitmap_cur, bitmap_sum ) AS newusercount  ---新买家数
FROM
 ( SELECT brand_cd,
       rb_or_agg(bitmap_cur) AS bitmap_cur,
       rb_or_agg(bitmap_sum) AS bitmap_sum
  FROM member_order
  WHERE statis_date = '20190520'
  GROUP BY brand_cd )a
ORDER BY newusercount DESC limit 100;
```

PG + Citus = 分布式HTAP

■ 适用场景
✓ 实时数据分析
✓ 多租户应用

■ 数据分布
✓ 分片表
- ➢ hash
  按分片字段hash值分布数据
- ➢ Append
  分片size增加到一定量自动创建新的分片

✓ 参考表
仅一个分片，每个worker一个副本，主要用于维度表。
✓ 本地表



注：目前苏宁已在生产上大量部署Citus，但尚未引入Greenplum，所以选择的Citus+roaringbitmap的分布式方案。并且根据德哥的博客，GP不支持把bitmap聚合下推到segment节点。参考:https://yq.aliyun.com/articles/405191

Citus + pg_roaringbitmap

1. 修改Citus支持roaringbitmap聚合函数

src/include/distributed/multi_logical_optimizer.h

```
typedef enum
{
                                AGGREGATE_INVALID_FIRST = 0,
                                AGGREGATE_AVERAGE = 1,
                                AGGREGATE_MIN = 2,
                                AGGREGATE_MAX = 3,
                                AGGREGATE_SUM = 4,
                                AGGREGATE_COUNT = 5,
                                AGGREGATE_ARRAY_AGG = 6,
                                AGGREGATE_RB_AND_AGG = 7,
                                AGGREGATE_RB_OR_AGG = 8
} AggregateType;
...
static const char *const AggregateNames[] = {
                                "invalid", "avg", "min", "max", "sum",
                                "count", "array_agg", "rb_and_agg", "rb_or_agg"
};
```

2. 安装配置citus

(略)

3. 安装pg_roaringbitmap插件

```
create extension roaringbitmap;
select run_command_on_workers('create extension roaringbitmap');
```

4.在所有节点上修改Citus的copy格式，优化网路传输(可选)

```
alter system set citus.binary_master_copy_format = on;
select pg_reload_conf();
```

5.创建分片表

```
create table member_order(
   id int default round(random()*100000000), -- 随机分片字段(用于打散数据)
   statis_date date,                --统计日期
   brand_cd text,                   -- 品牌
   ...,                             --其它维度(略)
   bitmap_cur roaringbitmap,        --当天买家
   bitmap_sum roaringbitmap); --历史买家

create index ...                    --略

select create_distributed_table('member_order','id');  --创建citus分片表
```

注： rb_and_cardinality_agg()和rb_or_cardinality_agg()必须改写成rb_cardinality(rb_and_agg(...))和rb_cardinality(rb_or_agg(...))

大结果集聚合带来的CN性能瓶颈

SQL：

```
app_db=# explain SELECT brand_cd,
     rb_and_cardinality( bitmap_cur, bitmap_sum ) AS oldusercount,       ---老买家数
     rb_andnot_cardinality( bitmap_cur, bitmap_sum ) AS newusercount  ---新买家数
FROM
  ( SELECT brand_cd,
        rb_or_agg(bitmap_cur) AS bitmap_cur,
        rb_or_agg(bitmap_sum) AS bitmap_sum
   FROM member_order
   WHERE statis_date = '20190520'
   GROUP BY brand_cd )a
ORDER BY newusercount DESC limit 100;
```

执行计划：

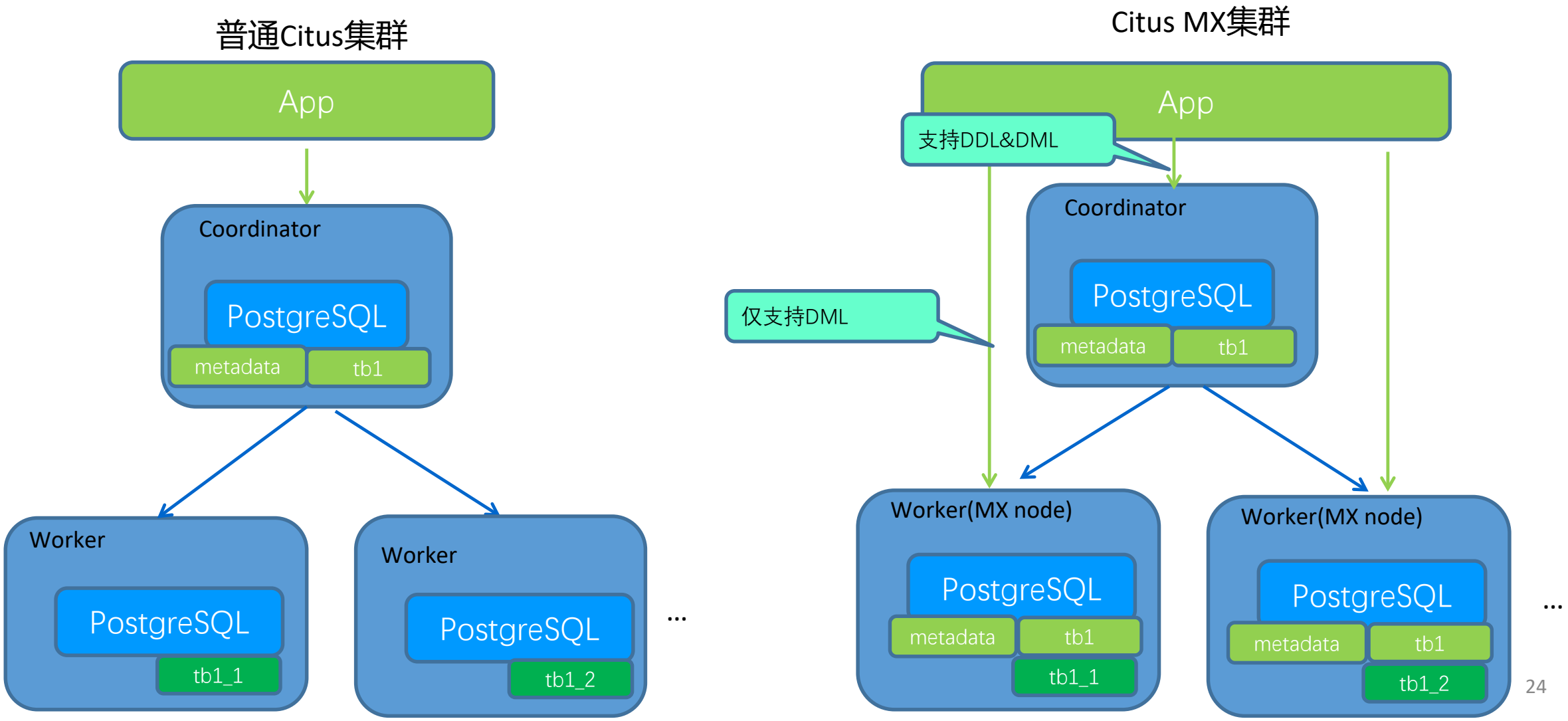
```
Custom Scan (Citus Router)  (cost=0.00..0.00 rows=0 width=0)
  ->  Distributed Subplan 3_1
     ->  HashAggregate  (cost=0.00..0.00 rows=0 width=0)
        Group Key: remote_scan.brand_cd
        ->  Custom Scan (Citus Real-Time)  (cost=0.00..0.00 rows=0 width=0)
           Task Count: 32
           Tasks Shown: One of 32
           ->  Task
              Node: host=10.47.147.130 port=6432 dbname=app_db
              ->  GroupAggregate  (cost=17.65..17.72 rows=3 width=96)
                 Group Key: brand_cd
                 ->  Sort  (cost=17.65..17.66 rows=3 width=96)
                    Sort Key: brand_cd
                    ->  Seq Scan on member_order_102033 member_order  (cost=0.00..17.62 rows=3 width=96)
                       Filter: (statis_date = '2019-05-20'::date)
 Task Count: 1
 Tasks Shown: All
 ->  Task
     Node: host=10.47.147.130 port=6432 dbname=app_db
     ->  Limit  (cost=53.22..53.47 rows=100 width=48)
        ->  Sort  (cost=53.22..55.72 rows=1000 width=48)
           Sort Key: (rb_andnot_cardinality(intermediate_result.bitmap_cur, intermediate_result.bitmap_sum)) DESC
           ->  Function Scan on read_intermediate_result intermediate_result  (cost=0.00..15.00 rows=1000 width=48)
```

CN上最终rb_or_agg聚合

WK上初次rb_or_agg聚合

- 当进行多维度分组时，维度组合可能达到100w量级， 32个分片的结果集汇总到CN就是大约6000w记录
- CN从Worker上收集6000w记录，以及对这6000w记录执行**最终的rb_or_agg()非常耗时（100秒以上）**

Citus多CN架构(Citus MX)



优化：把最终聚合分散到所有WK上

1. 配置Citus多CN架构

在CN节点的postgresql.conf中添加下面的参数

```
citus.replication_model='streaming‘
```

从CN复制元数据到所有worker节点

```
SELECT start_metadata_sync_to_node(‘$cituswk1_ip’, $cituswk1_port);
SELECT start_metadata_sync_to_node(‘$cituswk2_ip', $cituswk2_port);
SELECT start_metadata_sync_to_node('$cituswk3_ip', $cituswk3_port);
…
```

2. 创建中间表

```
set citus.shard_count =8; --建议中间表分片数设置为等于worker数

create unlogged table tb_dispatch(
   brand_cd text,
   …  --其他维度(略)
   bitmap_cur roaringbitmap,
   bitmap_sum roaringbitmap);

elect create_distributed_table('tb_dispatch','brand_cd',colocate_with=>'none');
```

选择其中一个分组维度作为分片字段

3. 初次聚合并将结果写入到中间表（在所有分片上并发执行）

```
truncate tb_dispatch;

select run_command_on_shards('member_order'::regclass,$$
                insert into tb_dispatch
                        SELECT brand_cd,
                                rb_or_agg(bitmap_cur) AS bitmap_cur,
                                rb_or_agg(bitmap_sum) AS bitmap_sum
                          FROM %s
                          WHERE statis_date = '20190520'
                          GROUP BY brand_cd
$$);
```

- ✓ 该SQL将并行在每个分片上产生一个SELECT和N(中间表分片数)个INSERT。如果原始表有32个分片,中间表有8个分片，将在WK上同时产生32+32*8=288个连接。因此该SQL不适宜过多并发执行。
- ✓ 本例中中间结果6000w

4. 从中间表收集数据进行最终聚合

1. 在WK上对中间表进行初次聚合（本例中初次聚合结果集100w）
2. 在CN上收集WK上初次聚合的结果进行最终聚合

注:中间表的分片字段包含在最终查询的分组字段中，不同分片上初次聚合的结果不存在重复。

```
with tmp as
(
SELECT brand_cd,
     rb_and_cardinality(rb_or_agg(bitmap_cur), rb_or_agg(bitmap_sum) ) AS oldusercount,       ---老买家数
     rb_andnot_cardinality(rb_or_agg(bitmap_cur), rb_or_agg(bitmap_sum) ) AS newusercount  ---新买家数
FROM tb_dispatch
GROUP BY brand_cd)
select * from tmp order by newusercount desc limit 100;
```

优化后，性能提升10倍，执行时间压缩到10秒以下

参考

- ✓ https://yq.aliyun.com/articles/405191
- ✓ https://www.infoq.cn/article/9*dSWwPT6UqSH1LSeNzq
- ✓ https://github.com/RoaringBitmap/CRoaring
- ✓ https://www.jianshu.com/p/af6a7ef67518
- ✓ https://zhuanlan.zhihu.com/p/55934827
- ✓ https://yq.aliyun.com/articles/647439

THANKS