



腾讯社交广告  
Tencent Social Ads

The Power to  
Connect Businesses and People  
赋能商业 | 始终于人

# 对2000多亿条数据做一次 Group By需要多久

易杰@腾讯



促进软件开发领域知识与创新的传播



关注InfoQ官方信息  
及时获取QCon软件开发者  
大会演讲视频信息



扫码，获取限时优惠



全球架构师峰会 2017 [深圳站]

2017年7月7-8日 深圳·华侨城洲际酒店

咨询热线: 010-89880682



全球软件开发大会 [上海站]

2017年10月19-21日

咨询热线: 010-64738142



## 关于我

06年加入腾讯  
现负责社交广告引擎研发  
关注高性能架构





# Contents

- 章节1 业务背景
- 章节2 系统架构
- 章节3 核心实现
- 章节4 性能数据
- 章节5 总结

# 腾讯社交广告



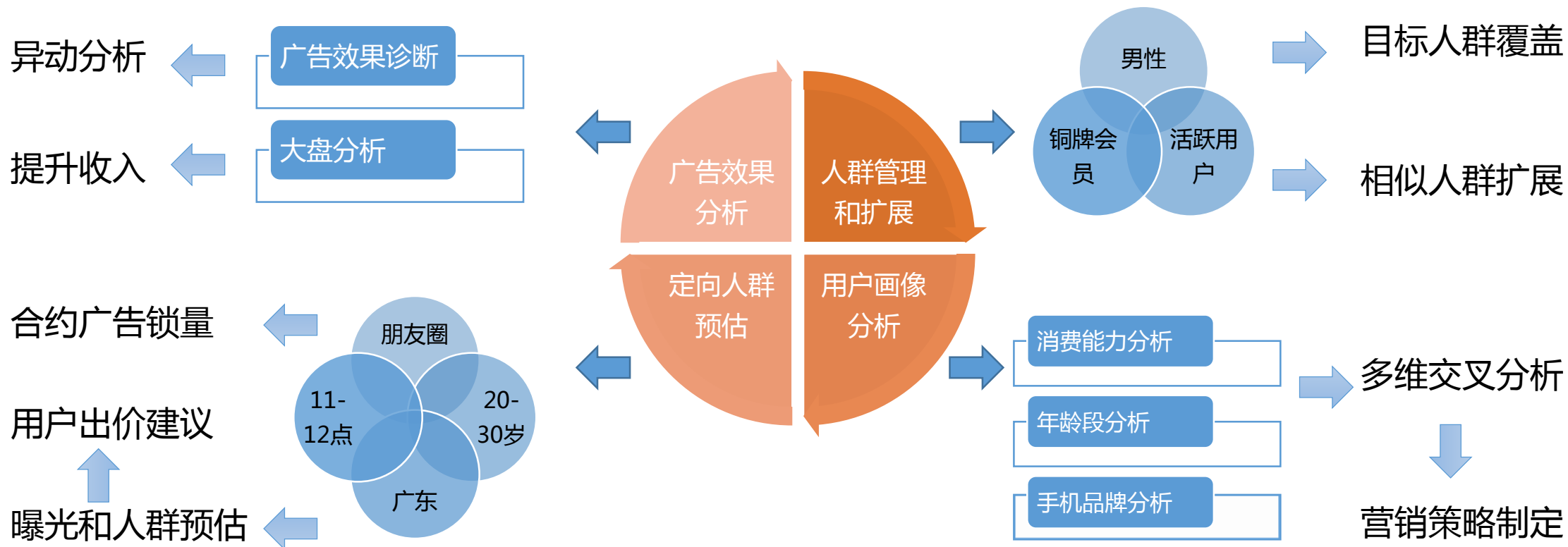
- 覆盖8亿优质用户



- 精准的定向能力



# 多维数据分析场景



# 多维数据分析场景



- 多维度人群下钻分析
- 相似人群扩展
- 时延<100ms



京腾魔方

自主上传

人群管理

已有人群

类型: 全部 用户包人群 ( 5 / 5 ) 标签人群 ( 67 / 80 ) 扩展人群 ( 4 / 5 )

状态: 全部 处理中 已生效

编号	名称	描述	量级	状态	已绑定用户群
4699	aaaa	fdsafd	4,346,000	已生效	0
4229	333333	333333333	205,000	已生效	0
4225	2222	2222	106,000	已生效	0
4057	1111	1111	166,000	已生效	0
4055	预体46000-L	test-扩展	1,010,000	已生效	0

扩展人群

类型: 全部 用户包人群 ( 5 / 5 ) 标签人群 ( 67 / 80 ) 扩展人群 ( 4 / 5 )

状态: 全部

编号

人群名称: aaaa-L 3/15

种子人数: 4,346,000 人

扩展倍数: 2 倍

扩展人数: 8,692,000 / 30,000,000

提示: 1.您最少扩展100万人,最多扩展3000万人;  
2.人群扩展预计需要1天时间;  
3.扩展后的人群只包含新客,不包含种子人群;

取消 确定



- 广告主查询用户年龄的分布

```
select age, count(*) from log where advertiser_id=123 group by age;
```

- 运营查询不同曝光次数的用户的占比、点击率、收入等

```
SELECT exposure_num, COUNT(*) as user_num,  
       SUM(sum_click) / SUM(exposure_num) as click_rate, SUM(sum_cost) AS total_cost  
FROM  
  (SELECT qq, COUNT(*) AS exposure_num,  
         SUM(click_count) AS sum_click, SUM(cost) AS sum_cost  
   FROM log  
   GROUP BY qq) temp_table  
GROUP BY exposure_num;
```



# 系统目标



流程描述：

原始数据集



过滤 Where



分组 Group by



聚合 Sum|Count|...



结果

## 高性能

- 千亿规模原始数据集
- (毫)秒级端到端响应

## 低成本

- 索引规模相对原始数据集膨胀可控
- 利用SSD磁盘，降低内存使用

## 可扩展

- 增量数据修改
- 接口易用，支持SQL/RPC

业界实现：SQL-on-Hadoop(Hive/Dremel/Kylin/Drill)、Druid

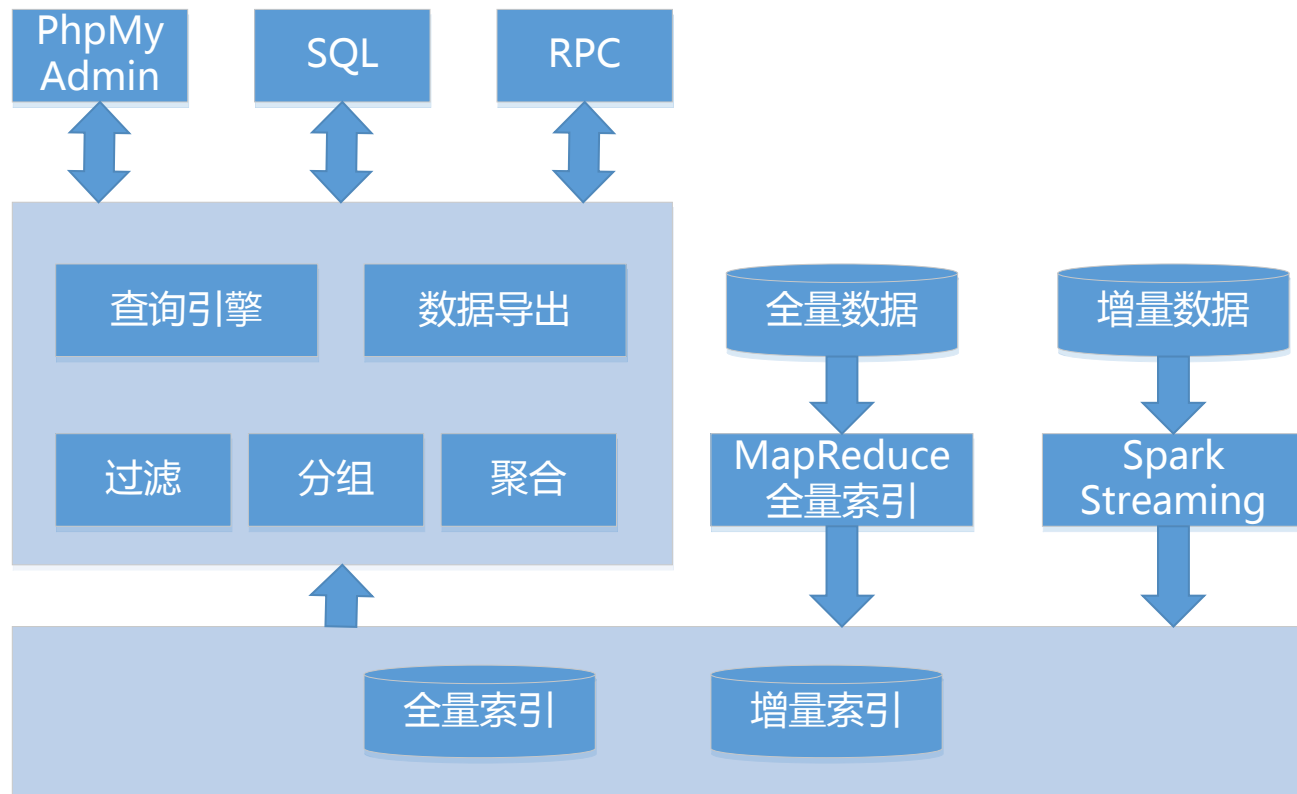
自研：Pivot



# Contents

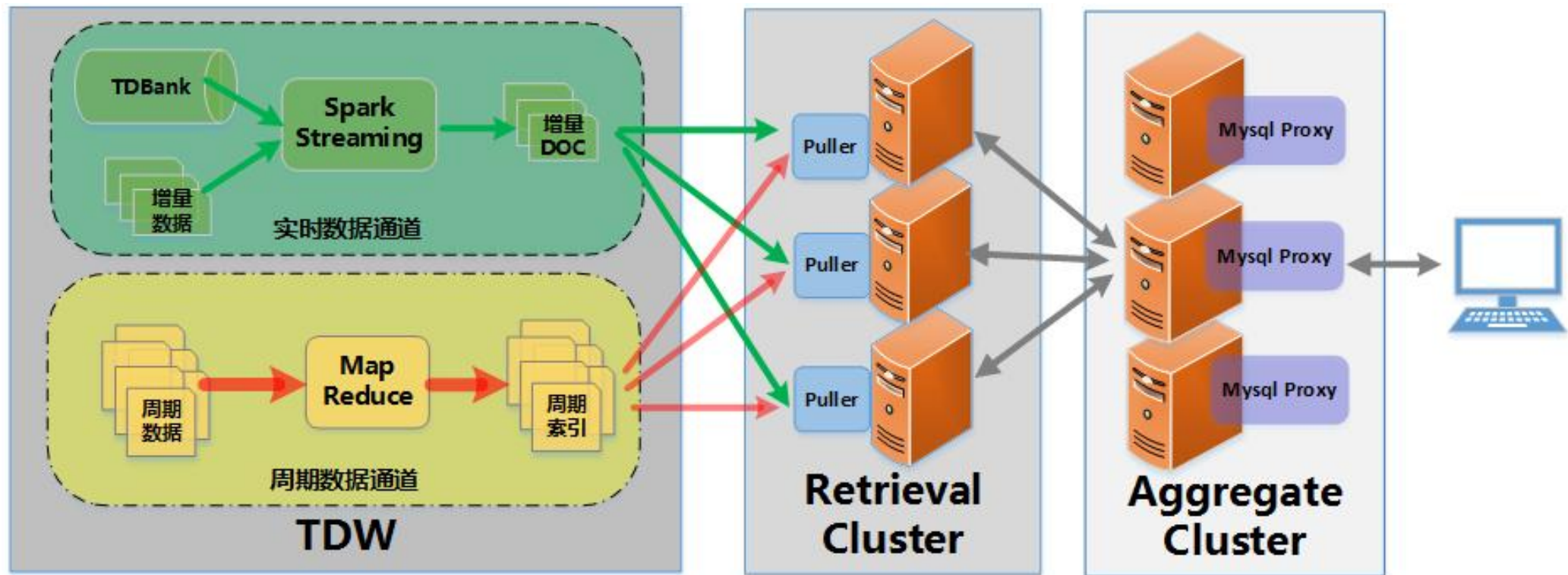
- 章节1 业务背景
- 章节2 系统架构
- 章节3 核心实现
- 章节4 性能数据
- 章节5 总结

# 系统架构



- 全量+增量，满足多种需求
- 索引分片，多级聚合
- 标准SQL接口，降低使用门槛
- 唯快不破

# Lambda架构





# Contents

- 章节1 业务背景
- 章节2 系统架构
- 章节3 核心实现
- 章节4 性能数据
- 章节5 总结


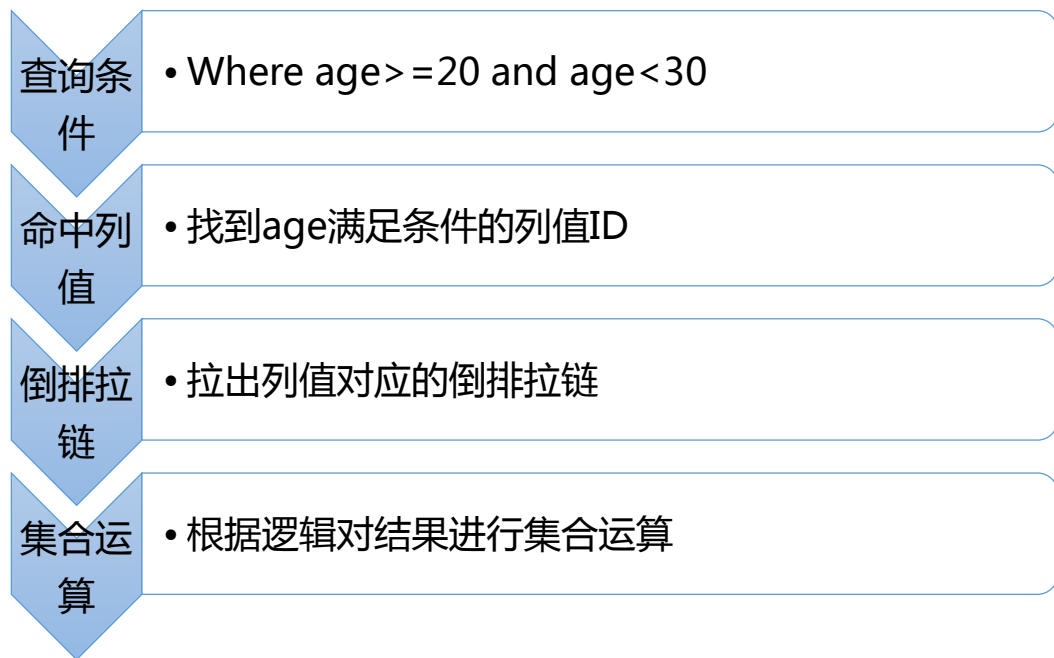
# 索引文件设计



索引数据结构：

全局信息	列值字典	倒排数据	正排数据
	每一列(Column)的值 编码为列值(Term) ID	列值ID对应的文档ID 列表	列式存储的压缩数据

检索查询流程：

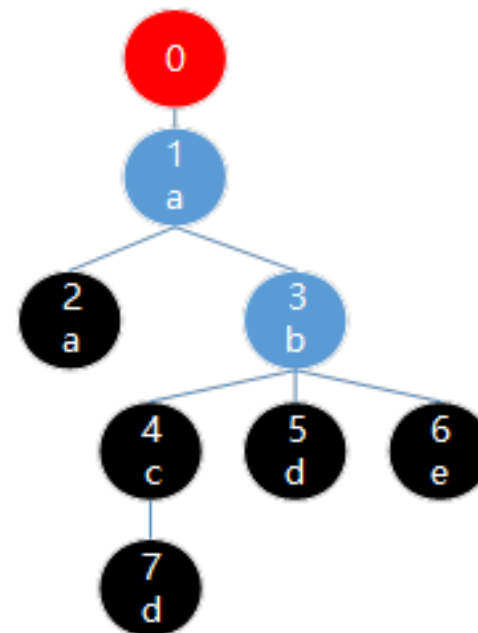
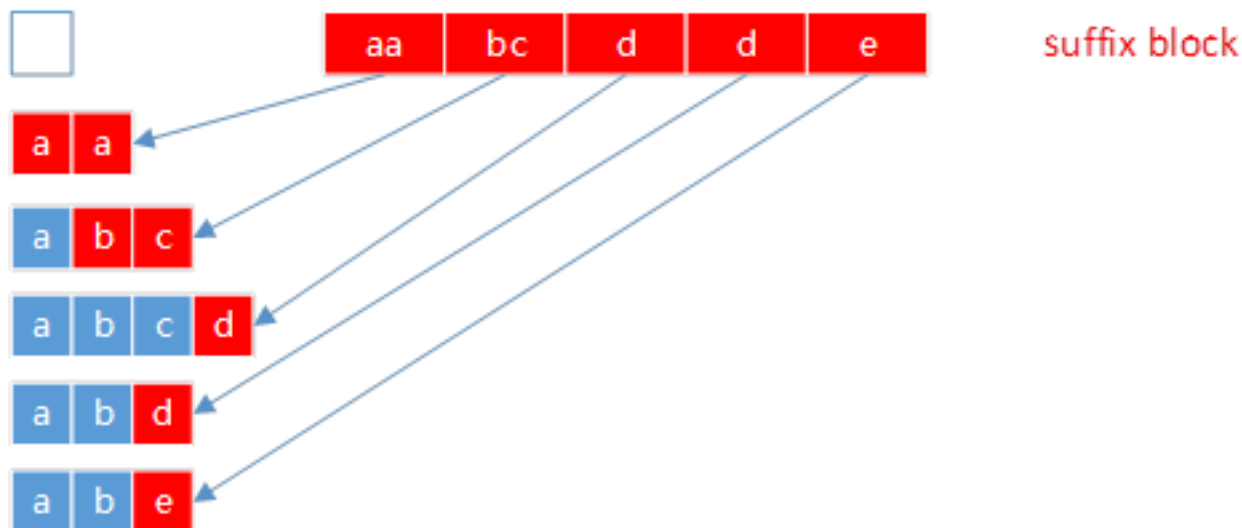


	Column 0	Column 1	Column 2	...
Doc 0	term list	term list	term list	term list
Doc 1	term list	term list	term list	term list
...	term list	term list	term list	term list

# 列值字典String压缩



- 前缀压缩节省空间
  - 例如词表：{ "aa", "abc", "abcd", "abd", "abe" }
- 实现采用更高效的Vector前缀压缩
  - 词ID快速定位



# 倒排数据存储



- 倒排 – 列值对应的文档ID列表
- 位图 ( Bitmap ) 压缩存放倒排拉链列表
  - RoaringBitmap支持AND、OR等运算，线性性能
  - 根据元素个数动态选择有序数组或bitmap存储
  - 基于RoaringBitmap源码做了存储和性能的优化

Roaring与其他位图算法的性能比较：

(a) Size expansion if Roaring is replaced with other schemes.

	CENSUS1881	CENSUSINCOME	WIKILEAKS	WEATHER
Concise	2.2	1.4	0.79	1.4
WAH	2.4	1.6	0.79	1.5
BitSet	42	2.9	55	3.5

(b) Time increase, for AND, if Roaring is replaced with other schemes.

	CENSUS1881	CENSUSINCOME	WIKILEAKS	WEATHER
Concise	920	6.6	8.3	6.3
WAH	840	5.9	8.2	5.4
BitSet	730	0.42	28	0.64

(c) Time increases, for OR, if Roaring is replaced with other schemes.

	CENSUS1881	CENSUSINCOME	WIKILEAKS	WEATHER
Concise	34	5.4	2.1	3.9
WAH	31	4.9	2.1	3.4
BitSet	29	0.43	6.7	0.48

- Roaring is 20 to 100 times faster than Concise on unions
- Roaring is 40 to 200 times faster than Concise on intersections
- Roaring can scan the set bits up to 3 times faster than Concise
- Roaring can offer better compression (up to 1.6x better than Concise)
- Roaring never uses more than 300 bytes per bitmap

Roaring性能最优



# 正排数据存储



- 60%的索引空间是正排数据
- 减少磁盘IO访问-列存储
- 最大程度节省磁盘空间-编码压缩



筛选11种编码算法

定长类型编码：定长数组编码、列值个数索引编码、定长列表编码、变长列表编码

String类型编码：单string长度索引编码、多string长度索引编码、单string列表编码、多string列表编码

简单字典编码：适用列值重复较多的string类型

Huffman字典编码：列值分布不均匀情况下对简单字典编码的优化

二进制String编码：较特殊的二进制数据类型

# 正排数据压缩编码



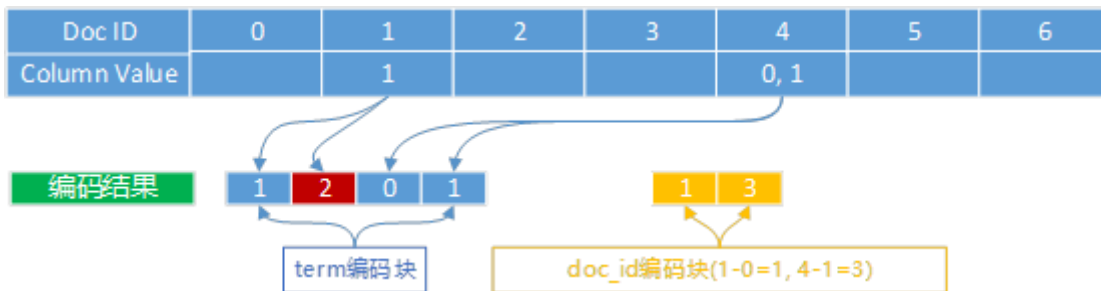
- 定长数组编码

- 适用列值取值平均个数约等于最大值，譬如年龄
- 数值压缩存放



- 定长列表编码

- 适用列值稀疏情形
- 文档ID差值压缩存放



# 分组 ( Group By ) 实现



- 基于排序

- 代价高,  $O(n \cdot \log(n))$
- 数据库Group by未命中索引时

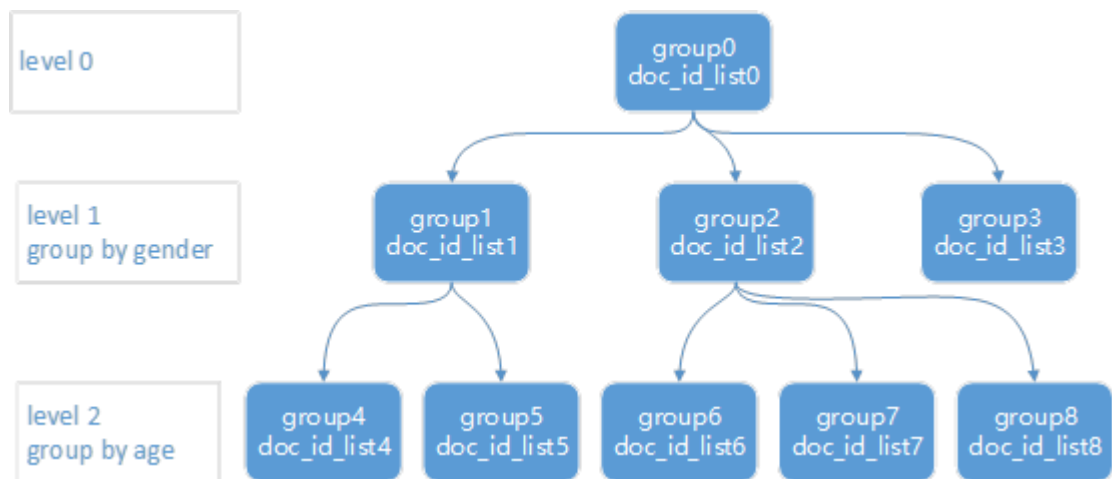
- 基于索引

- 同一组的数据在此索引(有序) 连续排列
- 数据库Group by有序命中索引时

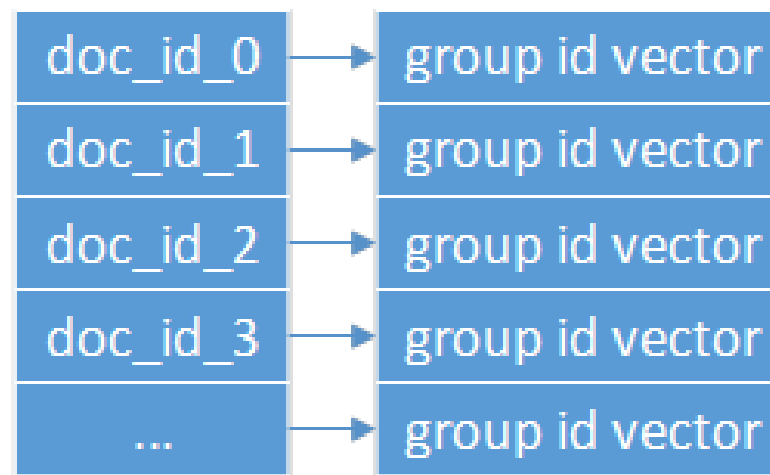
# 遍历正排数据实现分组



- 按照列的顺序逐列分组
  - 当列值作为key，map规模可控
- 按照树形结构进行分组
- 一个文档可能被分到多个组



分层Group By

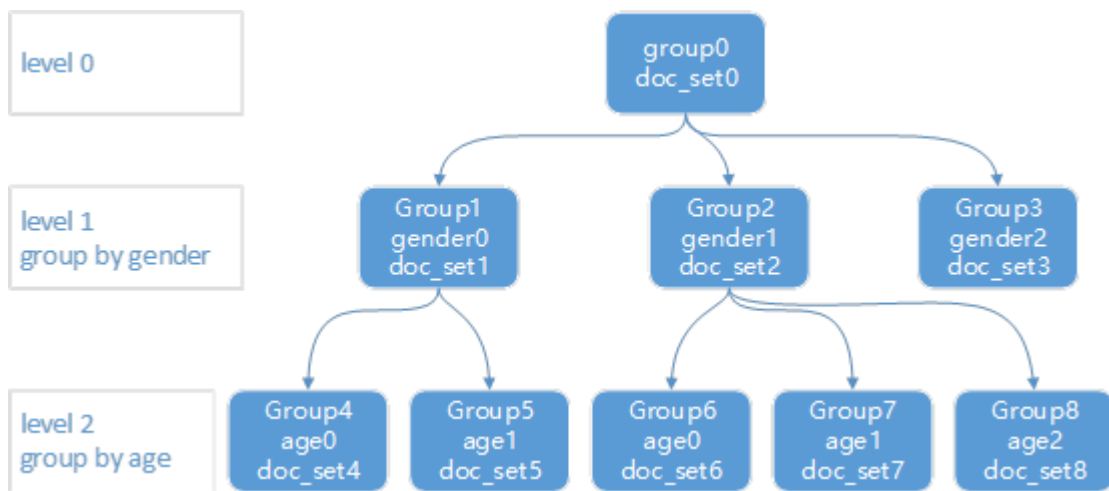


Doc Id到group Id映射

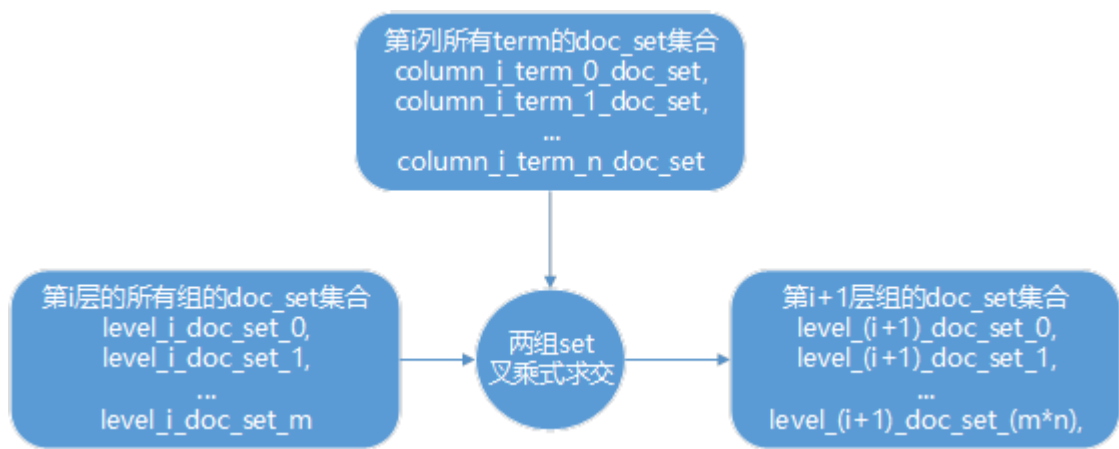
# 基于倒排数据实现分组



- 集合求交更快
- 适用分组数较少时



倒排分层Group By

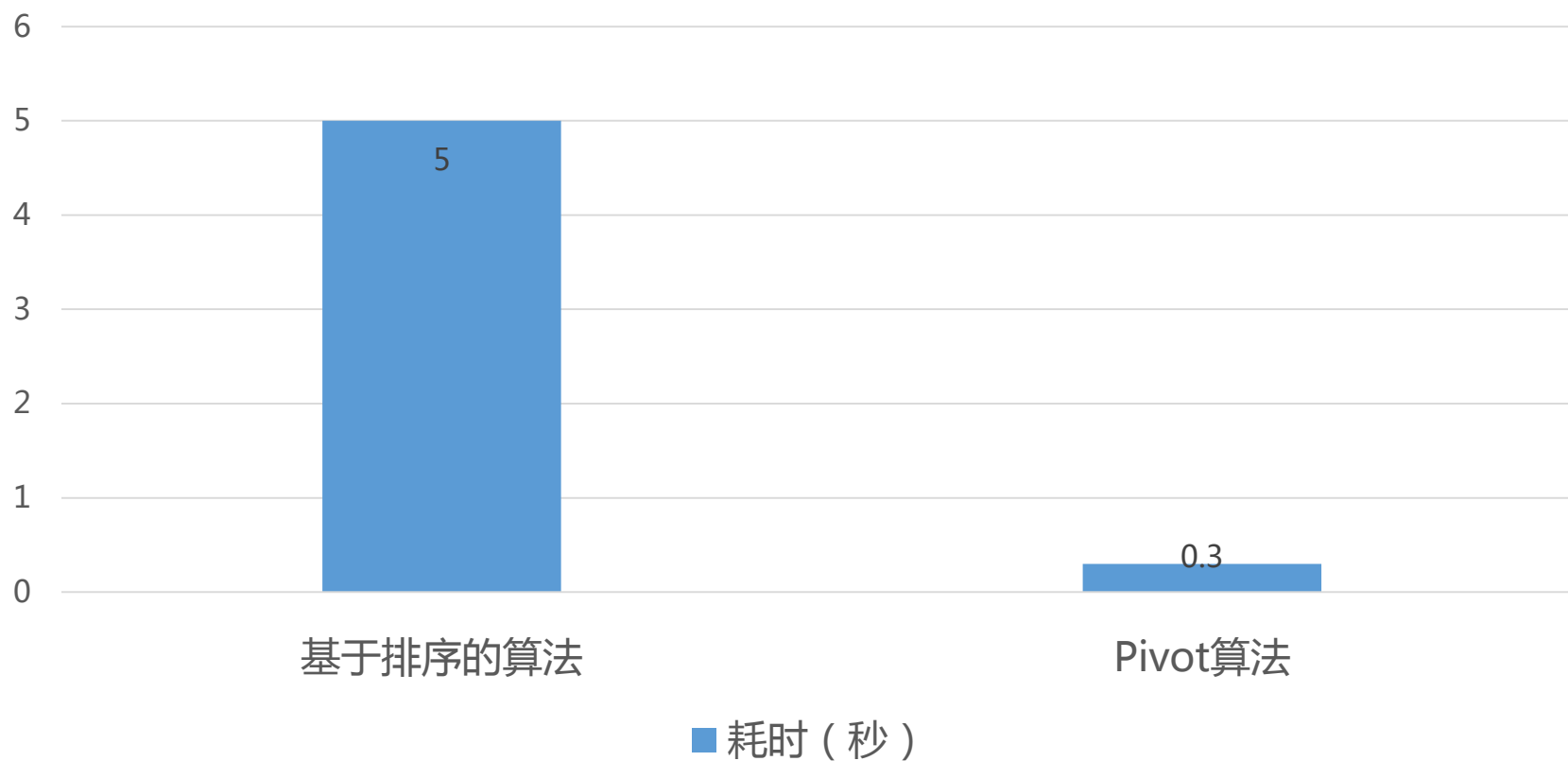


集合求交

# 分组性能对比



- 6000万条数据，按照性别、年龄两列分组查询



# 求和实现算法 ( SUM )



- Group By后对各个组的某列进行SUM求和
- 借鉴Group by算法，实现了两种SUM算法

## 基于倒排集合求交

- ⑩累加变乘法，操作结果集数量级优化
- ⑩适用结果集合总数较少时

## 基于列值字典遍历正排

- ⑩方法和Group By类似
- ⑩适用结果集合总数很大时



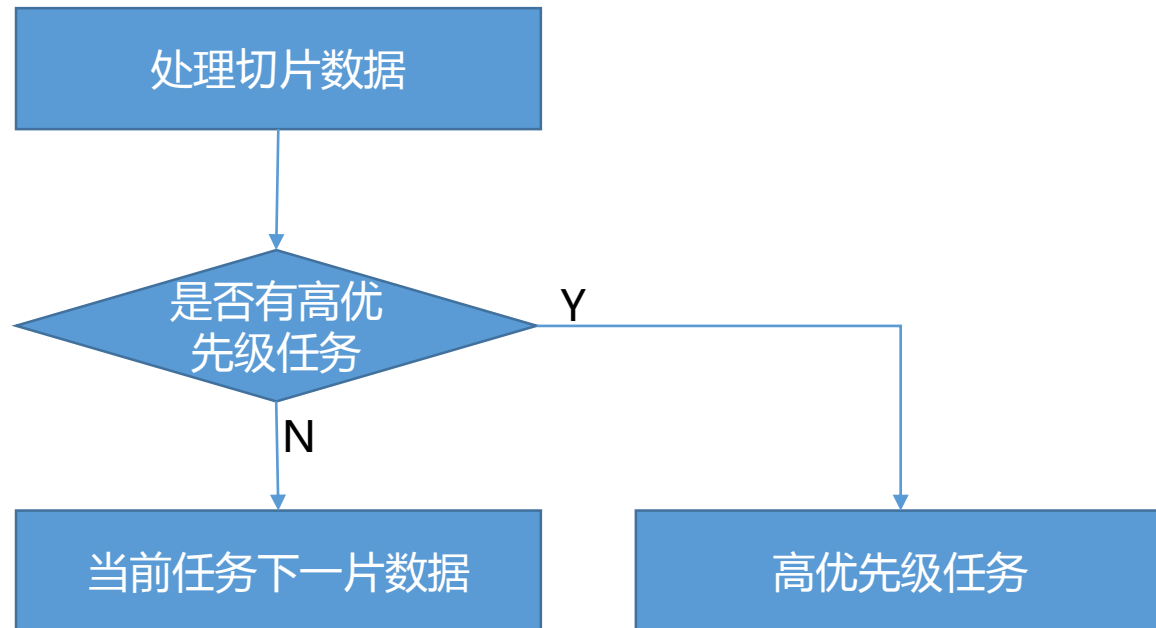
- 数据存储与管理
  - 方便对“过期”数据清理
- 提升查询性能
  - 大部分请求并不需要检索全量数据
  - 缩小数据扫描范围
- 关键字Partition
  - `select count(*) from log partition('20170201','20170202');`



# 优先级调度



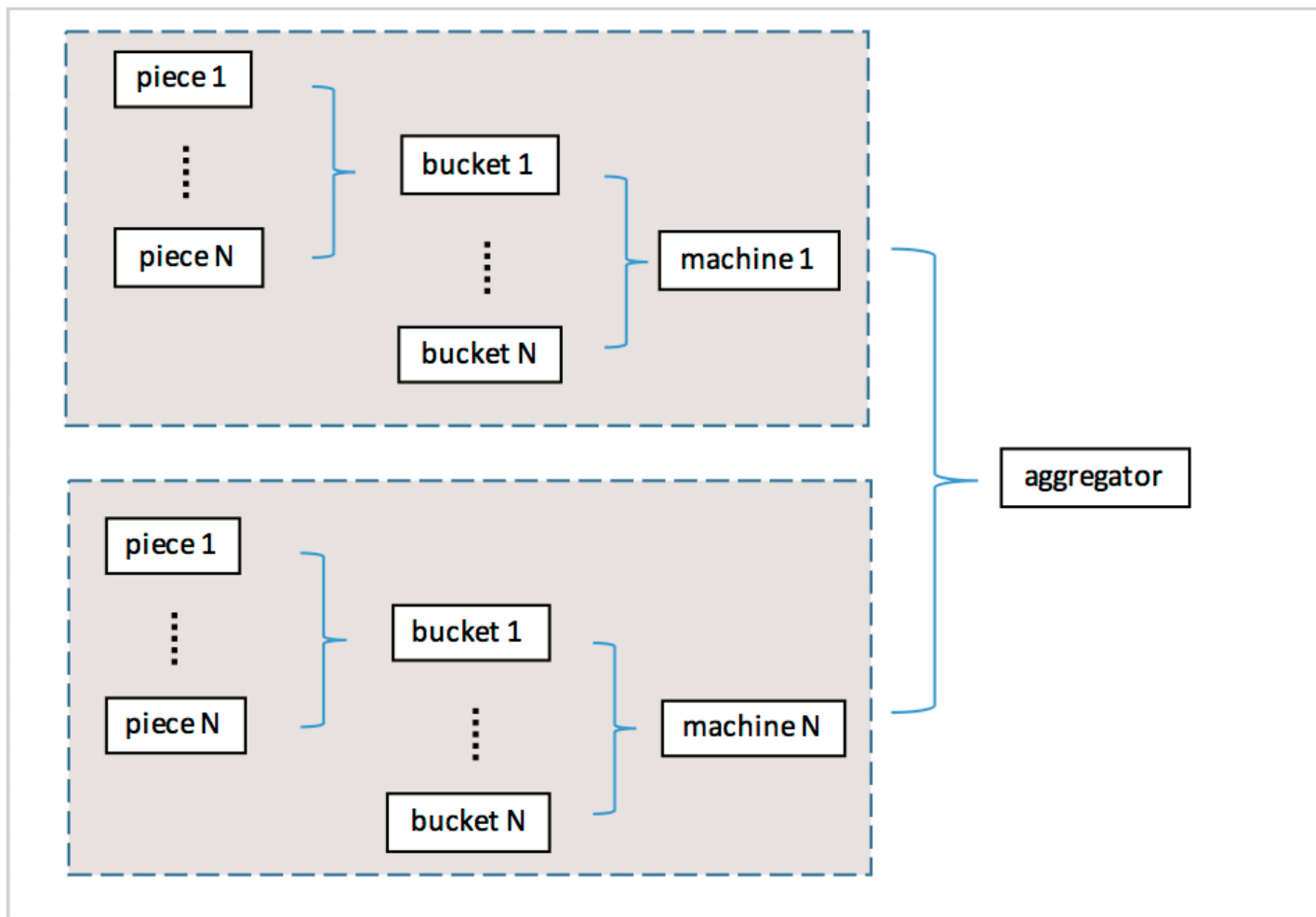
- 不抢占
  - 先到先处理
- 任务级优先调度
  - 先处理优先级高的任务
- 分片级优先调度
  - 更细粒度的调度



# 插件设计



- 开放Functor

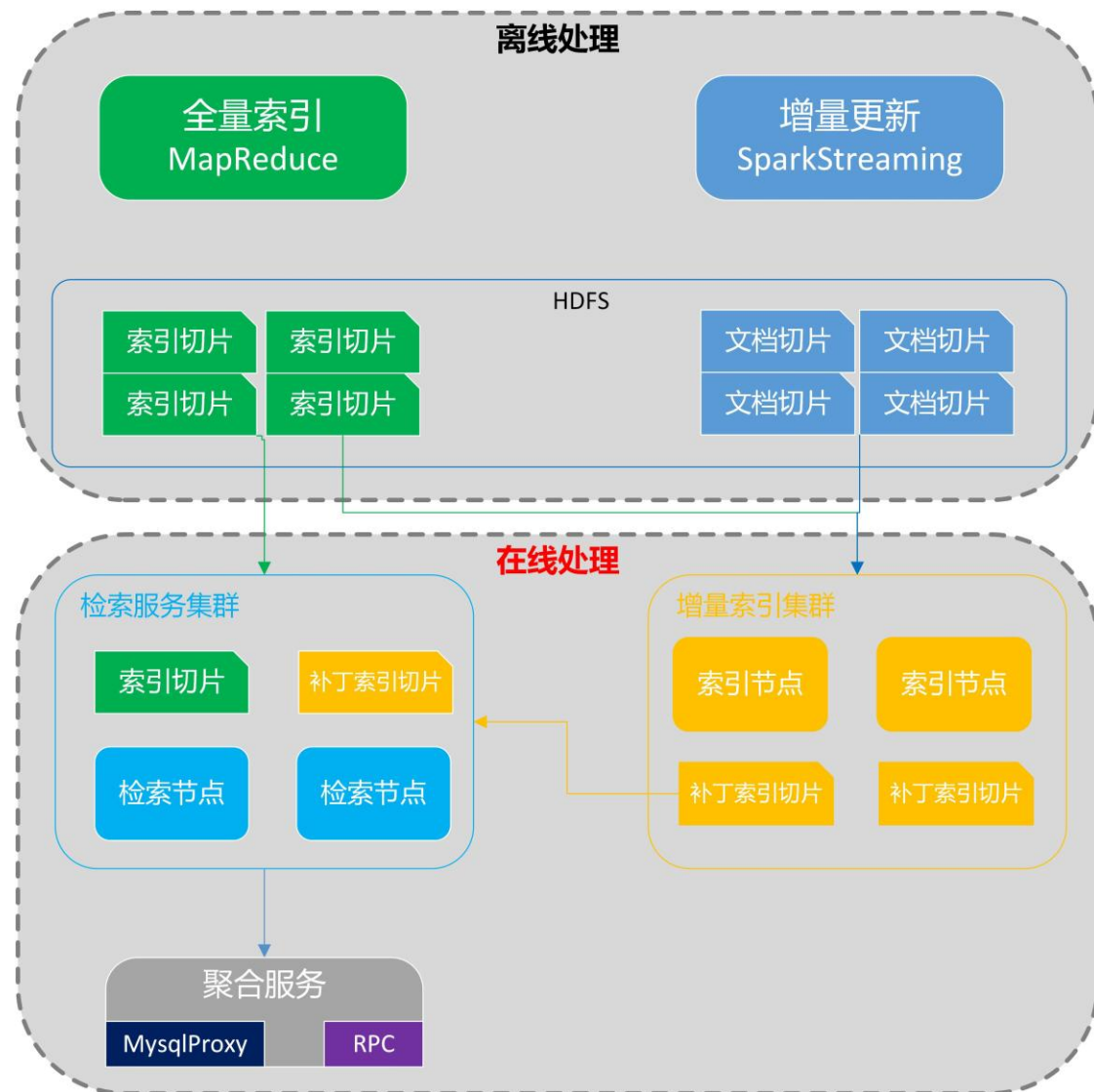
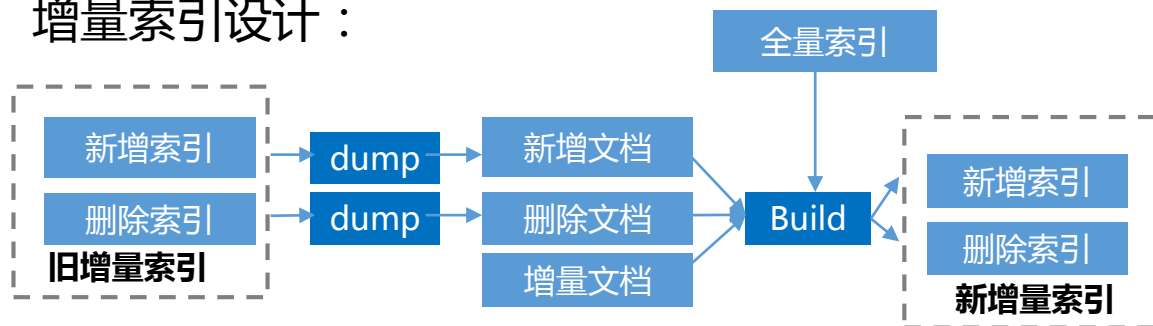


# 索引更新设计

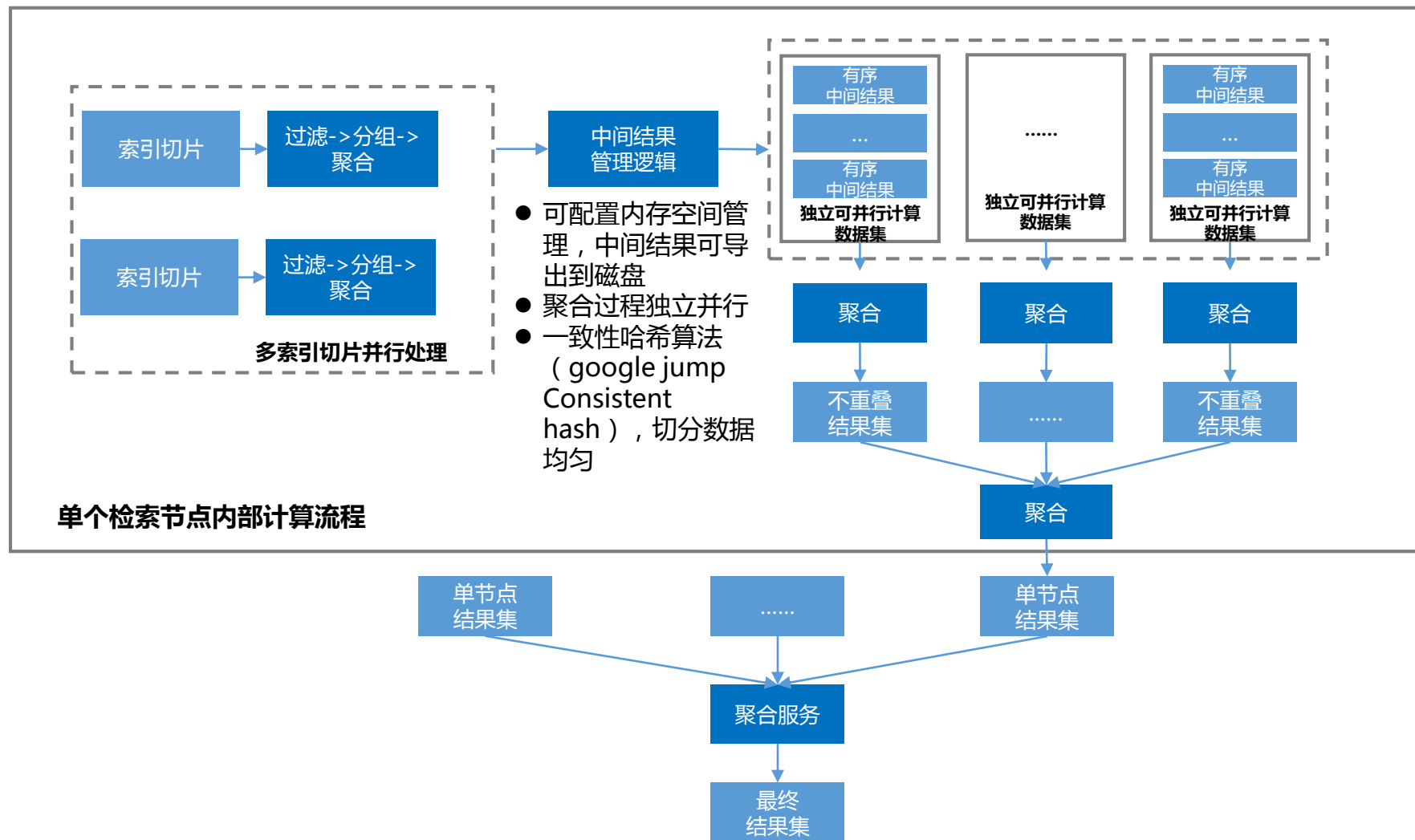


- 公司TDW集群管理索引，可靠性高
- 支持增量更新
  - 譬如用户的行为（标签）会随时变更
- 打补丁更新方式，记录新增和删除的增量索引
- 增量索引和检索独立部署 – 避免IO干扰

增量索引设计：



# 并行计算设计





# Contents

- 章节1 业务背景
- 章节2 系统架构
- 章节3 核心实现
- 章节4 性能数据
- 章节5 总结



- Pivot VS Druid : Druid索引小 , Pivot查询快数量级

测试项 ( 机型TS8 : CPU 24核 , 内存32G , 硬盘290*11 )	druid	pivot
数据源记录数 ( Druid单机上试图导入更多数据时 , 构建失败 )	6000万	6000万
索引大小	7.2G	12G
select count(*) from log	10ms	2ms
select producttype, count(*) from log group by producttype	4.7s	57ms
select gender, count(*) from log group by gender	4.7s	27ms
select age, count(*) from log group by age;	4.9s	102ms
select gender, age, count(*) from log group by gender, age	7.3s	325ms
select gender, sum(impression_count) from log group by gender	4.7s	51ms
select producttype, sum(impression_count) from log group by producttype	5.0s	91ms
select count(distinct(qq)) from log	12.5s	1.85s

# 性能数据



```
mysql> select count(*) from log;
```

```
+-----+  
| COUNT(*) |  
+-----+  
| 249243862771 |  
+-----+
```

```
1 row in set (0.03 sec)
```

```
mysql> select new_gender, count(*) from log group by new_gender;
```

```
+-----+-----+  
| new_gender | COUNT(*) |  
+-----+-----+  
| 0 | 23752244705 |  
| 1 | 125602742040 |  
| 2 | 99888876026 |  
+-----+-----+
```

```
3 rows in set (2.27 sec)
```

# 性能数据



```
mysql> select count(*) from log where advertiser_id=3665472;
```

```
+-----+  
| COUNT(*) |  
+-----+  
| 79174926 |  
+-----+
```

```
1 row in set (0.07 sec)
```

```
mysql> select new_gender, count(*) from log where advertiser_id=3665472 group by new_gender;
```

```
+-----+-----+  
| new_gender | COUNT(*) |  
+-----+-----+  
|          0 |    980955 |  
|          1 |   40963790 |  
|          2 |   37230181 |  
+-----+-----+
```

```
3 rows in set (1.82 sec)
```



# 性能数据



```
mysql> select new_gender, count(*) from log group by new_gender partition('20170411');
+-----+-----+
| new_gender | COUNT(*) |
+-----+-----+
|          0 | 419041045 |
|          1 | 2391438901 |
|          2 | 1907457933 |
+-----+-----+
3 rows in set (0.37 sec)
```



# Contents

- 章节1 业务背景
- 章节2 系统架构
- 章节3 核心实现
- 章节4 性能数据
- 章节5 总结

# 总结



- 进一步完善SQL
- 资源管理



腾讯社交广告  
Tencent Social Ads

The Power to  
Connect Businesses and People  
赋能商业 | 始终于人

