如何突破腾讯大数据分析架构瓶颈（发布版）

—彭渊 腾讯T4专家

典型业务场景和系统架构瓶颈（略）

实时计算框架选型问题：

当时团队现状：storm和spark streaming研发支持和运维能力较成熟，原始flink还需要优化，尚不具备直接大规模工程化实施的条件。

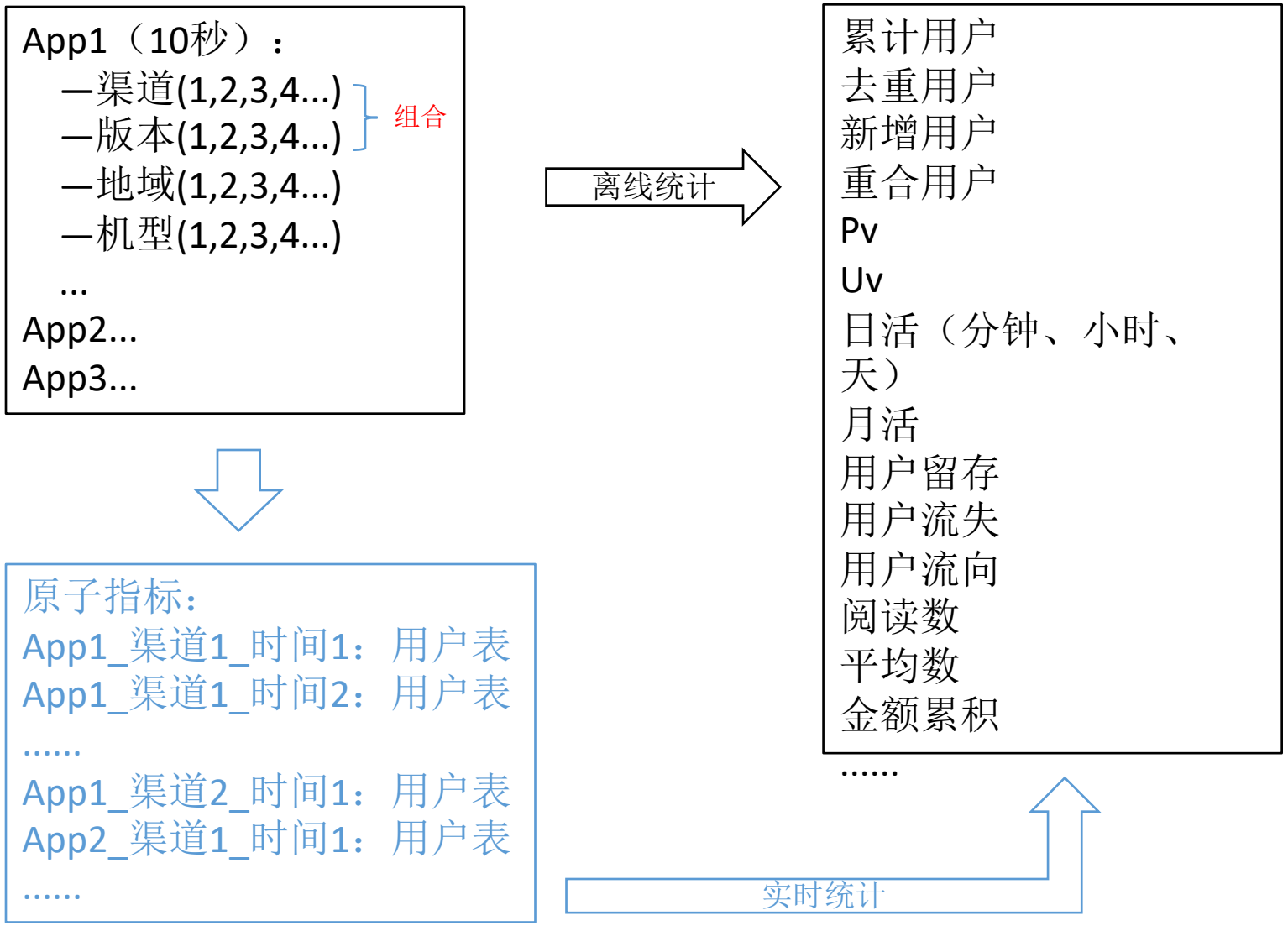
1、storm的算子封装和开发成本较大，但是能很好解决1分钟到1小时实时计算和离线资源释放，sql支持弱

2、 spark streaming开发成本低，能解决5分钟10分钟实时批量计算，但是1小时计算无法释放离线资源，很多小分钟任务的调度成本

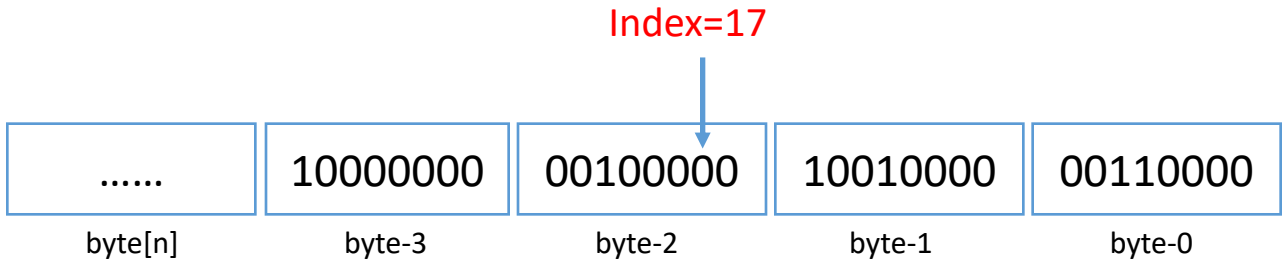
3、flink提供流式和批量结合的实时处理和完善sql支持，适合完成清洗和count计算，但其有限时间窗口并不太适合大批量用户去重和统计

流处理+bitmap的实时解决方案

业务场景


App1（10秒）：
—渠道(1,2,3,4...)
—版本(1,2,3,4...)
组合
—地域(1,2,3,4...)
—机型(1,2,3,4...)
...
App2...
App3...
离线统计
累计用户
去重用户
新增用户
重合用户
Pv
Uv
日活（分钟、小时、天）
月活
用户留存
用户流失
用户流向
阅读数
平均数
金额累积
.....
原子指标：
App1_渠道1_时间1：用户表
App1_渠道1_时间2：用户表
.....
App1_渠道2_时间1：用户表
App2_渠道1_时间1：用户表
.....
实时统计

BitMap位存储和位计算

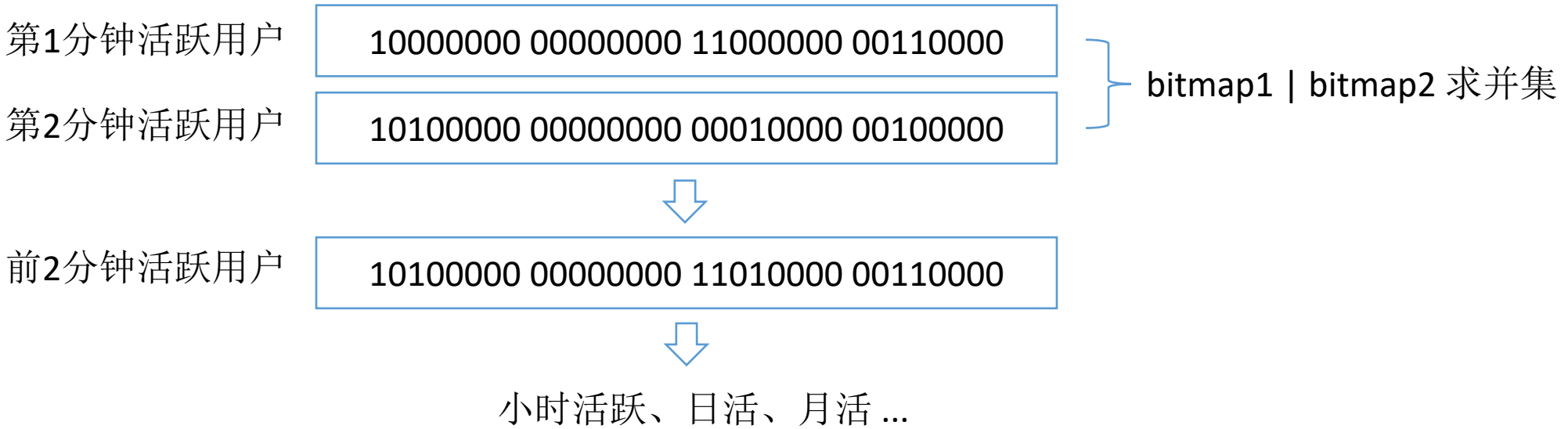


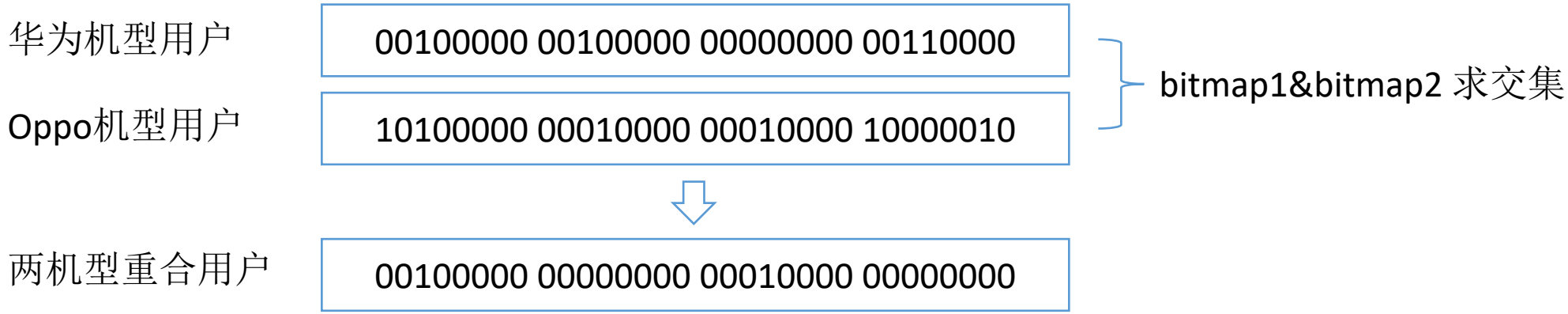
使用bitmap有两个非常显著的优势：位存储占用空间低，位计算效率高。

1、将需要做统计计算的id转换成数字序号，每个只占1个bit，对于20亿的用户id，只需要20亿bit约238m大小，压缩后占用空间更小，最少只要200k；

2、通过单个bitmap可以完成去重操作，通过多个bitmap的且、或、异或、反等位操作可以完成日活、月活、小时分钟活跃、重度用户、新增用户、用户流向等绝大部分的统计计算，而且能在单机毫秒级完成，真正做到实时计算出结果，同比hadoop/hive离线计算执行“select distinct count…from…groupby join…”类似sql的方式统计，往往需要几百台机器，耗用30分钟才能完成，对比非常悬殊，而且容易形成大量sql任务调度和大表join给集群带来繁重压力。

BitMap聚合计算


第1分钟活跃用户
10000000 00000000 11000000 00110000
第2分钟活跃用户
10100000 00000000 00010000 00100000
bitmap1 | bitmap2 求并集
前2分钟活跃用户
10100000 00000000 11010000 00110000
小时活跃、日活、月活 ...


华为机型用户
00100000 00100000 00000000 00110000
Oppo机型用户
10100000 00010000 00010000 10000010
bitmap1&bitmap2 求交集
两机型重合用户
00100000 00000000 00010000 00000000

BitMap聚合计算

1、去重用户：求1的总数
2、活跃用户：取或bitmap1 | bitmap2
3、非活跃用户：取反：～bitmap1
4、重度用户：取且：Bitmap1 & bitmap2
5、新增用户：取或加异或：(Bitmap1 | bitmap2)^bitmap1
6、流失用户： Bitmap1 相对于bitmap2的新增用户
7、用户流向：app1time1的流失用户&app2time2
8、多种指标组合：Bitmap1 & bitmap2 & bitmap3 &…
等等

BitMap聚合计算

		u0	u1	u2	u3	u4	u5	u6	u7
机型	hw	1	0	0	0	0	1	0	0
	oppo	0	1	1	1	0	1	1	0
	vivo	1	1	0	0	1	0	0	1
	mi	0	0	0	0	0	0	0	1
城市	BJ	1	1	1	0	0	0	1	0
	SH	1	0	0	0	1	0	0	0
	SZ	0	0	1	0	0	1	1	0
	GZ	0	1	1	0	1	0	0	0
	kashi	0	0	0	1	0	0	0	0
聚合	hw_BJ	1	0	0	0	0	0	0	0
	mi_kashi	0	0	0	0	0	0	0	0

产品指标

用户画像

字段A(共10个取值)和字段B(共10个取值)，如果聚合AB，有两种方式：

第1种，An&Bn，共10+10=20个bitmap；

第2种，An_Bn，共10*10=100个bitmap。

第1种存在数据叠加的影响，第2种不存在；

第1种消耗更少的空间。第2种消耗更多空间，但是根据业务实际数据出现建立bitmap，实际占用空间小于理论组合全部值（比如mi_kashi不存在）。

BitMap压缩分析

数据大小：16000000字节，连续压缩10次，测试结果如下：

压缩方式	压缩时间（毫秒）	解压时间（毫秒）	压缩后大小（字节）
jdk deflate LEVEL1/9	1256/65093	635/631	2979891/1995271
xz 1.5	151347	2684	1388960
jdk gzip	7246	651	2143725
common1.11 bzip2	20370	5226	1785490
common1.11 deflate	3355	636	2193916
common1.11 gzip	7126	523	2143725
common1.11 xz	154148	2463	1388960
lz4 1.3.0	471	314	5112117
snappy 1.1.2.4	571	200	3988367

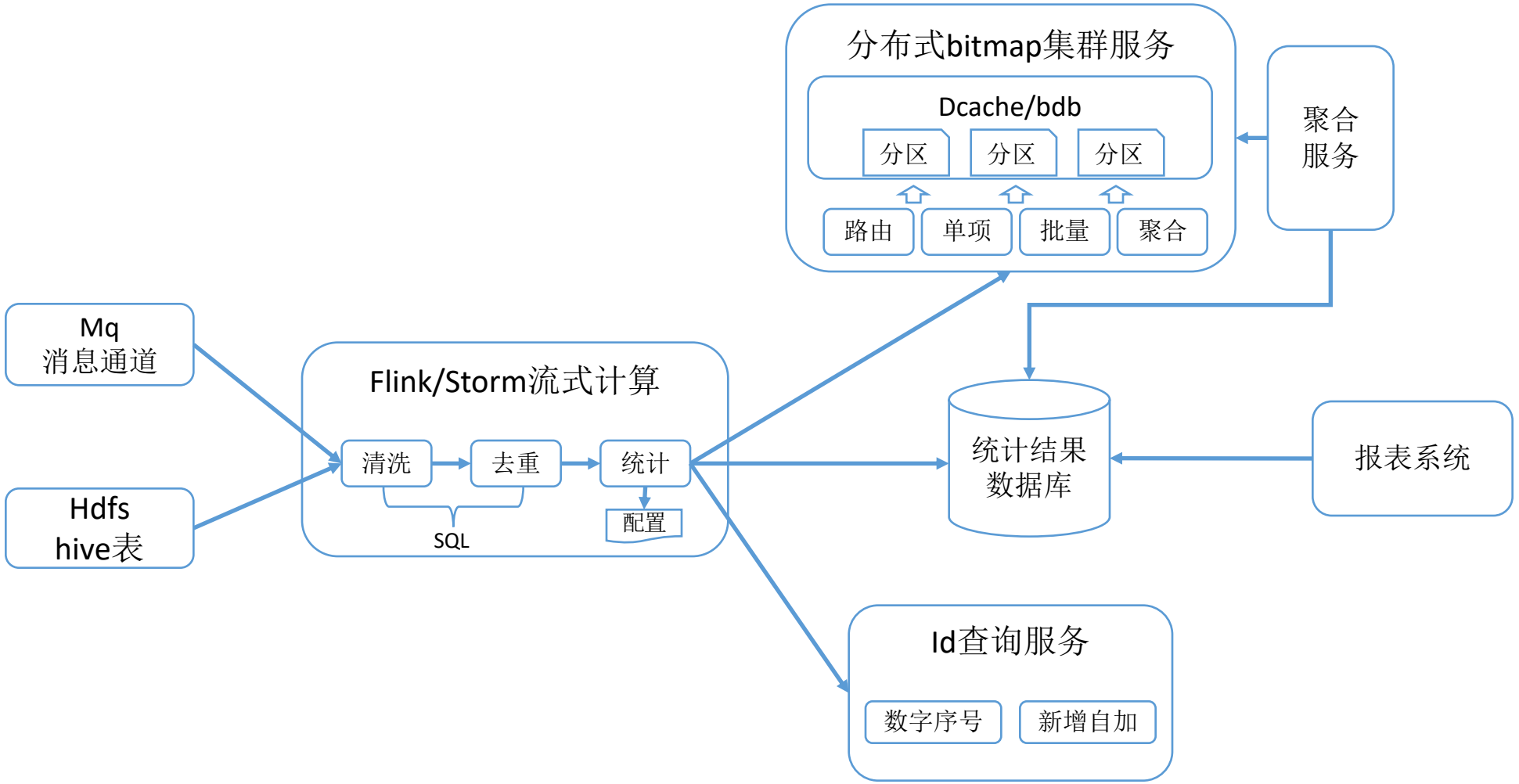
1、对于bitmap数量很多的场景，压缩有利于节省大量空间，对于20亿的用户id，需要20亿bit约238m大小，压缩后占用空间最少只要200k。

2、按照灯塔当前业务流量，42亿范围的id不会全部来，去重后1-2亿进入bitmap，按照2:42的数据分布，压缩后还是能省很多空间。

3、如果业务流量单位时间内（10秒），42亿用户全部来或者来30多亿，这时bitmap大部分为1，压缩率反而很高，空间耗用不大。反而，对于只来10-20亿去重用户，这是要面临极端最坏的情况，数据分布广且稀疏，压缩有限，空间耗用很大。

4、不同的压缩算法压缩率和耗时成反比，考虑实时性和空间节省，选用压缩率和耗时比较平衡的gzip压缩。

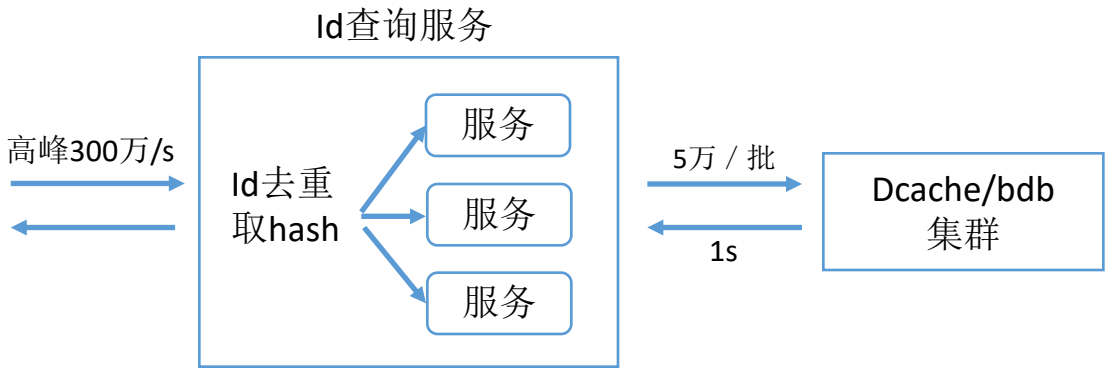
流式处理+BitMap实时计算架构



Id查询服务

初始化：通过离线任务，按照最后活跃时间初始化用户id的数字序号，很久没来的用户放前面，最近的用户放后面，新增的用户在后面加1，这样直接从bitmap的数字ID范围知道是大致什么时间的用户，目前约50亿范围的用户id，除去虚假用户和僵尸用户，还有20亿左右正常用户。**结果：仅通过数字序号可以区别最近的用户还是很久以前的。**

运行时：

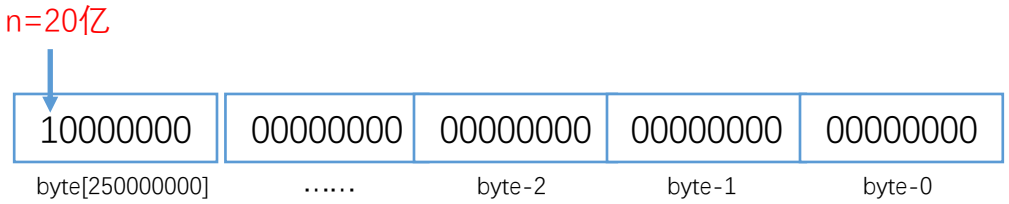


初始化20亿常用用户，其他查询新增

未来：手机设备上，灯塔sdk直接携带设备id的数字序号进入通道，逐步减少id查询

BitMap的空间耗用

1、大范围数字的空间浪费，如果不分区，一个值为20亿的数字需要耗用238m的空间；



2、分区bitmap的优势和问题，对于数字最大范围为20亿，按照不同的大小分区如下，如何做到最优的空间消耗呢

每100byte一个分区

0	800	100byte
1	800	100byte
……	800	100byte
2500000	800	100byte

每8k一个分区

0	65536	8k
1	65536	8k
……	65536	8k
30518	65536	8k

每64k一个分区

0	524288	64k
1	524288	64k
……	524288	64k
3815	524288	64k

每2m一个分区

0	16777216	2m
1	16777216	2m
……	16777216	2m
120	16777216	2m

BitMap的空间耗用

3、RoaringBitmap的核心原理和不足：

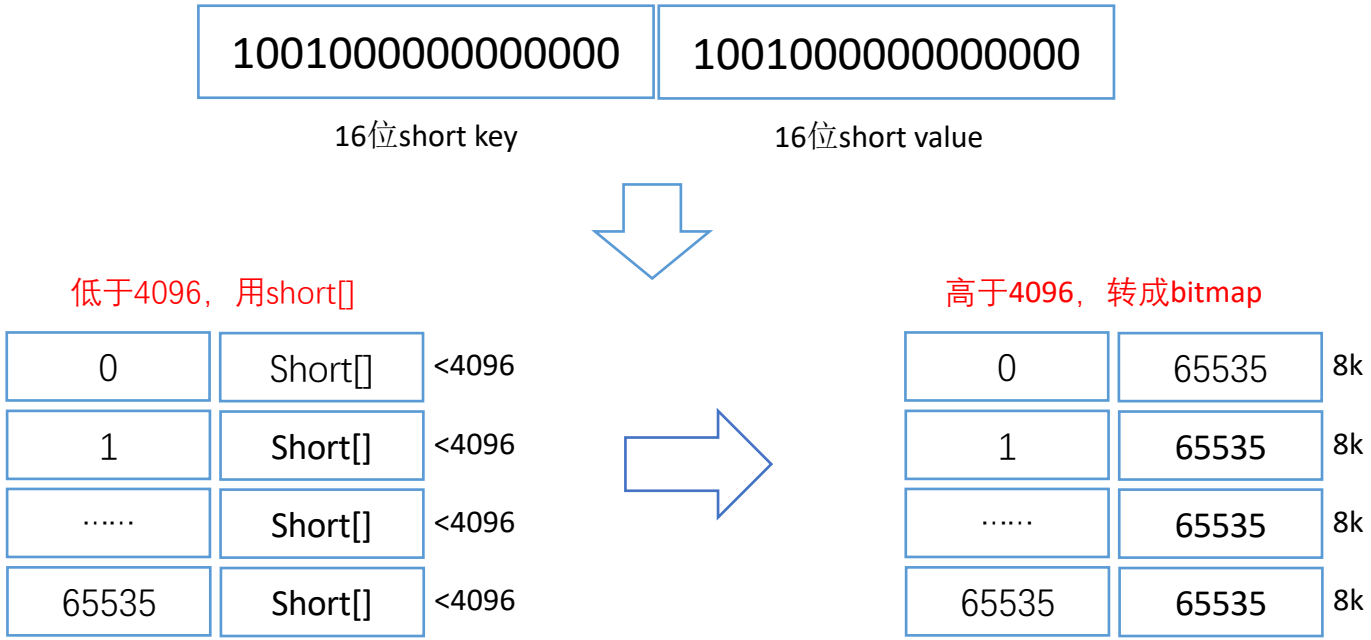
将一个整型数字拆分成高低2个短整型，并共用高位做key，可以知道最大分区为65535，低位做value，可以用一个short[][]来存储，相同key的value存储在同一个short[]里，进行二分查找去重，当value的数量到达4096时，空间消耗已经等同于一个长度为65535的bitmap，这时进行bitmap转换。 RoaringBitmap在理论上通过高低分位成两个short类型数字，从而有效节省数据量小时的空间开销，并在数据增长到临界点4096时转换成bitmap，数据量大时不再增长空间开销。

但是RoaringBitmap在落地的实现和应用中仍然通常在面临以下问题，不能做到性能和空间最优：

a、高低分位后，实际上形成一个65535*4096的固定分区，对不同的数据特点，不能灵活设置分区大小和分区数量以满足不同数据范围和不同增长幅度的场景需求；

b、在其java的实现版本里，由于short数组的动态变长，产生大量的无用对象不能及时垃圾回收（gc），而且只有当每个分区抵达4096数量时才能转为bitmap停止内存增长，容易导致在分区均匀情况下接近4096的时候空间开销很大，这时bitmap的数量很少，而65535个short[]动态增长产生的大量垃圾内存，其实际空间消耗已经远远超出理论的预估值。

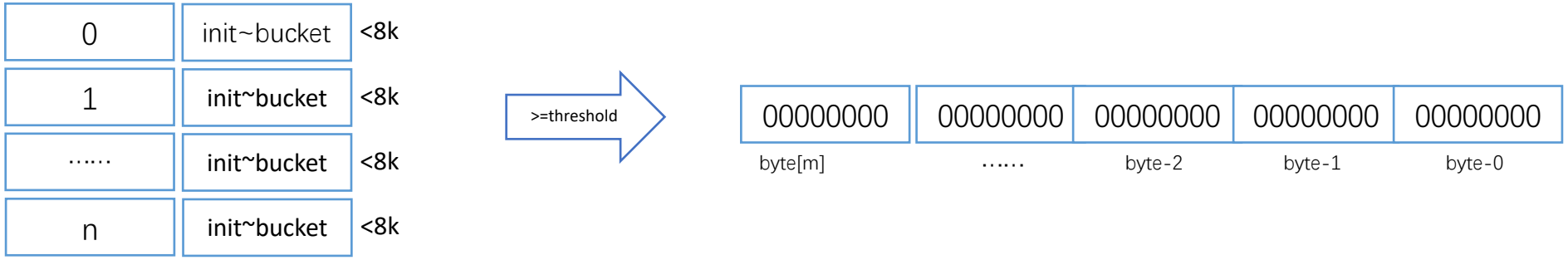
c、耗时慢，由于二分查找去重需要进行数组排序，会产生额外的性能消耗，特别是数据量低于5000万时，二分查找的性能并不比线性查找有优势，反而耗时更多。当数据量很大时，数组结构已经转成bitmap，此时二分查找已经不再起作用。



BitMap的空间耗用
4、原创实现的高效ArrayBitmap

最大范围：range
分区大小：min(bucket, 65535)
初始大小：min(init, 65535)
变长系数：auto
Bitmap阀值：threshold
分区数：n=range/bucket+(1?0)
Bitmap长度：m=range/8+(1?0)

为了克服数据量小而且稀疏时导致的bitmap空间浪费问题，经过实践摸索寻找到一种动态增长的分桶数组结合bitmap数据结构的新设计方案来解决，经过大量测试证明，相对于RoaringBitmap有更好的性能和更优的内存空间消耗。



性能测试对比：

经过和roaring bitmap不同数据大小的测试对比（20亿范围随机），情况如下：

1、10万、50万、100万、500万、1000万数据范围：内存占用略同于roaring bitmap，但耗时只有一半；
2、5000万数据：由于超过3000万转bitmap存在拷贝空间耗用，内存占用大于roaring bitmap，但是耗时更低；
3、1亿数据：内存占用和roaring bitmap接近，但是耗时只有三分之一。
4、2亿及以上数据：内存占用低于roaring bitmap，耗时只有四分之一。

锋刃大数据平台

- 经过1年多的建设，Flink+bitmap从最开始的架构方案和程序demo，已经实施落地为完整的大数据计算系统“锋刃”，系统边界不仅解决实时计算，还包括离线提速和olap，和当前数据工厂（hadoop+hive）互补，并且作为平台提供bitmap结构的文件存储，以及olap的大数据分析系统。
- PCG运营数据应用框架团队提供平台开发和场景实施支持。

业务场景实施及架构升级

腾讯灯塔产品

腾讯灯塔是基于腾讯海量大数据开发的移动应用智能数据分析平台，聚焦数据驱动用户增长，为业务提供分析云与营销云服务。提供包括应用分析、广告效果监测、广告渠道反作弊、DMP标签、市场指数等全链路大数据运营服务。腾讯灯塔秉承独立第三方的数据服务理念，去伪存真，指引有价值的增长。目前日均处理4000亿+日志，覆盖MAU 13亿，积累7大类，1000+标签。

产品官网：beacon.qq.com

灯塔实时统计上线

1、灯塔实时计算经历“实时清洗上线—实时统计开发实施—试运行1个月—故障演练”共3个月，目前已经全量上线，所有产品可以查看一维1分钟和10分钟到实时新增、启动用户、启动次数。当前运行状况正常。

灯塔实时统计上线

2、确定了实时系统服务保障量化指标，保证实时查看结果，对延迟情况保障：平均延时不超过5分钟，异常延时（版本更新重启等）不超过30分钟，超过30分钟属故障。vip产品（1分钟，10分钟，日）保证和离线统计结果偏差不超过万分之五，长尾产品结果偏差不超过千分之五。

3、数据核对情况：以10分钟为标准，目前实时计算结果和离线统计结果核对一致。1分钟离线没有统计，日统计离线有策略补充基准不一样，经过和灯塔业务确认，不影响核对结果。

资源耗用情况：
约35台机器（去重服务：6台m10，flink：11台m10，去重dcache：4台m10共200g，id server：10台docker合计1台m10，id server dcache：10台m10+3ts80）

灯塔2期：
1、启动用户改成前台活跃，而不是后台启动（联网用户），资源许可两者都支持
2、开通更多二维三维的数据实时统计
3、流失用户，回流用户，留存用户的离线提速计算

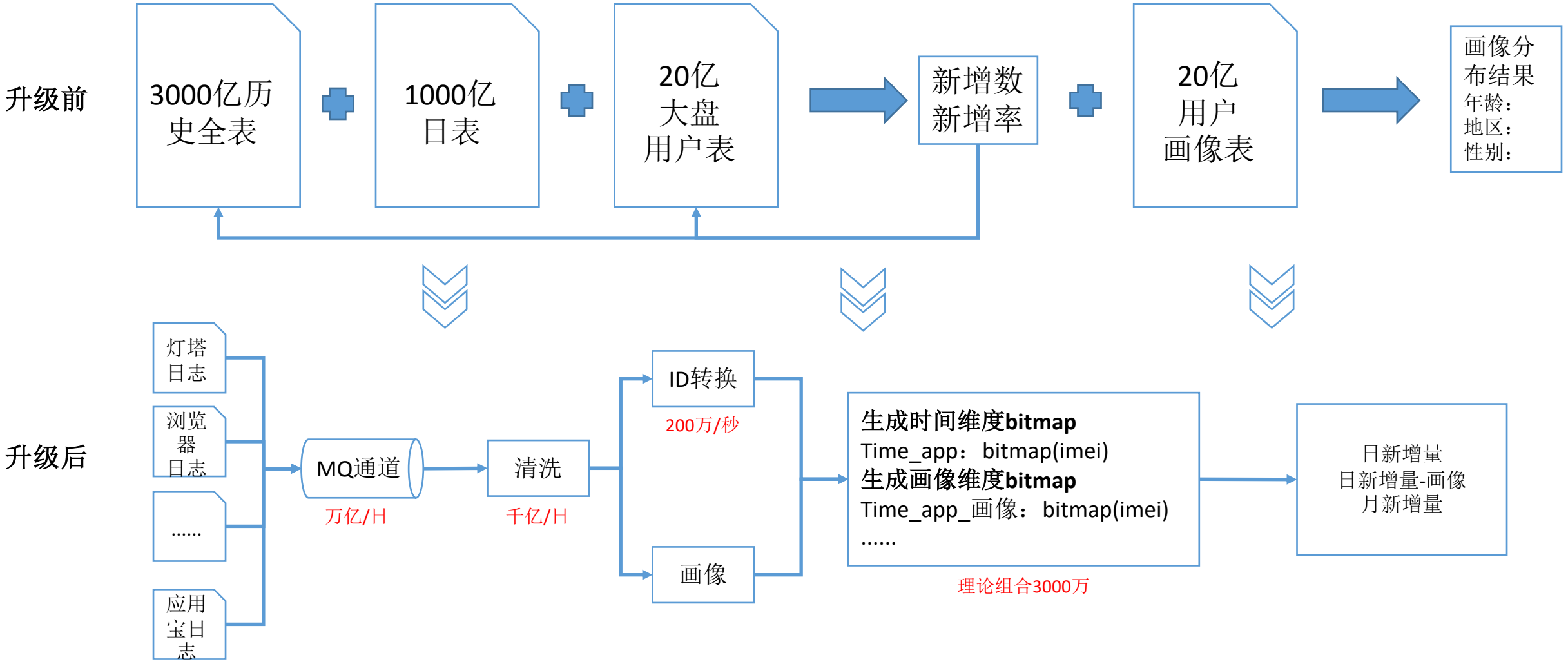
安全云apk离线提速上线

1、需求：当前离线hive计算每日25亿数量大app的卸载留存很难算出，复杂任务计算耗时从早上8点到晚上11点。
2、方案：现改用bitmap方案对1200个app进行优化提速，按app_渠道_日划分为12万个bitmap进行聚合计算。
3、效果：现统计留存卸载的耗时为，日10秒，周20秒，月1-2分钟（90个bitmap聚合）
4、耗用资源：20台机器（flink2台、去重服务7台、dcache10台（400g）、id查询1台）
目前已经上线一期、二期

三期需求：
1、增加全量app的历史bitmap
2、给定一批imei设备号，聚合查询指定多个app bimap
3、增加更多维度的bitmap支持
4、整个雷达从通道端迁入实时处理规划

天玑离线分析架构升级

需求：当前离线hive跑3000亿全表join1000亿日表耗时7小时以上，难以满足模型频繁验证。
评估：3000亿全表按照app维度理论生成3000万的bitmap（其中用于统计的数量在100以上有200万），1000亿日表用于统计的数量100以上的有48万，20亿用户大盘表生成1个bitmap，通过三类bitmap求新增并更新历史全表和大盘用户表。



ABtest实时数据分析平台

运营类应用场景：浏览器框架改版

1、交互稿

A1.主页

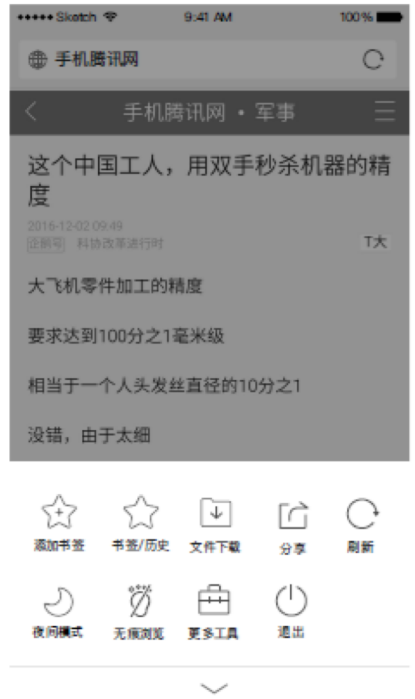
❶ 底bar个人中心入口仅出现在主页，替换原前进按钮。

A2.主页–feed置顶

❶ feed置顶时，后退按钮变为刷新。

A3.个人中心

A4.全局菜单



ABtest实时数据分析平台

应用场景：

1、算法类：浏览器资讯、广告、搜索、推送

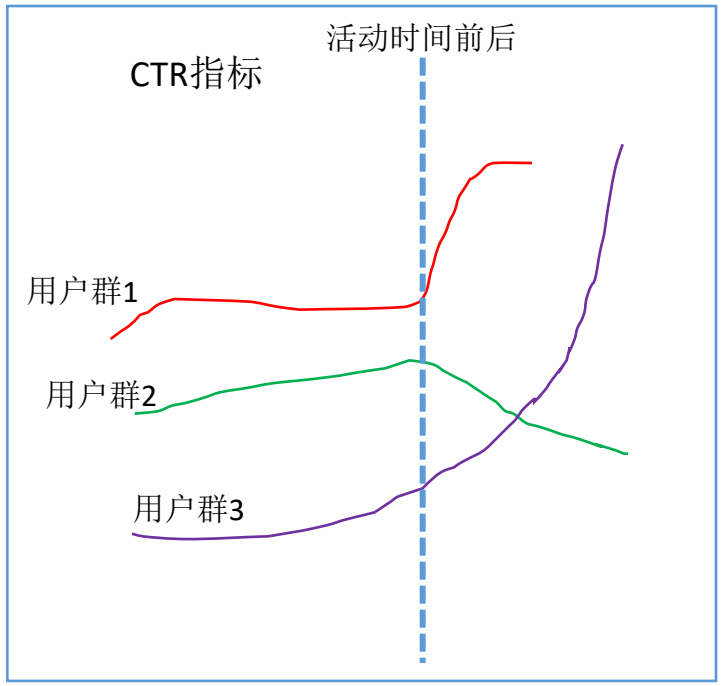
2、运营活动：浏览器框架改版、浏览器vivo装机新用户推送、应用宝活动类

用户标签：（搜索类、看资讯类、点快链类、无行为类）

Abtest需求：

1、PV，UV，CTR；活跃，留存，新增，收入...等实时 / 离线指标提升

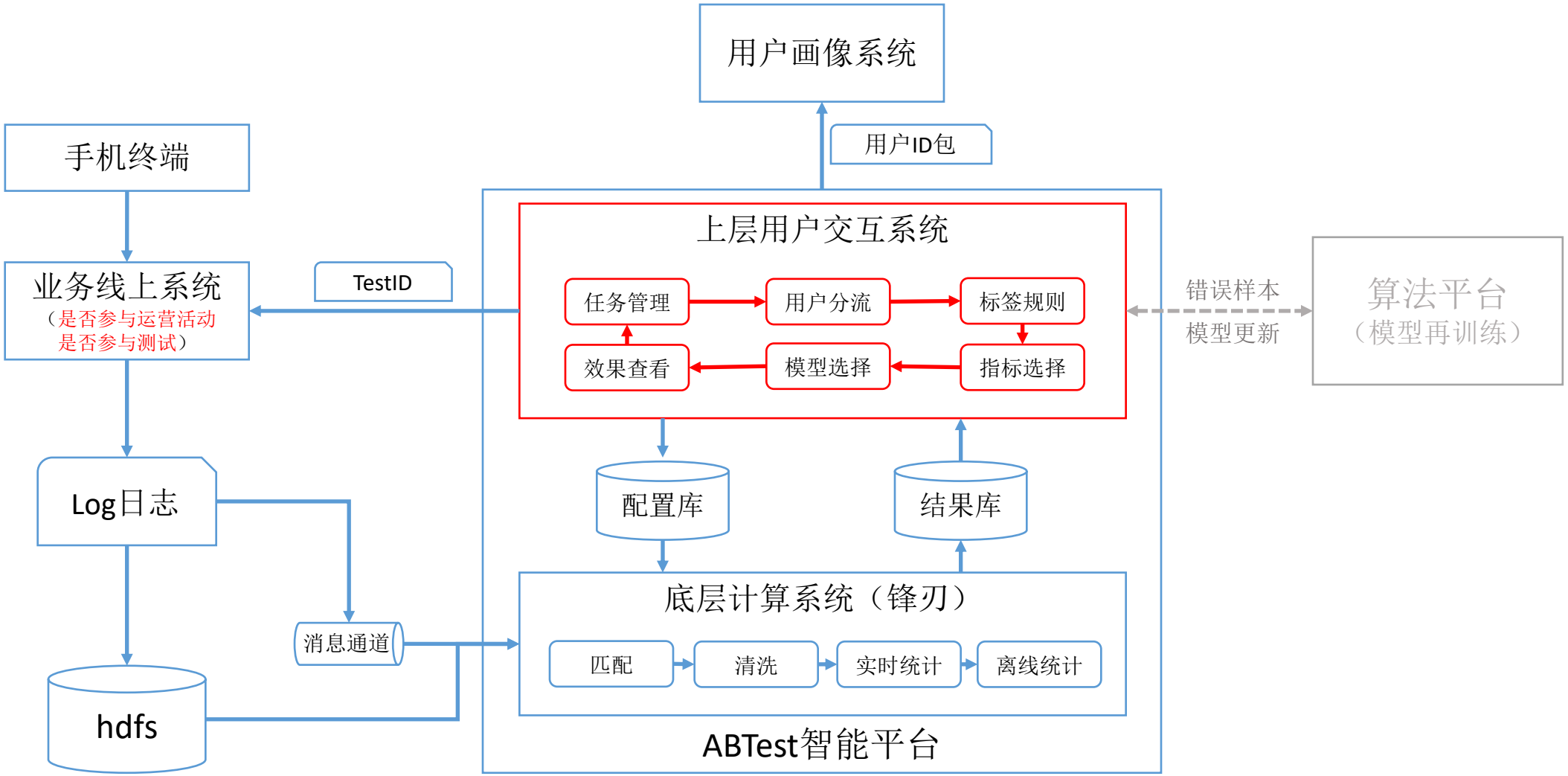
2、模型量化计算综合提升



	活动前		活动后	指标效果
CTR	用户群1			
	用户群2			
	用户群3			
pv				
uv				
留存				
...	...		...	...
综合模型效果				模型量化

ABtest实时数据分析平台

系统架构方案

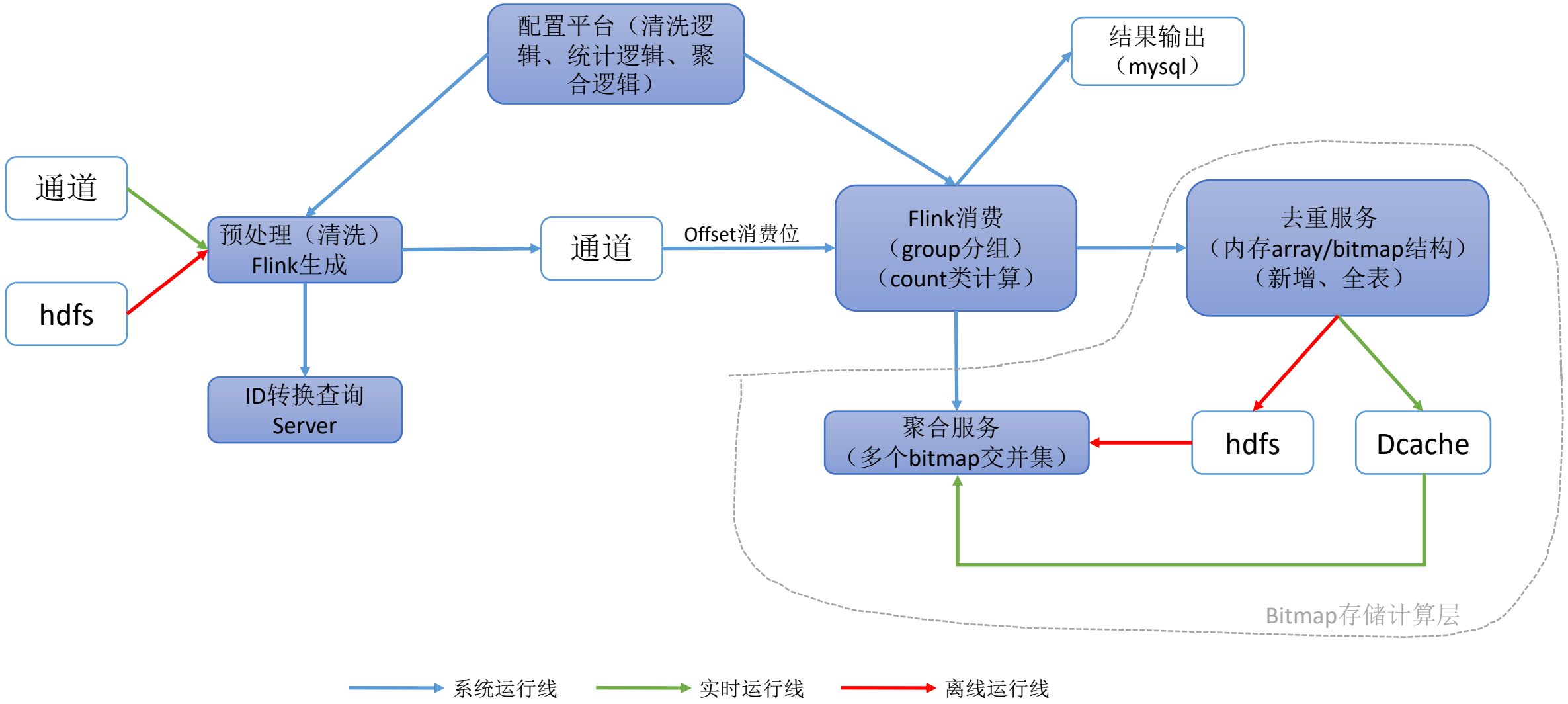


ABtest实时数据分析平台上线

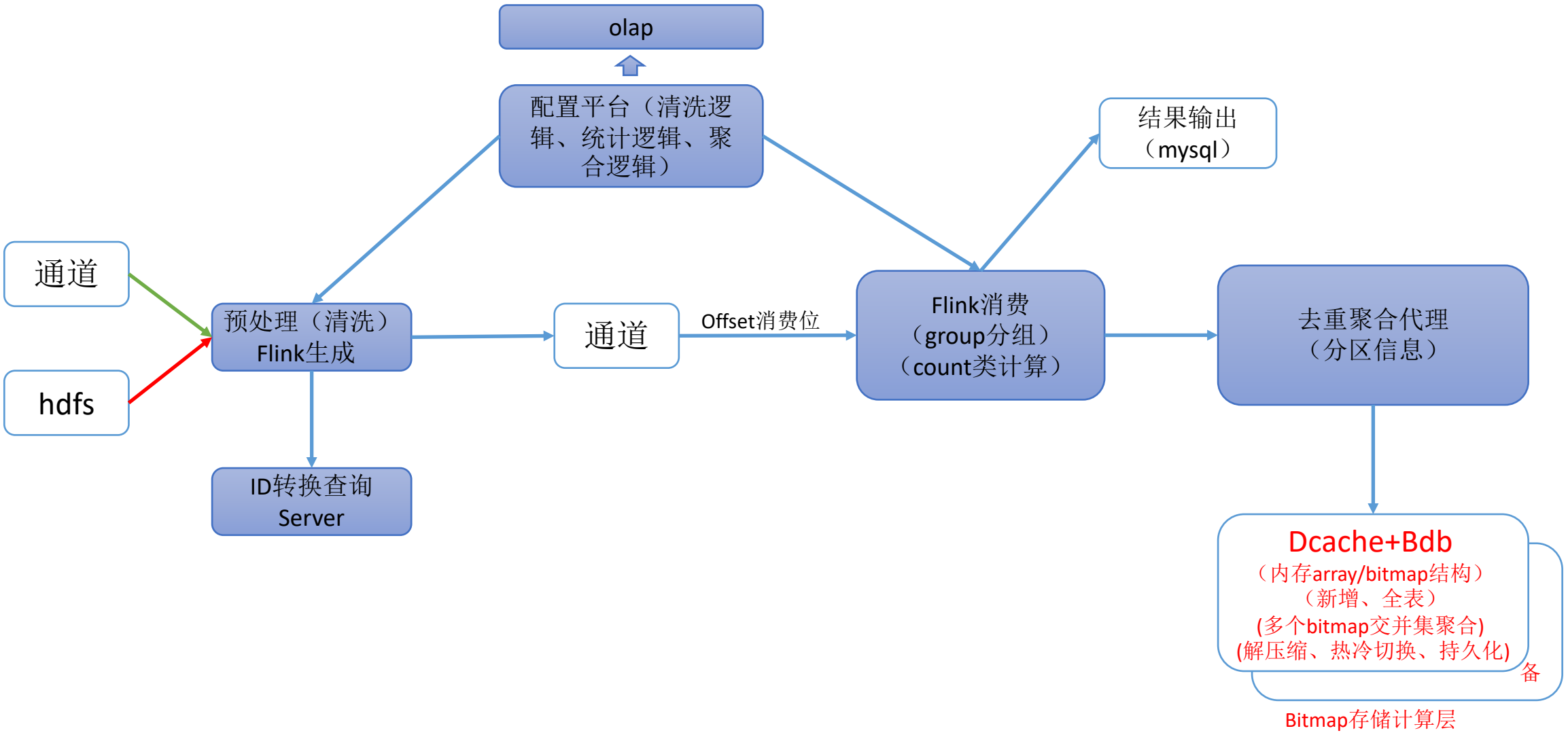
之前几个小时才能看到线上数据，现在只要五分钟！

实时数据能让我们能更快的看到实验数据，及时发现并下线异常实验。同时也能实时监控实验，及大盘核心指标，发现异常数据。更快发现问题就能更及时解决问题，从而降低异常对线上用户的的影响。

业务实施推动架构升级


配置平台（清洗逻辑、统计逻辑、聚合逻辑）
结果输出（mysql）
通道
hdfs
预处理（清洗）Flink生成
通道
Offset消费位
Flink消费（group分组）（count类计算）
去重服务（内存array/bitmap结构）（新增、全表）
ID转换查询Server
聚合服务（多个bitmap交并集）
hdfs
Dcache
Bitmap存储计算层
系统运行线
实时运行线
离线运行线

业务实施推动架构升级


olap
配置平台（清洗逻辑、统计逻辑、聚合逻辑）
结果输出（mysql）
通道
hdfs
预处理（清洗）Flink生成
通道
Offset消费位
Flink消费（group分组）（count类计算）
去重聚合代理（分区信息）
ID转换查询Server
Dcache+Bdb
（内存array/bitmap结构）
（新增、全表）
(多个bitmap交并集聚合)
(解压缩、热冷切换、持久化)
备
Bitmap存储计算层

Olap愿景目标

druid大数据olap主要设计原理

1 聚合表列存储

rowid	timestamp	dimensions				metrics	
		省份	版本	运营商	...	点击数	下载数
1	01:01	gd	1.1	hw		100	50
2	01:02	bj	1.1	oppo		120	50
3	01:03	sh	1.2	vivo		80	50
4	01:04	gd	1.1	oppo		60	50
		列压缩存储	列压缩存储	列压缩存储		列压缩存储	列压缩存储

5 第1行，第4行
点击数=100+60=160

2 bitmap索引

按维度groupby构建rowid的bitmap索引

省份 { ... }
- 0 bj ...00000000 00000010
- 1 sh ...00000000 00000100
- 2 gd ...00000000 00001001

版本 { .. }
- 0 1.1 ...00000000 00001011
- 1 1.2 ...00000000 00000100

按31位wah压缩

3 Select 点击率 where 省份=gd and 版本=1.1?

4 gd&1.1 ...00000000 00001001

druid、锋刃、impala各自适用场景

1、druid最擅长解决“点击数 / 下载数”这样的指标，并且维度取值不是太大的业务场景；解决几十亿imei统计场景比较吃力，需要约束业务范围和数据量，按app和业务分类分表，针对不同业务特定分析。但是对于含有历史新增imei去重的olap、以及多宽表关联仍然不太合适。

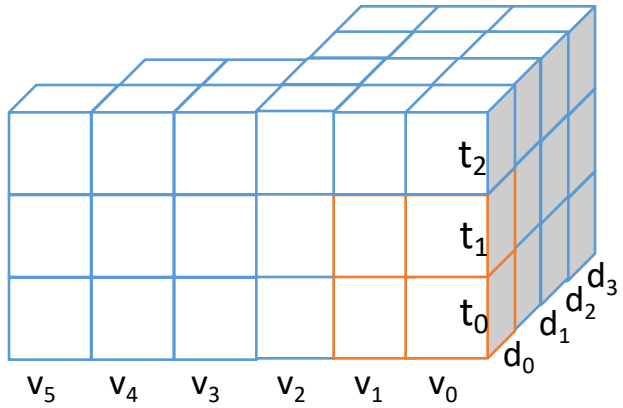
2、锋刃当前适合解决预先定义的关键维度的实时统计和离线统计，是针对imei场景的高清确性的，但是设计上还不能覆盖所有维度的自定义olap。

3、impala适合解决数据范围不大的集群内存能覆盖的业务场景，超出内存限制性能会直线下降。

Cube模型归纳

Cube[t][d][v] = bitmap(rowid)， bitmap(imei)
t：时间(Z轴)，d：维度(Y轴)，v：取值(X轴)

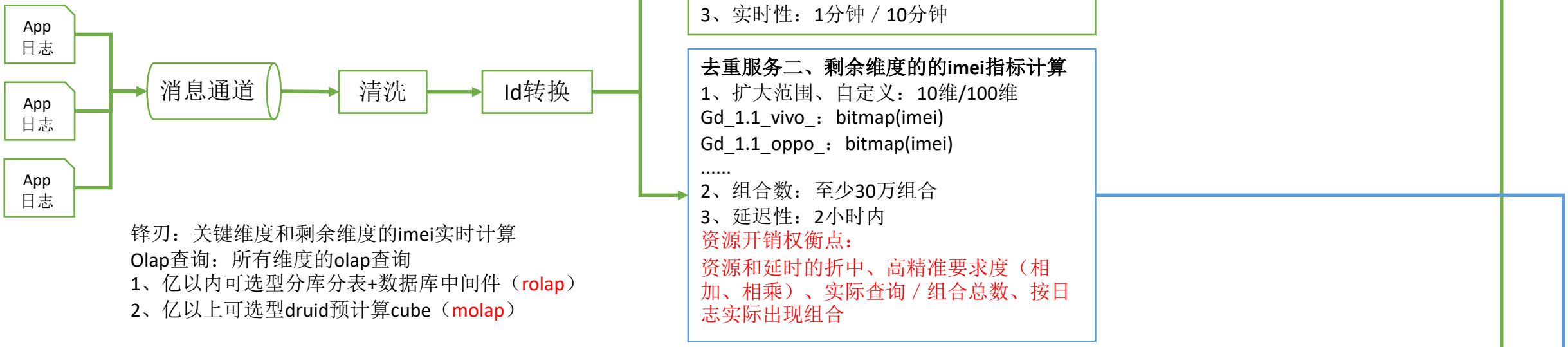
查找t_2时间数据：cube[t_2][ ][ ]
查找d_2维度数据： cube[ ][d_2][ ]
查找t_0-t_1时间，d_0=v_0 and d_0=v_1的数据：cube[<2][d_0][<2]



Olap架构方案：数据预计算和聚合

一、关键维度的实时监控

二、所有维度的olap查询



锋刃：关键维度和剩余维度的imei实时计算

Olap查询：所有维度的olap查询

1、亿以内可选型分库分表+数据库中间件（rolap）

2、亿以上可选型druid预计算cube（molap）

rowid	timestamp	dimensions							metrics	
		省份	版本	运营商	...	去重用户	新增用户	留存用户	点击数	下载数
1	01:01:08	gd				1000个	100个	950个	0	0
2	01:01:09	gd	1.1			200个	30个	180个	0	0
3	03:01:20	gd	1.1	vivo	...	60个	10个	50个	80	50
4	03:01:25	gd	1.1	oppo	...	80个	15个	70个	60	50
5	03:02:10	bj	1.1	hw	...					
									sum	avg

实时导入

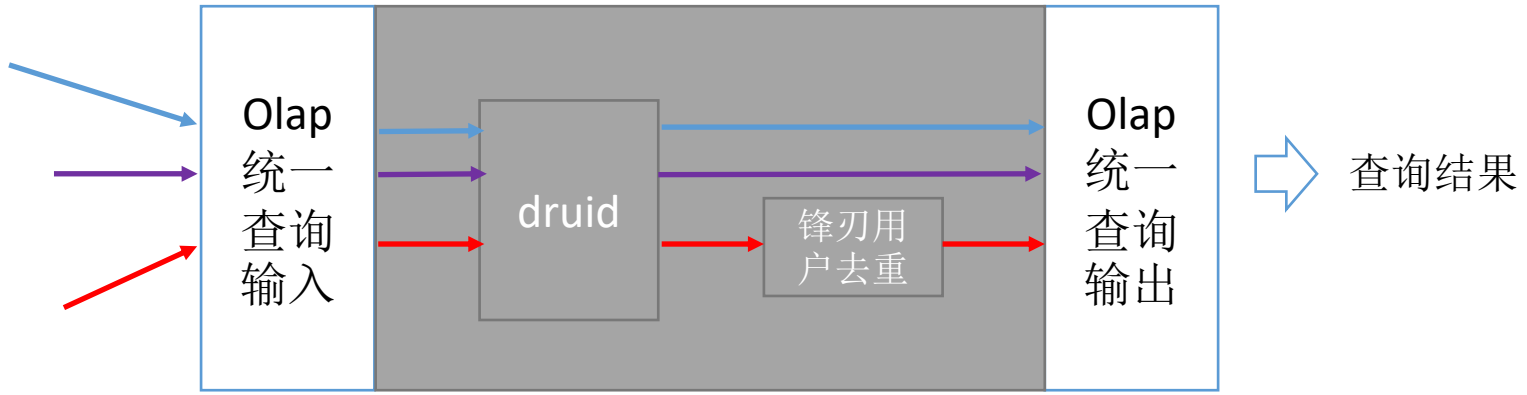
延时导入

Olap架构方案：数据自定义查询过程
统一的用户输入输出
一、大部分的数量指标和去重用户结果直接查询druid
二、少数的去重用户结果的动态聚合经过锋刃再处理

1、Select sum(点击数) where 省份=gd

2、Select sum(去重用户) where 省份=gd

3、Select sum(去重用户) where 省份=gd and timestamp=[from, to]



锋刃+druid Olap平台