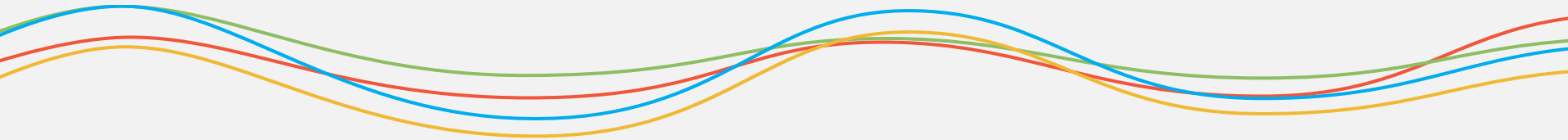
APMCon

实时OLAP数据仓库架构优化演进

张海雷@优酷土豆



提纲

APMCon

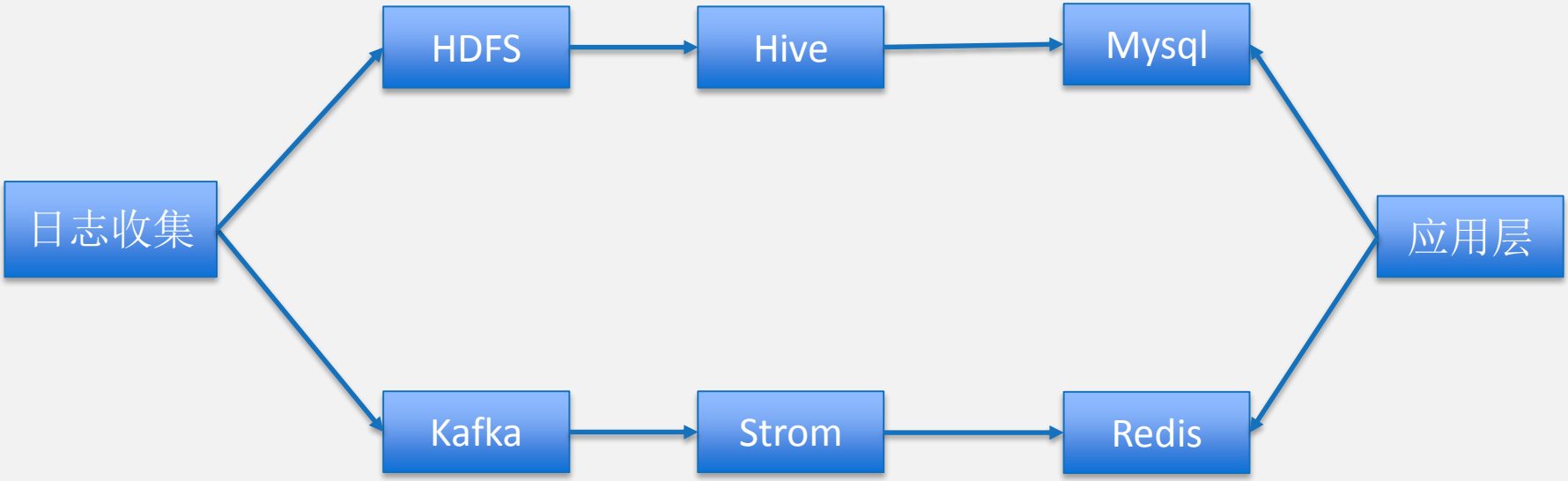
- 最初的架构
- Druid简介
- Druid和其他OLAP的对比
- 使用Druid以后的架构

需求背景

为DSP的广告主提供实时多维分析报表

- 多维钻取 16个维度 22个metrics
- 海量数据 10亿量级
- 实时性 延迟为分钟级别
- 多维下的UV 每天独立访客千万量级

最初的架构



实时采用预算维度组合的metric

- 维度组合作为Key，各项Metric按照一定格式拼装作为Value

问题一 维度组合膨胀

- 穷举的方式预算各种组合。
- 维度的个数、组合稀疏程度以及基数
- 增加维度以后，有可能带来指数式的增长。

问题二 如何支持Group By查询

- Redis不支持Range Scan，支持前缀Scan，需要扫描整个HashTable
- 在应用层穷举维度组合，然后采用mget

问题三 如何实现UV

- Redis支持HyperLogLog

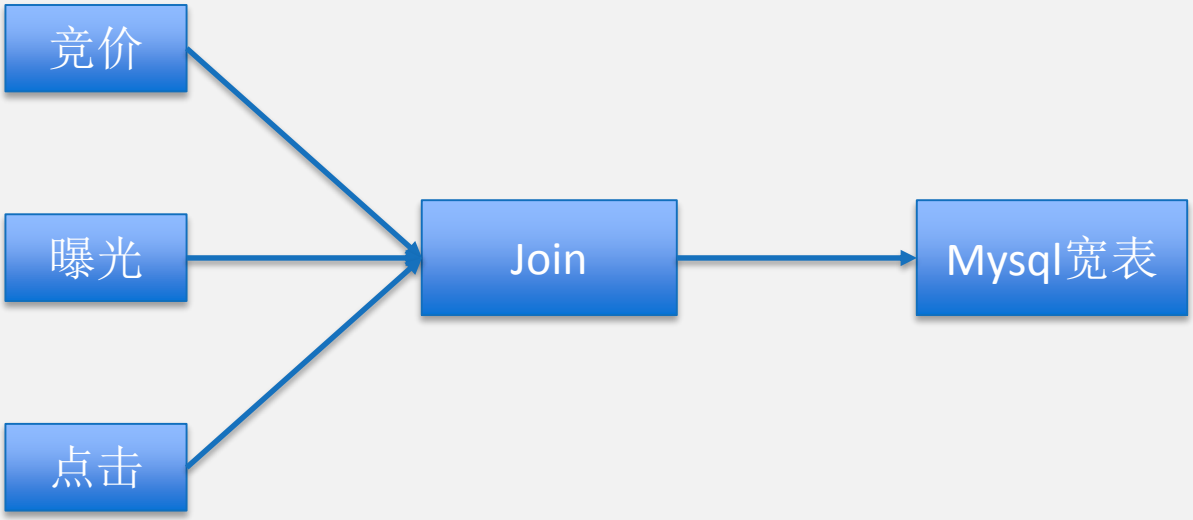
举个例子

假设有三个维度，每个维度的基数都是3。
维度A有A1、A2和A3
维度B有B1、B2和B3
维度C有C1、C2和C3
最差情况的维度组合：
D-3: ABC * 3^3
D-2: AB * 3^2 + AC * 3^2 + BC * 3^2
D-1: A * 3 + B * 3 + C * 3

总结:
增加维度，维度组合数量有可能是指数式增长
这只是最差的情况，现实数据的维度组合是稀疏。

离线架构

采用Hive将多种日志源的数据经过ETL以后写到mysql全维度的宽表
瓶颈以及挑战： 全维度组合的数量



架构-功能降级

APMCon

由于预算所有维度的实现成本很大，功能降级。

- 选取固定（两级）的维度组合存放在Redis中
- Mysql按照维度组合分表

缺点以及改进

APMCon

缺点

- 增加维度以后，给存储容量以及计算量带来很大的挑战
- Redis不支持Range Scan，Group By查询需要客户端穷举维度组合
- Redis和Mysql异构数据库，实时和历史相结合的查询需要在应用层合并

改进

- 使用HBase替换redis+mysql？
- Druid？

Druid是什么

APMCon

Druid是开源的分析型时序数据库，为OLAP而生。

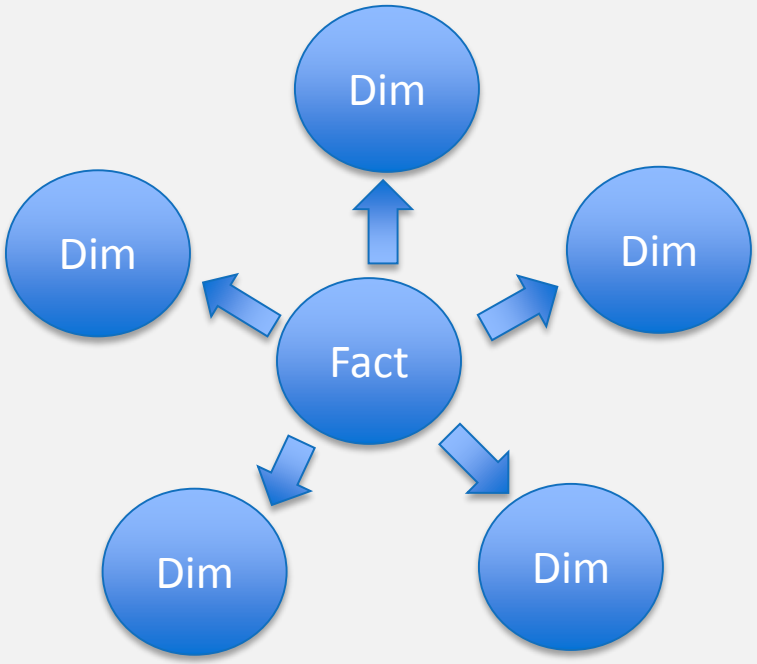
- 多：处理海量处理，可以扩展到PB级
- 快：亚秒级响应，实时导入，导入既可查询
- 好：高可用，分布式容错架构，可以做到永不宕机
- 省：列存储，高效压缩
- 高：支持高并发，面向用户的应用

Druid支持的查询

- 支持TimeSeries、GroupBy、TopN、Select等常用查询
- 适合星型模型，不支持大表之间的join，其lookup功能实现和维度表的join
- 支持聚合函数，count和sum，以及使用javascript实现自定义UDF
- 支持扩展Aggregator，阿里贡献了利用Bitmap实现精准的DistinctCount
- 支持近似查询

  HyperLoglog 计算UV

  DataSketches 计算UV以及留存分析



数据格式

APMCon

- Druid数据由Timestamp、Dimension和Metric组成
- Timestamp是特殊的Dimension

timestamp	publisher	advertiser	gender	country	click	price
2011-01-01T01:01:35Z	bieberfever.com	google.com	Male	USA	0	0.65
2011-01-01T01:03:63Z	bieberfever.com	google.com	Male	USA	0	0.62
2011-01-01T01:04:51Z	bieberfever.com	google.com	Male	USA	1	0.45
2011-01-01T01:00:00Z	ultratrimfast.com	google.com	Female	UK	0	0.87
2011-01-01T02:00:00Z	ultratrimfast.com	google.com	Female	UK	0	0.99
2011-01-01T02:00:00Z	ultratrimfast.com	google.com	Female	UK	1	1.53

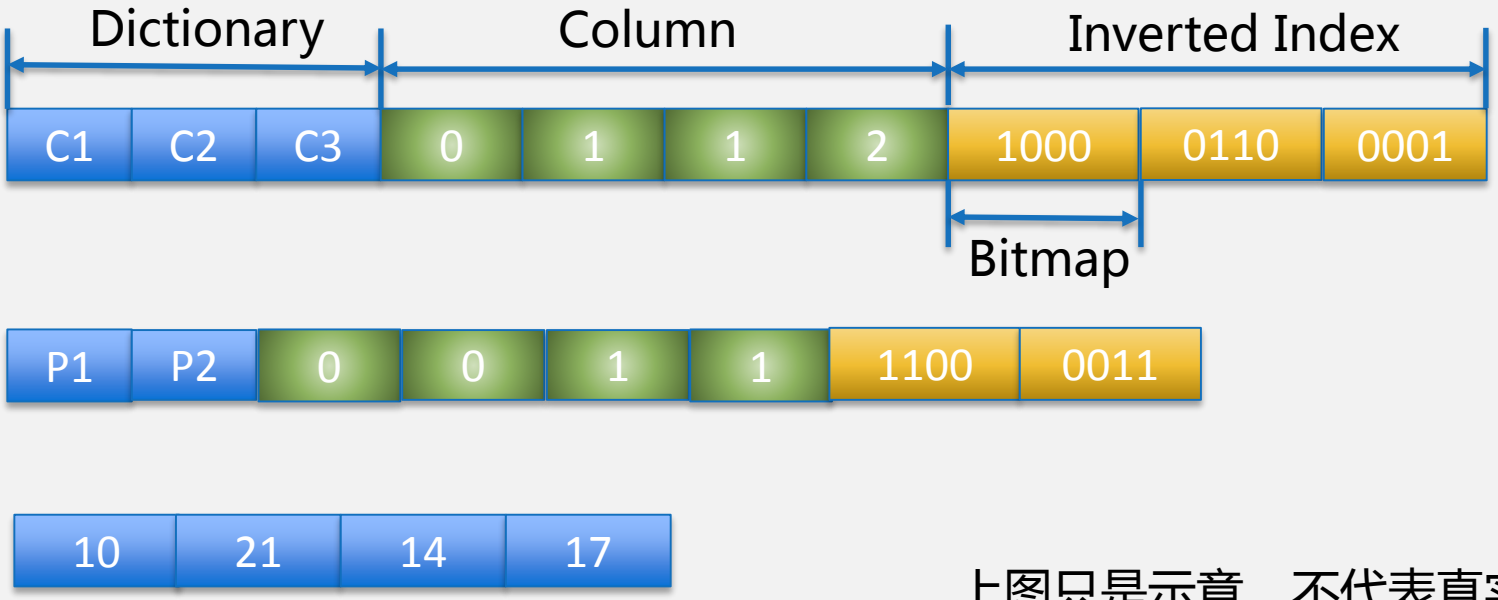
Roll-up

Roll-up是Druid的特性，其有力地保障了低延迟的响应时间。
Roll-up是在数据导入阶段时进行一次聚合，第一层聚合。
Roll-up过程类似做一次全维度的Group By：

```
GROUP BY truncate(timestamp), publisher, advertiser, gender, country :: impressions = COUNT(1),
 clicks = SUM(click), revenue = SUM(price)
```

优点:减少数据集的大小，**甚至是百倍以上**
缺点:不能查明细数据

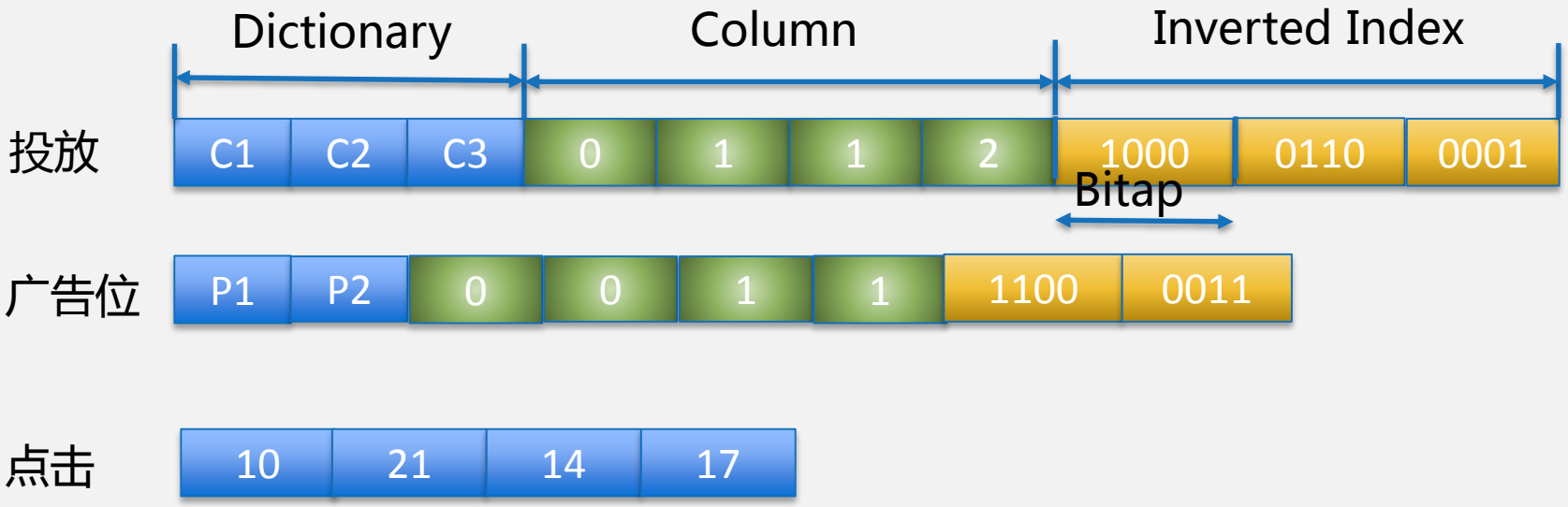
投放	广告位	曝光数	点击数
C1	P1	17	10
C2	P1	31	21
C2	P2	29	14
C3	P2	30	17



上图只是示意，不代表真实结构

Select Position ,sum (click) from table where cast=C2 group by Position

Step1: 根据过滤条件cast=C2,找到C2的字典值是1
Step2: 根据字典值1找到bitmap 0110
Step3: 根据bitmap得到**offset**是1和2
Step4: 根据Offset构建Cursor遍历
Step5: 单次迭代 offset1得到Position是0，反查字典是P1，在click中得到21



Druid高效的奥秘-列存储和压缩

APMCon

列存储

BitMap

- Concise
- Roaring

高效压缩

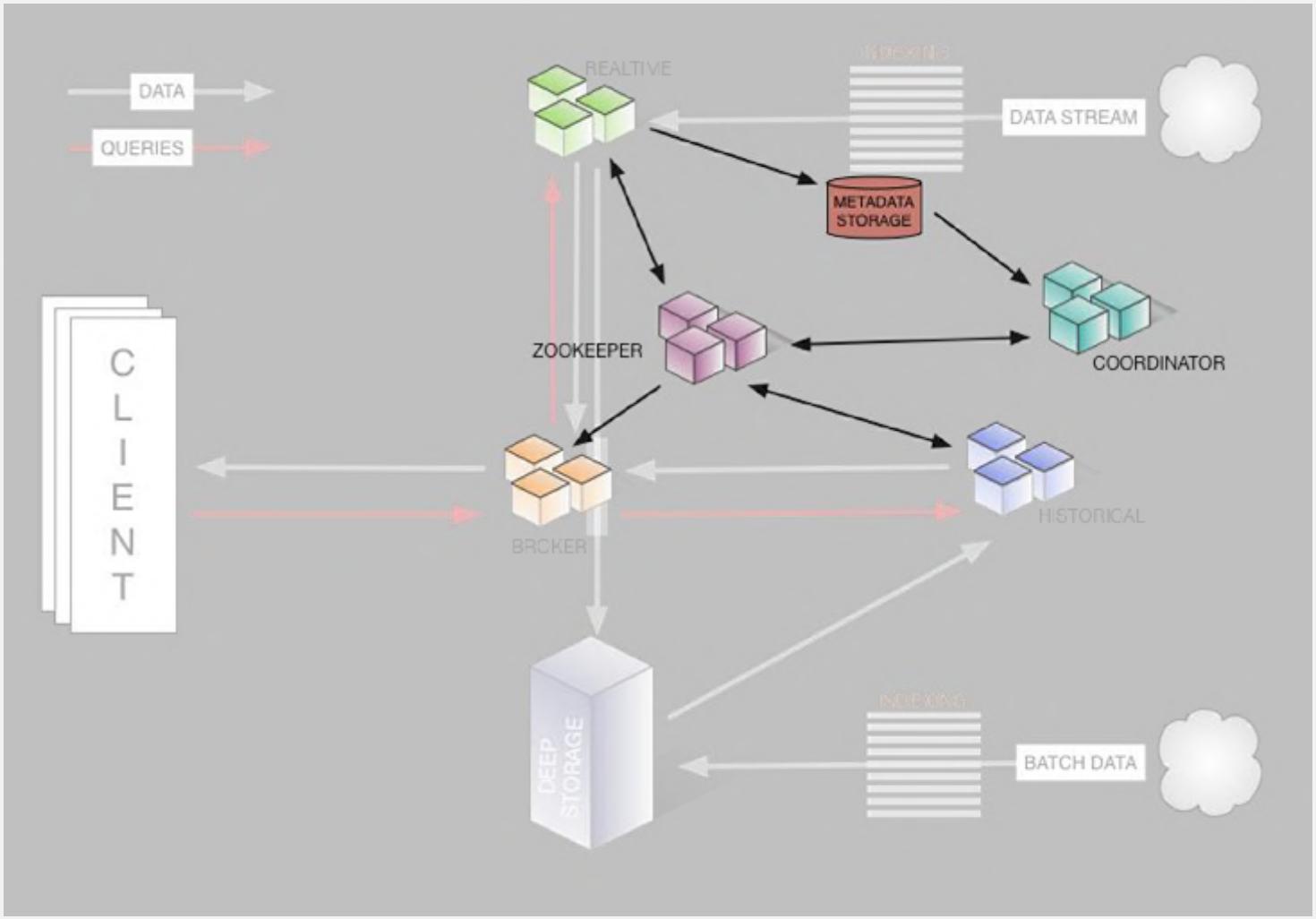
Dimension:

- 正排： 字典编码、变长整数编码和按块压缩
- 倒排： BitMap

Metric：

- 按块压缩，默认64K，必须是2的整数倍，方便定位。
- 例如存储Long，那么sizePer=64K/8=8192
- **blockNum = index / sizePer;**
- **blockIndex = index % sizePer;**

Druid架构


DATA
QUERIES
REALTIVE
DATA STREAM
METADATA STORAGE
ZOOKEEPER
COORDINATOR
CLIENT
BRCKER
HISTORICAL
DEEP STORAGE
BATCH DATA

Realtime

APMCon

- 实时接收(拉取)数据，生成Segment
- 负责实时部分的查询
- 采用类LSM-tree的架构，对写友好，支撑高并发和高吞吐量的写入
- 内存增量索引采用ConcurrentSkipListMap
- 达到阈值以后，异步线程将内存增量索引持久化成倒排索引
- 达到Segment设定的时间粒度时，将这一时间段内的持久化索引合并成Segment
- 具体实现有Realtime Node和Indexing Service两种方式

Coordinator

APMCon

- 负责协调Segment的均衡加载
- 动态均衡Historical节点的负载
- 根据用户指定的Load Rule加载指定时间段的Segment

Historical

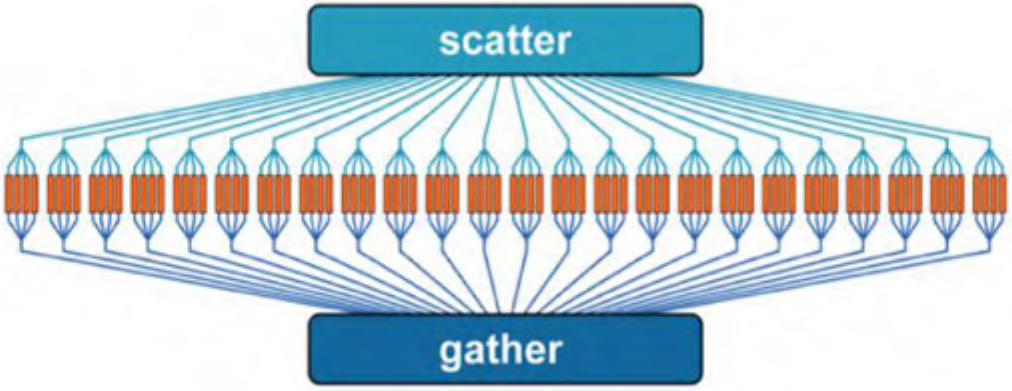
APMCon

- Historical是查询的主力
- 从Deep Storage中拉取Segment，采用mmap的方式Load
- Historical可以配置负载能力
- Scatter/Gather 模型。查询时分发给多线程执行，每个Segment分配一个线程，然后再聚合。
- Shared-Nothing架构，扩展性强

Broker

APMCon

- 通过ZooKeeper获取TimeLine.
- Scatter/Gather模型，把时间段的查询转化成Segment的查询分发给Historical或者RealTime 然后再在Broker层聚合
- 采用Http Restful API提供查询
- 自实现NettyHttpClient，异步Nio的方式
- 缓存查询结果



外部依赖

Zookeeper

- LoadQueue，Coordinator分发Segment的实现是将Segment加到指定节点的LoadQueue
- Coordinator的failover实现，利用Zookeeper的选主
- 时间轴，用于查询路由分发到指定的节点

Deep Storage 用于Segment的备份存储

- HDFS、S3和Cassandra等
- 使用其他任何对象存储，自定义模块实现

Meta信息存储

- Mysql
- 其他关系型数据库，自定义模块实现

如何保障高可用

RealTime

- RealTime Node 采用从Kafka中拉取数据的模式，不支持多副本，不能保障高可用
- Indexing Service 实现了导入阶段多副本，保障高可用

Historical

- 采用多副本加载

Coordinator

- 主备模式，采用Zookeeper选主

Broker

- 利用Zookeeper的节点发现以及客户端负载均衡

Druid vs Elasticsearch

APMCon

- Druid在导入过程会对原始数据进行Roll-up，而ES会保留原始数据。
- Druid专注于OLAP，针对数据导入以及快速聚合操作做了优化。
- Druid不支持全文检索。

Druid vs. Key/Value Stores (HBase/Cassandra/OpenTSDB)

APMCon

KV存储的通用方式：

① 预算所有可能的维度组合

② 在事件记录上进行Range Scan，所有的维度组合作为key，metrics作为value

缺点：

第一种，给空间和计算带来挑战，增加维度以后指数式增长。

第二种，所有的维度组合作为key ，Range Scan没有索引的话会扫描大量的数据

Druid vs SQL-on-Hadoop (Impala/Drill/Spark SQL/Presto)

APMCon

SQL-on-Hadoop提供了执行引擎可以在多种数据格式上进行查询，它也可以查询Druid的数据

查询模式

- Druid Scatter-Gather模型，算子下放，算子执行和存储在同一节点，broker只是聚合结果
- SQL-on-Hadoop一般采用MPP，执行计划的不同算子分布在不同节点执行，增加数据序列化以及网络传输的开销

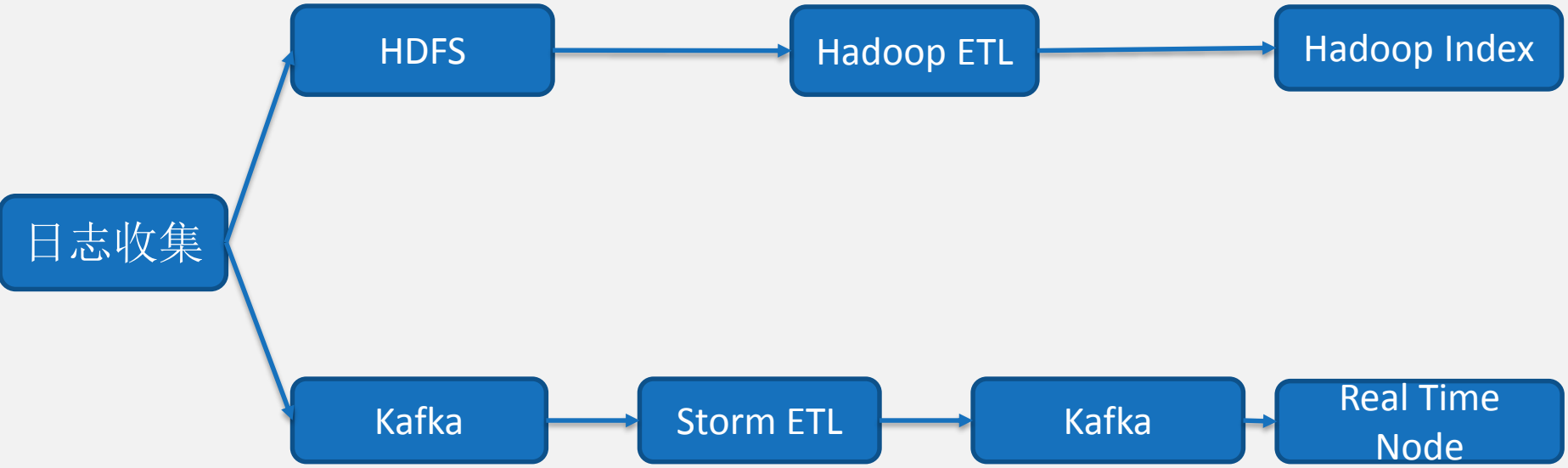
数据摄入

- Druid 支持实时导入

SQL支持

- Druid只支持单表，不支持大表之间的Full join

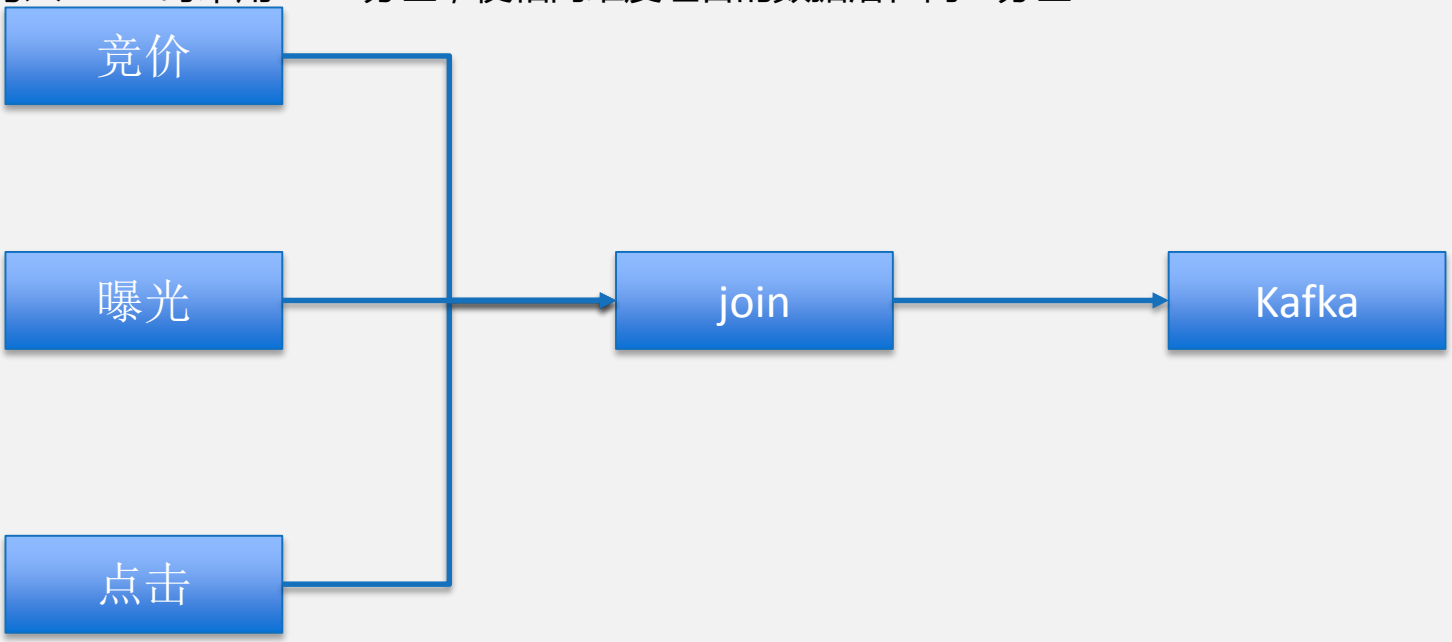
使用Druid以后-Lambda架构



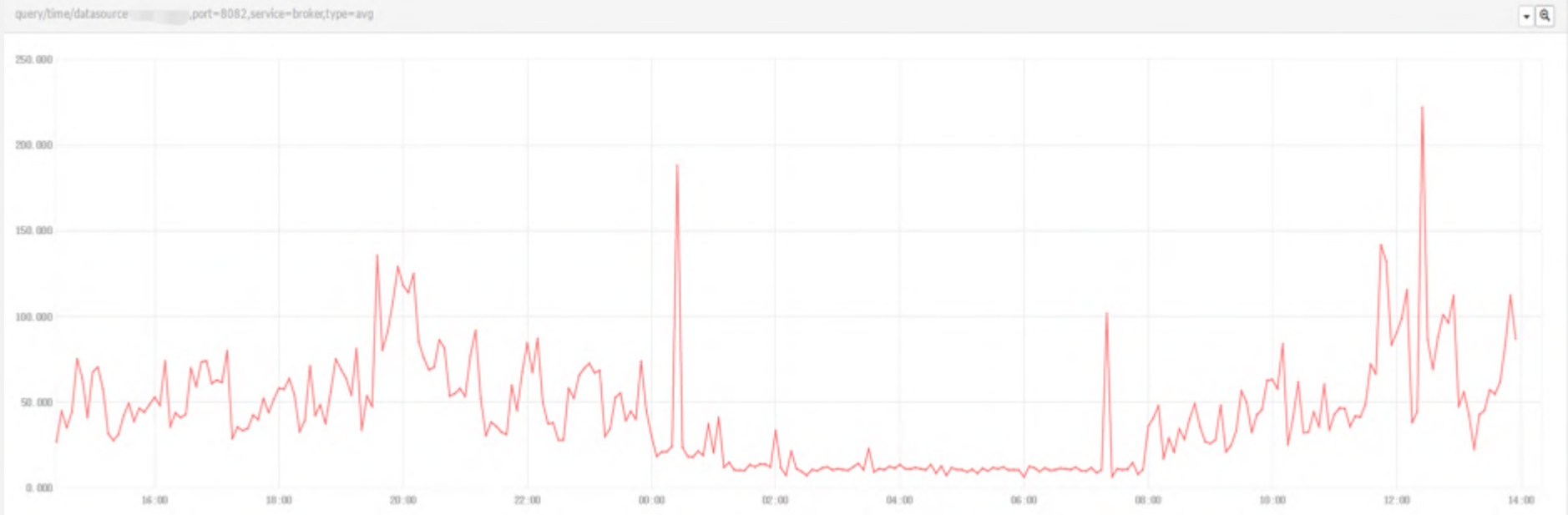
使用Druid以后-实时部分

APMCon

- Druid只支持单表，采用把多表预先Join成单表
- 每种日志维度冗余，采用Storm的Field Grouping把相同维度的不同日志连接在一起
- Storm中在一定时间窗口内预聚合，减小Druid的压力
- 写入Kafka时采用Hash分区，使相同维度组合的数据落在同一分区



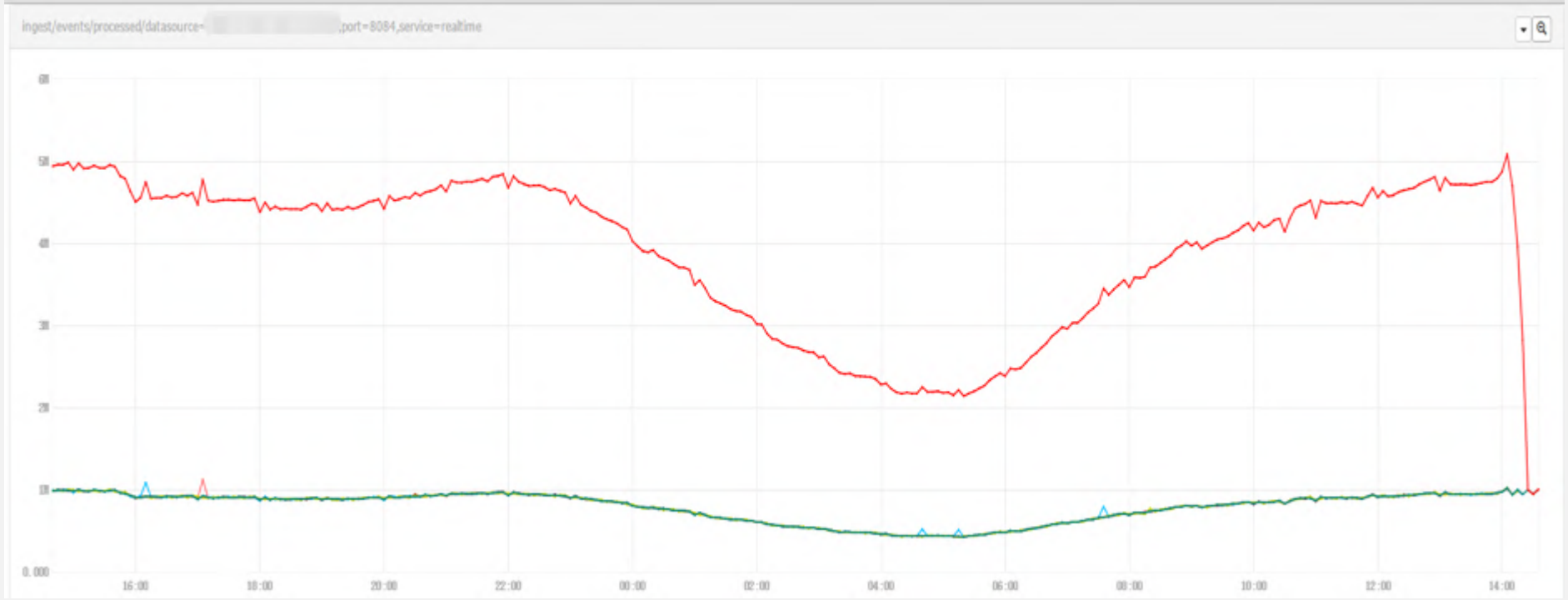
使用Druid以后-性能表现



使用Druid以后-数据导入吞吐量

APMCon

下图和上一页的响应时间不是同一张表，只是为了说明Druid数据导入的吞吐量



THANK YOU