China · Beijing

腾讯OMG广告大数据实践

李曙鹏
高级工程师

[北京站]

主办方 Geekbang 极客邦科技 InfoQ

大纲

业务概况

典型广告数据应用

数据应用架构支撑

- ✓ 日志收集利器CloudDumper
- ✓ 日志实时session化
- ✓ 查询引擎：在sparksql上构建cubes

腾讯OMG广告业务概况

- 服务视频和新闻为主的媒体广告变现


腾讯视频


腾讯新闻 & 天天快报



微信 & 手Q新闻插件


腾讯体育

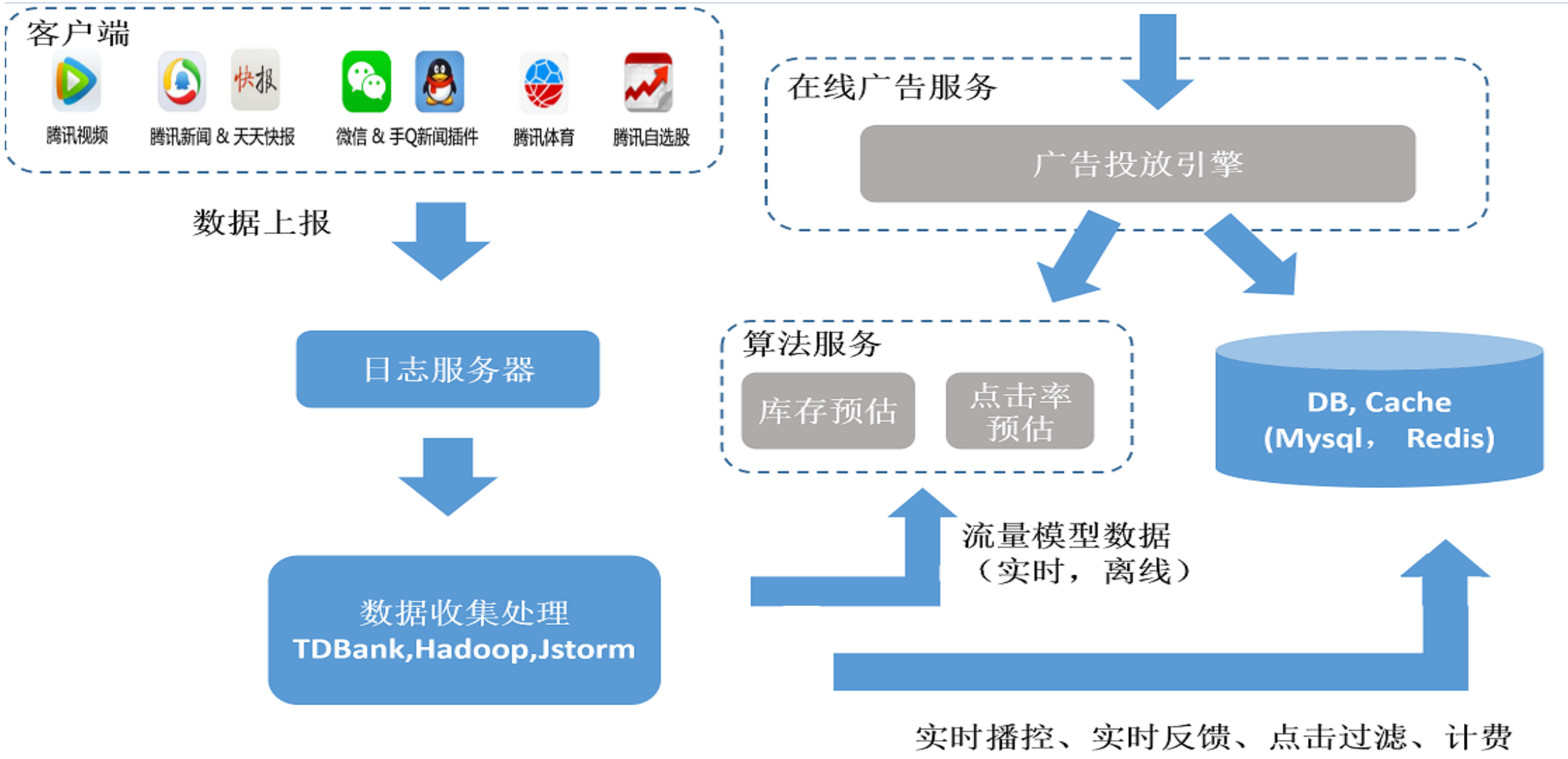

腾讯自选股

- 每日服务数百亿广告曝光

- 年广告收入达百亿级

典型广告数据应用

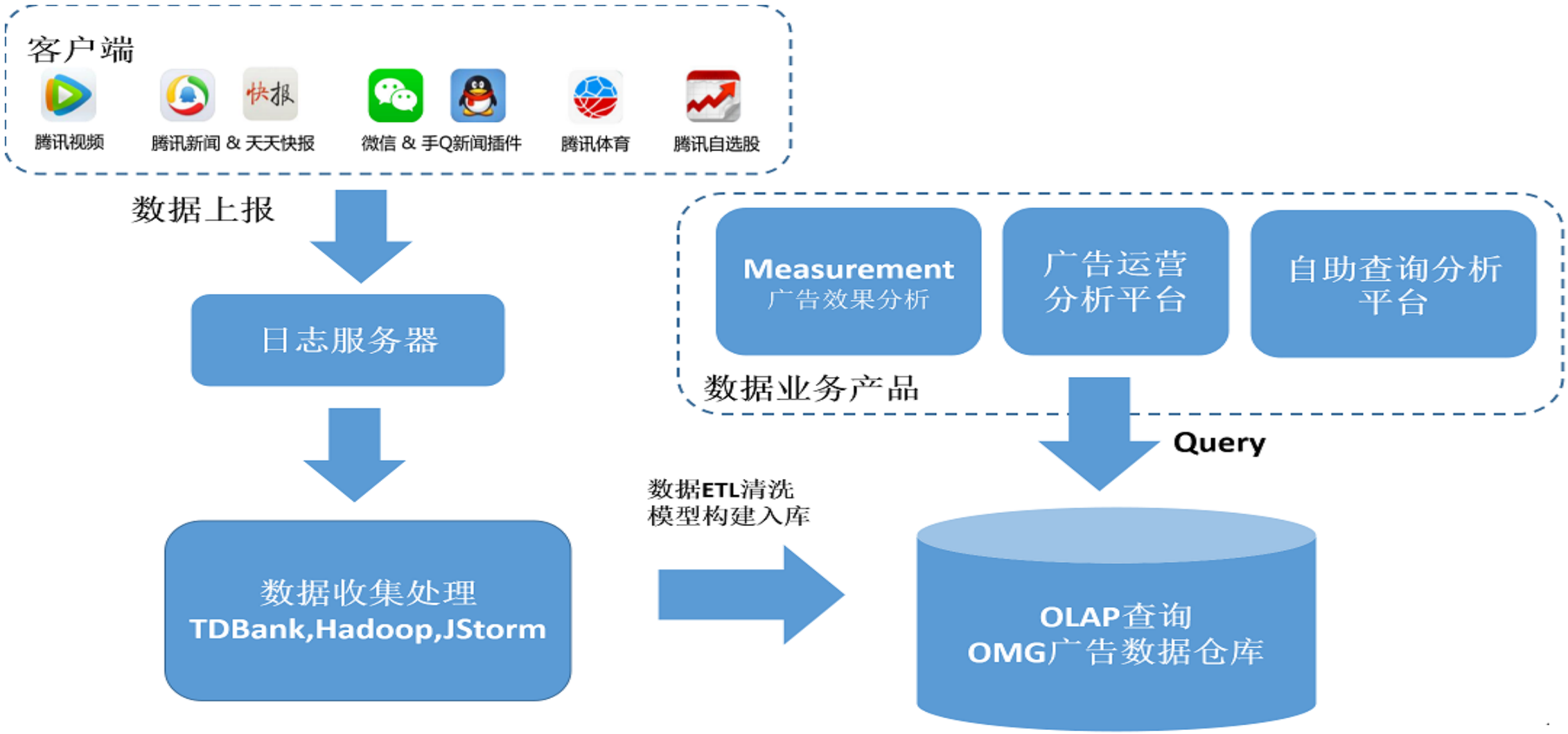
在线数据服务



- ✓ **实时播控：** CPM合约广告，实时计算订单级别的曝光频次，并进行播放控制。
- ✓ **点击率预估：** CPC效果广告，基于在线曝光、点击流量生成带特征的正负样本数据，点击率模型训练，在线预估。

典型广告数据应用

离线数据分析



- ✓ **报表类查询：** 售卖漏斗分析、流量漏斗分析、收入结构分析等
- ✓ **自助查询分析：** 满足各部门的商业分析需求，支持任意维度的交叉分析

关键挑战

- 三种数据形态以及对应的关键挑战

CloudDumper
收集分拣系统

关键挑战：
快速可靠实时接入

广告日志数据：
检索、曝光、点击原始日志，每天30T+，300亿+条

实时session系统

关键挑战：
超长时间窗口聚合，高吞吐和低延时

广告SESSION：
包括检索属性、曝光属性、点击属性、用户特征兴趣、历史广告行为、订单信息、广告位信息等

广告数据查询引擎

关键挑战：
灵活的业务模型、更丰富的维度、高效查询响应、数据一致性（尤其是收入指标）

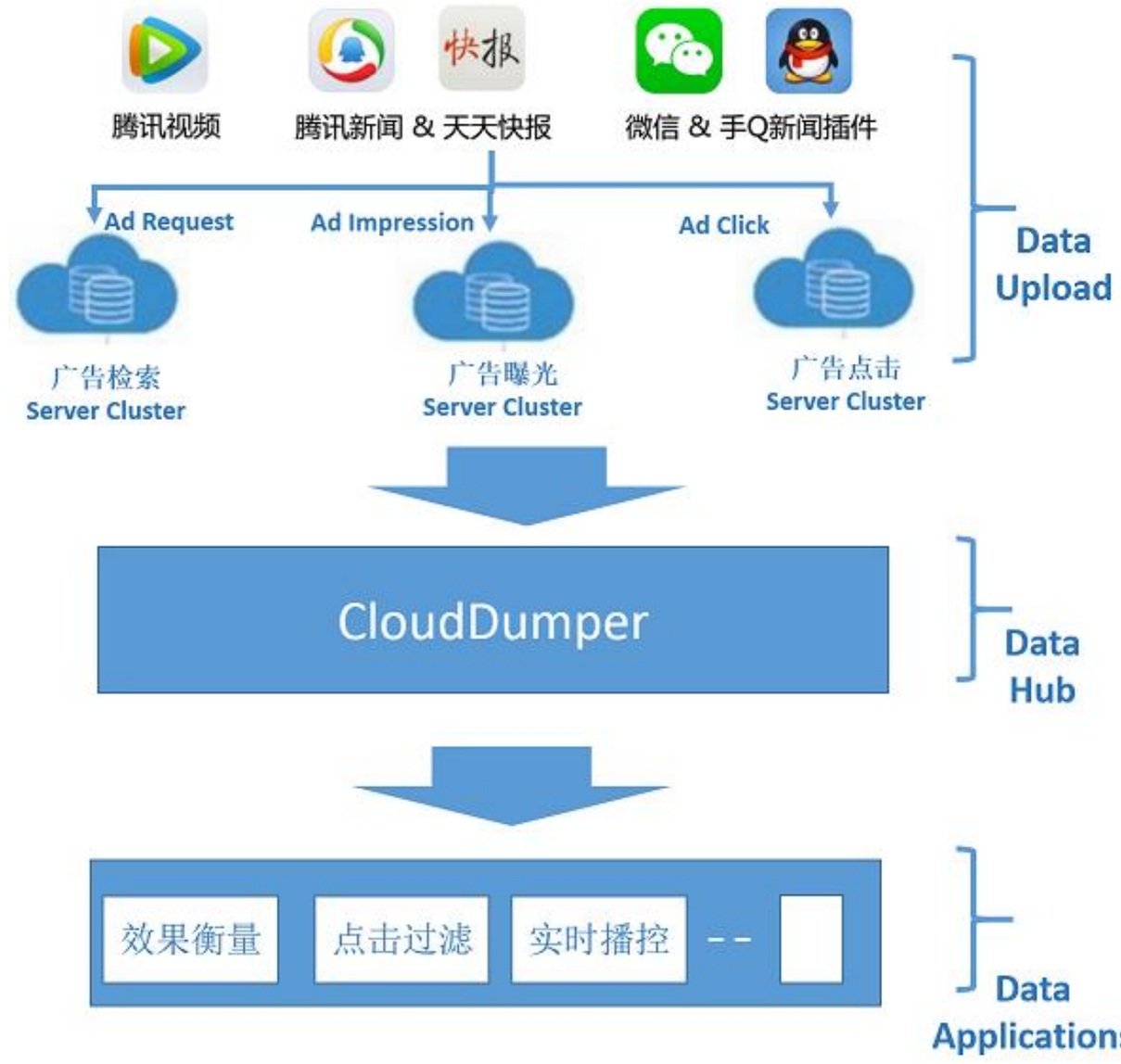
广告维度模型：
面向维度（广告位、平台、地域等）、指标（曝光量、点击量、收入）的可聚合数据，历史总量PB级、单表最大200T，100+维度。

数据收集利器: CloudDumper

- 业务接入概况
  - ✓ 接入腾讯网、腾讯视频、新闻客户端、微信/手Q新闻插件等业务
  - ✓ 下游对接15+个实时计算应用

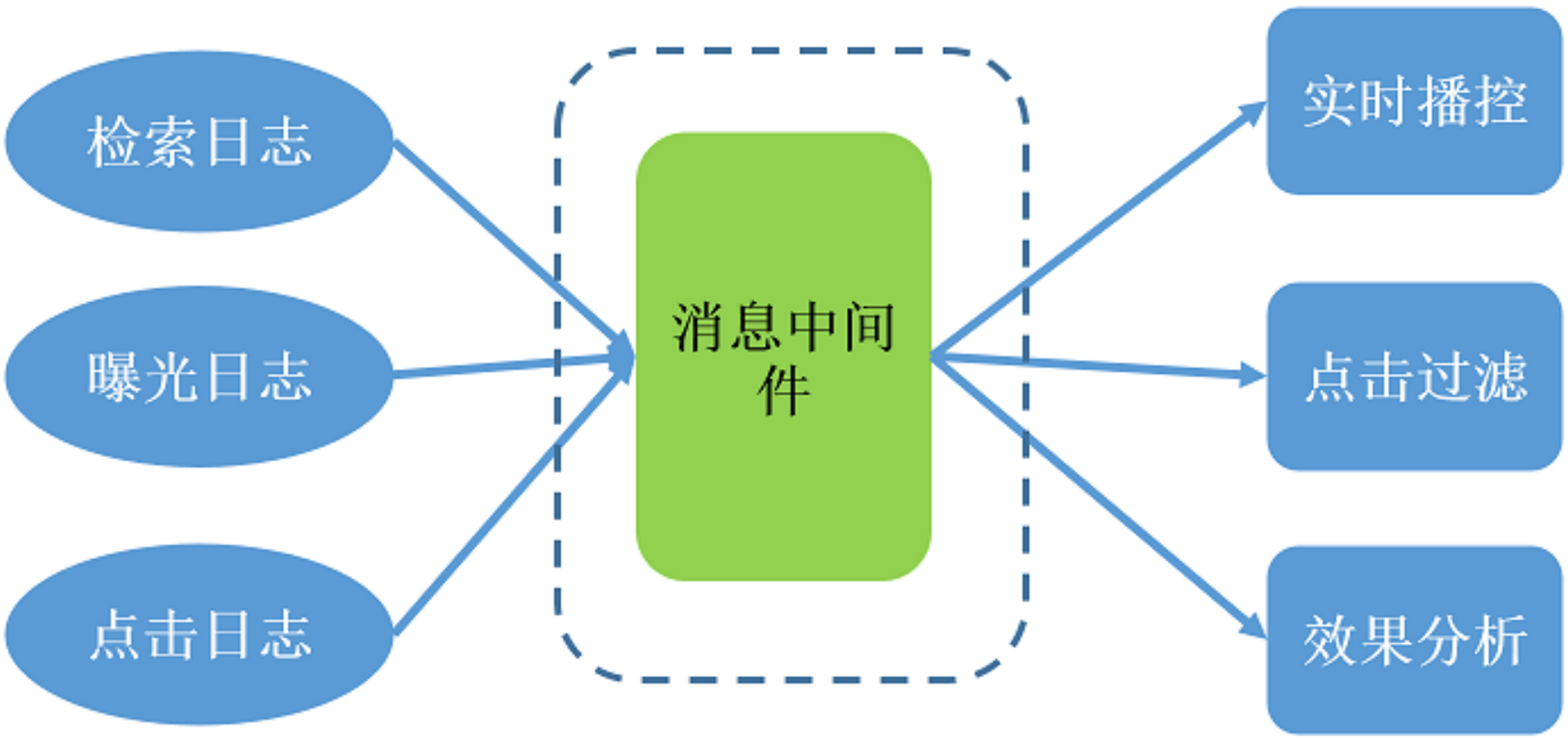
- 一些运营数据：
  - ✓ 日均原始日志约300亿条，约30T+
  - ✓ 峰值QPS 100W/S, 平均延时秒级

- 打造一个“**数据总线**”，满足：
  - ✓ 数据集中接入
  - ✓ 秒级实时处理
  - ✓ 按需分拣落地
  - ✓ 业务升级不停流



技术方案选型

- 通常方案：

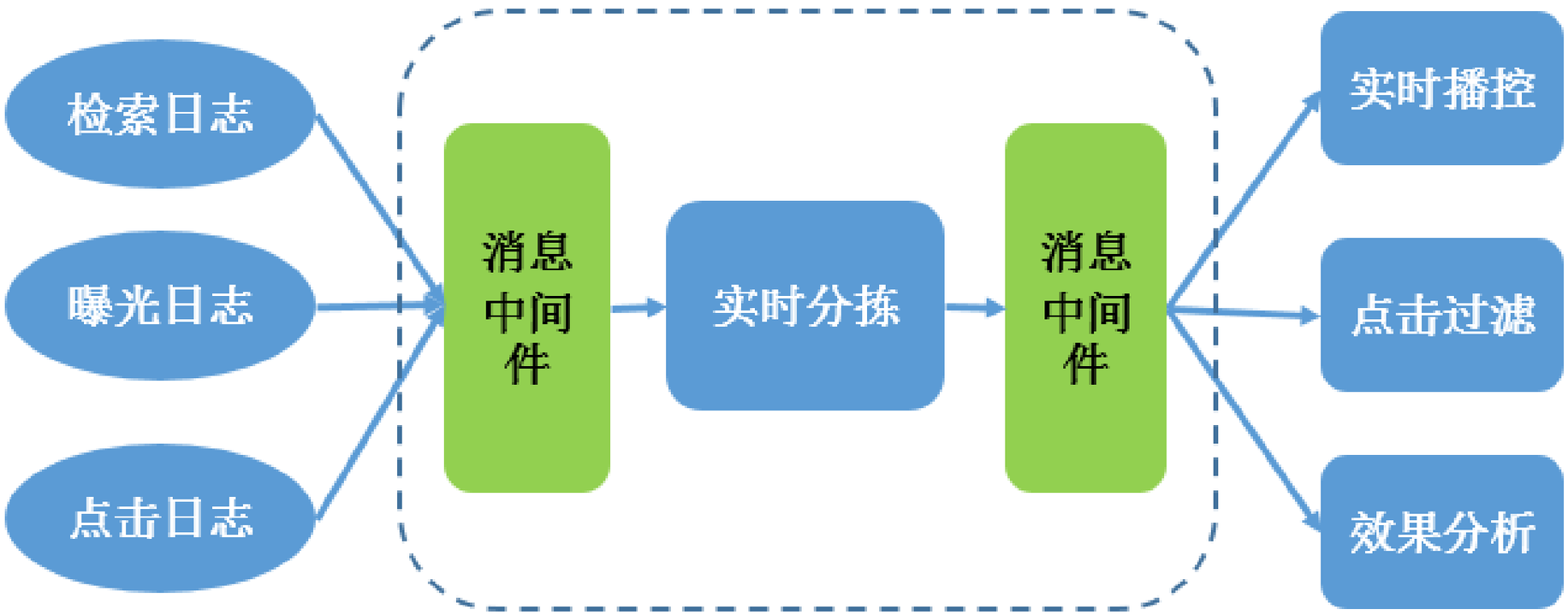


- 存在的问题：
  - ✓ **日志端集中式接入、业务应用只关心自己需要的数据和字段**

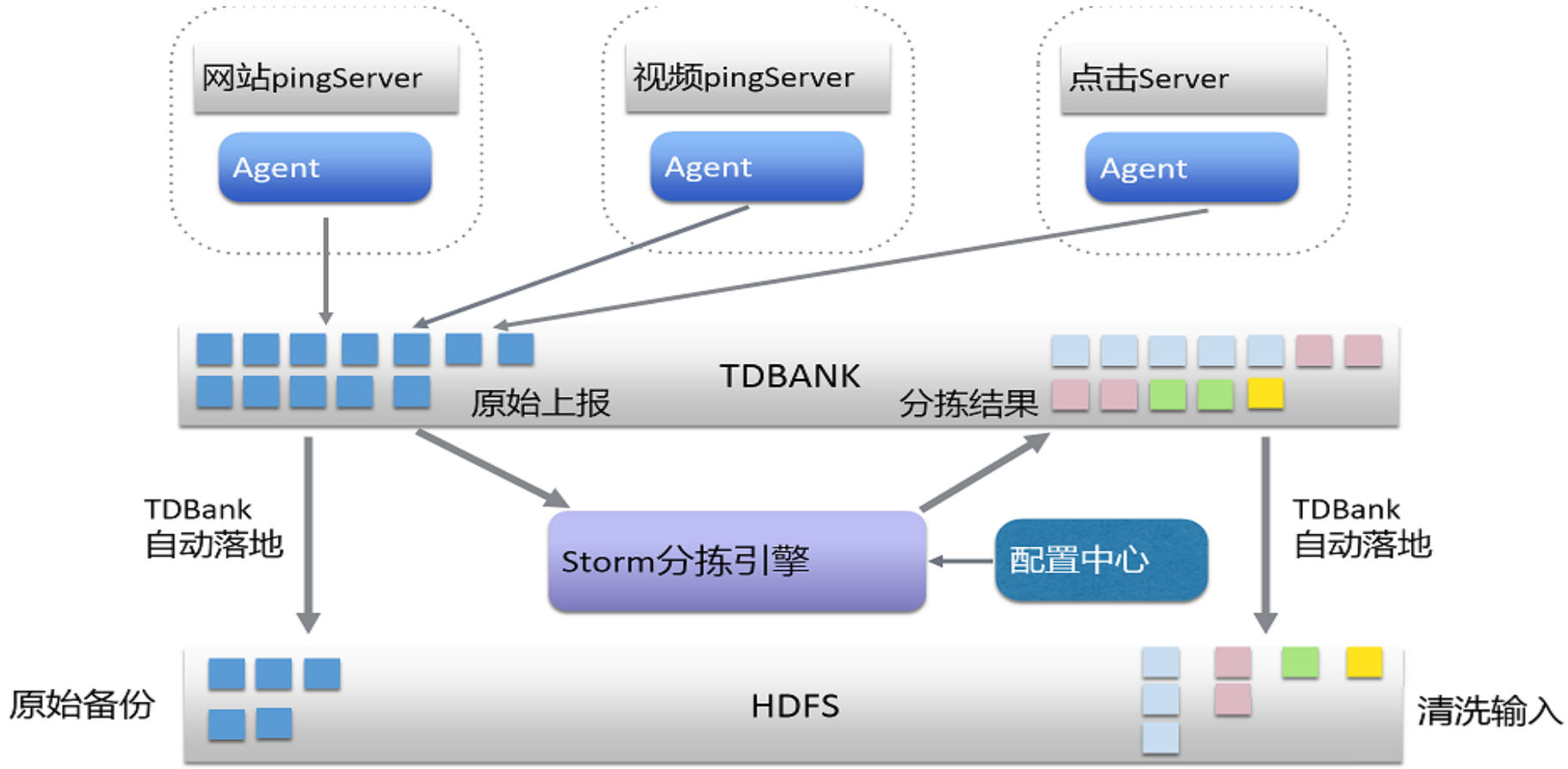
    例如：某个下游应用只关心**新闻客户端、品牌广告**的曝光数据
  - ✓ **日志端分拣：** 与server端耦合、分拣配置不一致、无可恢复数据现场
  - ✓ **应用端分拣：**网络开销大，增加业务接入成本

技术方案选型

- **分拣逻辑云端化：**
  - ✓ **日志端：**集中式接入、保持业务无关性。
  - ✓ **Cloud端：**负责分拣逻辑
  - ✓ **应用端：**轻量级数据订阅



实现架构



- **日志端：**
  - ✓ **Agent：** 全量上报、业务无关、高可靠性、高性能
- **Cloud端：**
  - ✓ **消息中间件：**腾讯TDBANK（类Kafka)，实时高吞吐、支持自动落HDFS
  - ✓ **分拣引擎：**

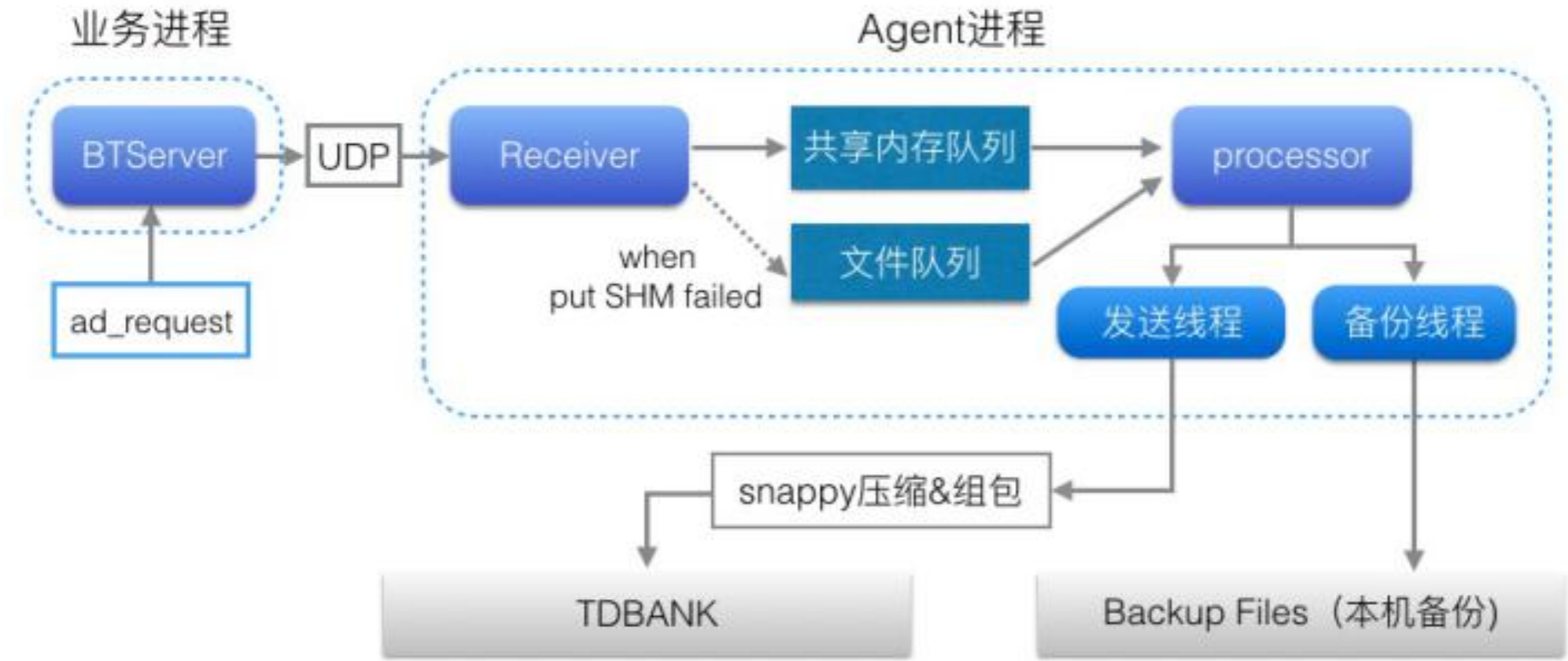
    Streaming为主(JStorm)、Batch为辅，分拣逻辑配置化、业务与框架解耦

Agent

- **数据接口协议：** KV形式、UDP接口、 全量上报、黑名单过滤字段

TIMESTAMP	TAB	DATA<K=V>, TAB分割

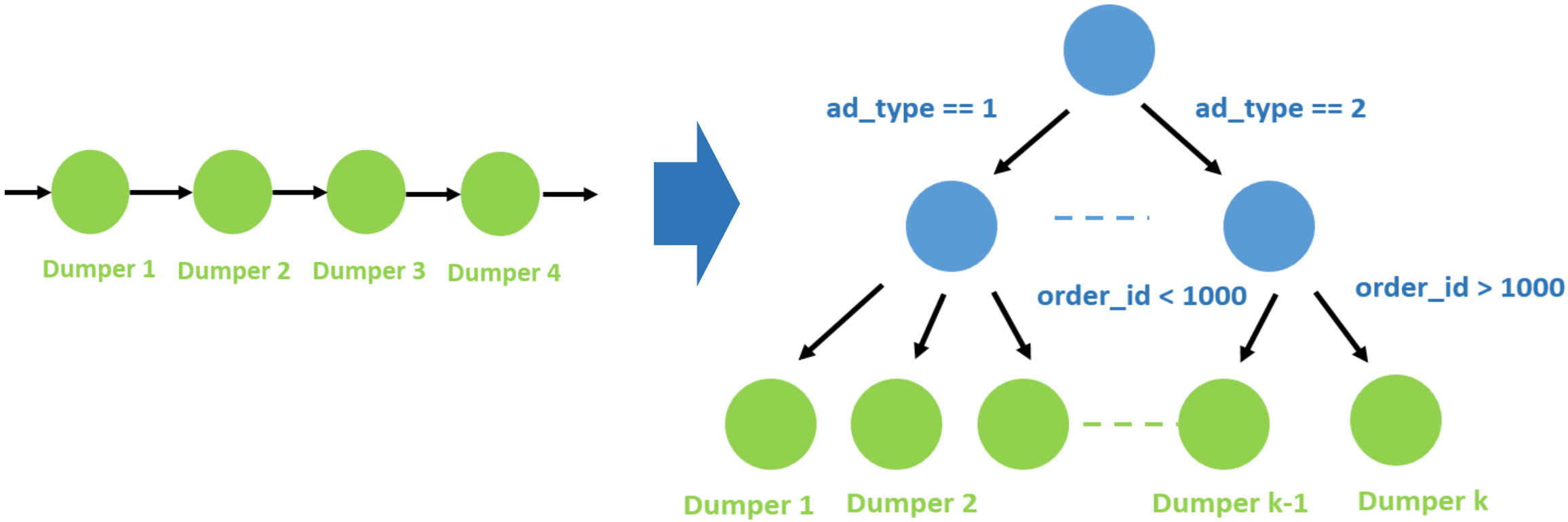
- **轻量级：**无业务逻辑，资源占用小，不影响业务进程。
- **可靠性：**无状态、不丢数据（即使出现网络拥塞）



分拣引擎

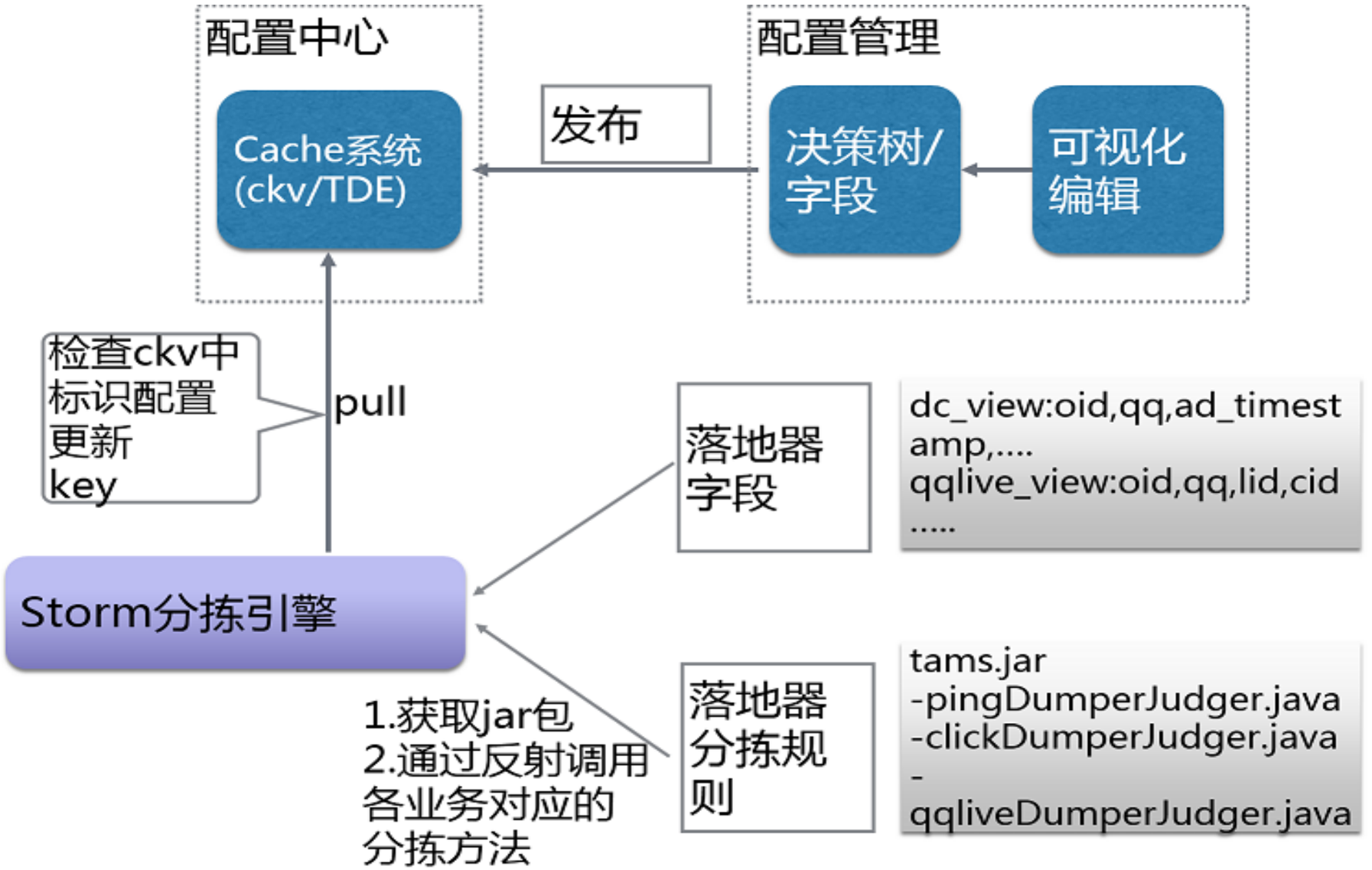
分拣逻辑

- ✓ **流式处理：** TDBank + JStorm，条件判定，落地器
- ✓ **链式：** 串行遍历，低效
- ✓ **决策树：** 条件驱动，高效可扩展，可配置化，可读性强



分拣引擎

- **在线发布更新：**业务升级不停流、 灰度控制、一致性保障



日志实时Session化

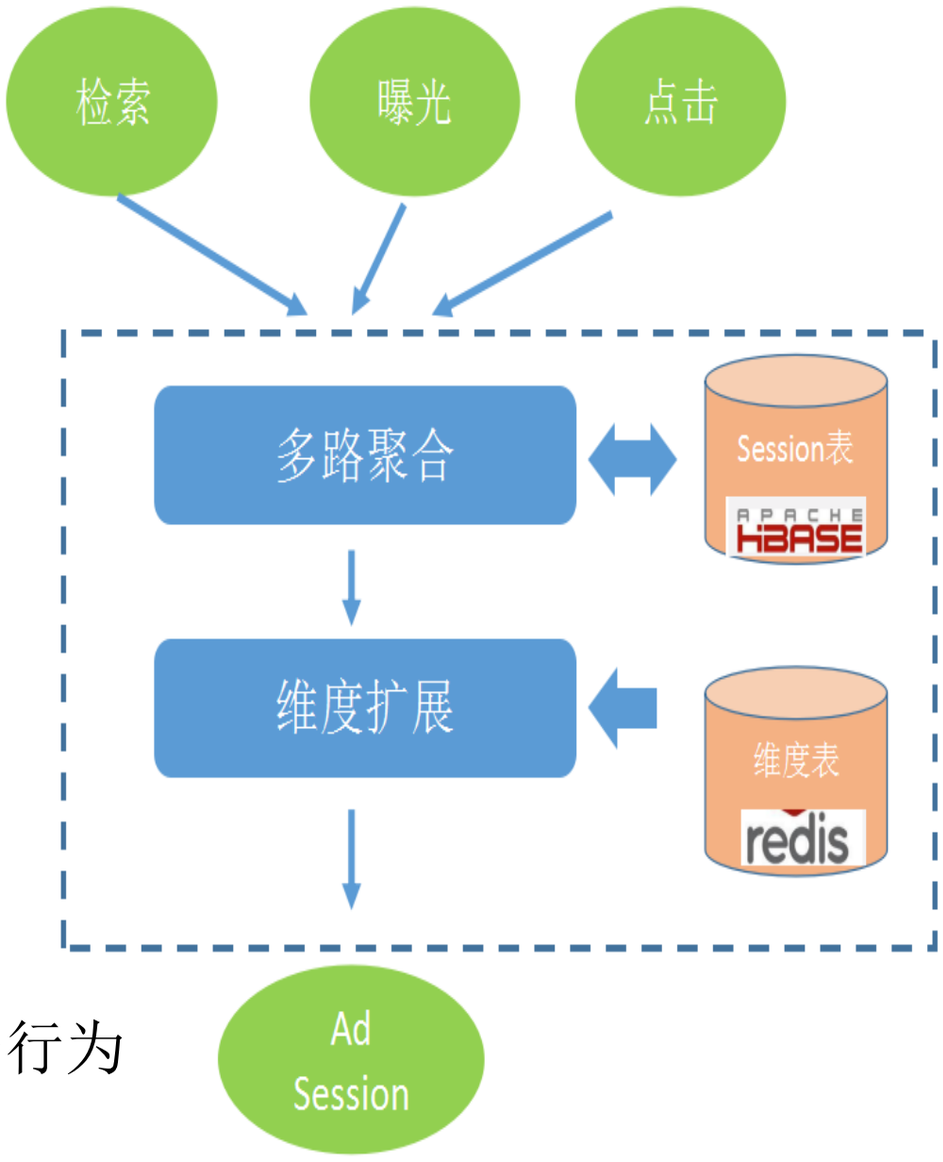
- 业务诉求
  - ✓ 实时CTR预估
  - ✓ 数据一致性和完整性
  - ✓ 实时多维分析

- 实时Session化
  - ✓ **多路聚合（ Log Join）：**

1. 基于session_id聚合检索、曝光、点击
2. **挑战：** 长窗口期状态、数据乱序

- ✓ **维度扩展：**

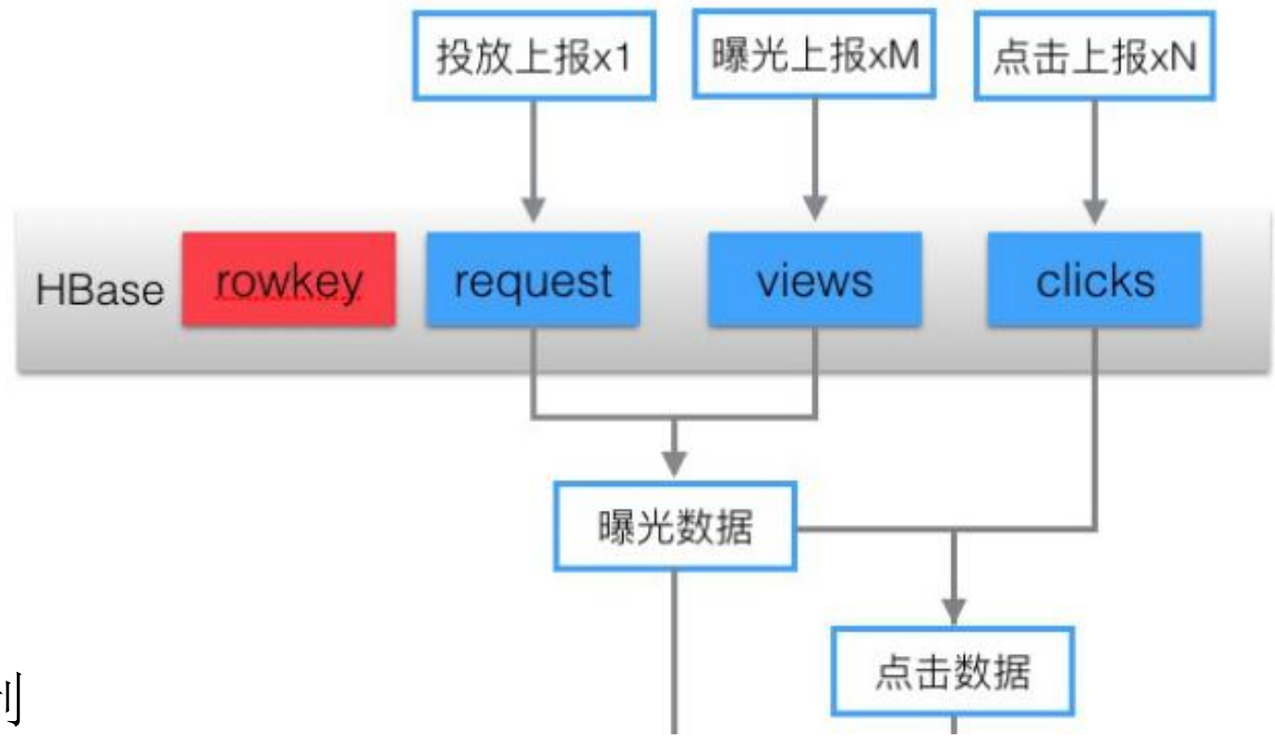
1. 基础业务维度：广告位信息、订单信息等
2. 用户特征：基础属性、即时兴趣、历史广告行为



实时多路聚合

长窗口期状态

- ✓ 有效SESSION、持续长达**一周，**历史状态总量**TB级**
- ✓ 内存方案代价高，采用HBase。
- ✓ 以session_id生成rowkey,　ColumnFamiy: session
- ✓ 实时性如何满足：写优化设计、BlockCache缓存、batch接口

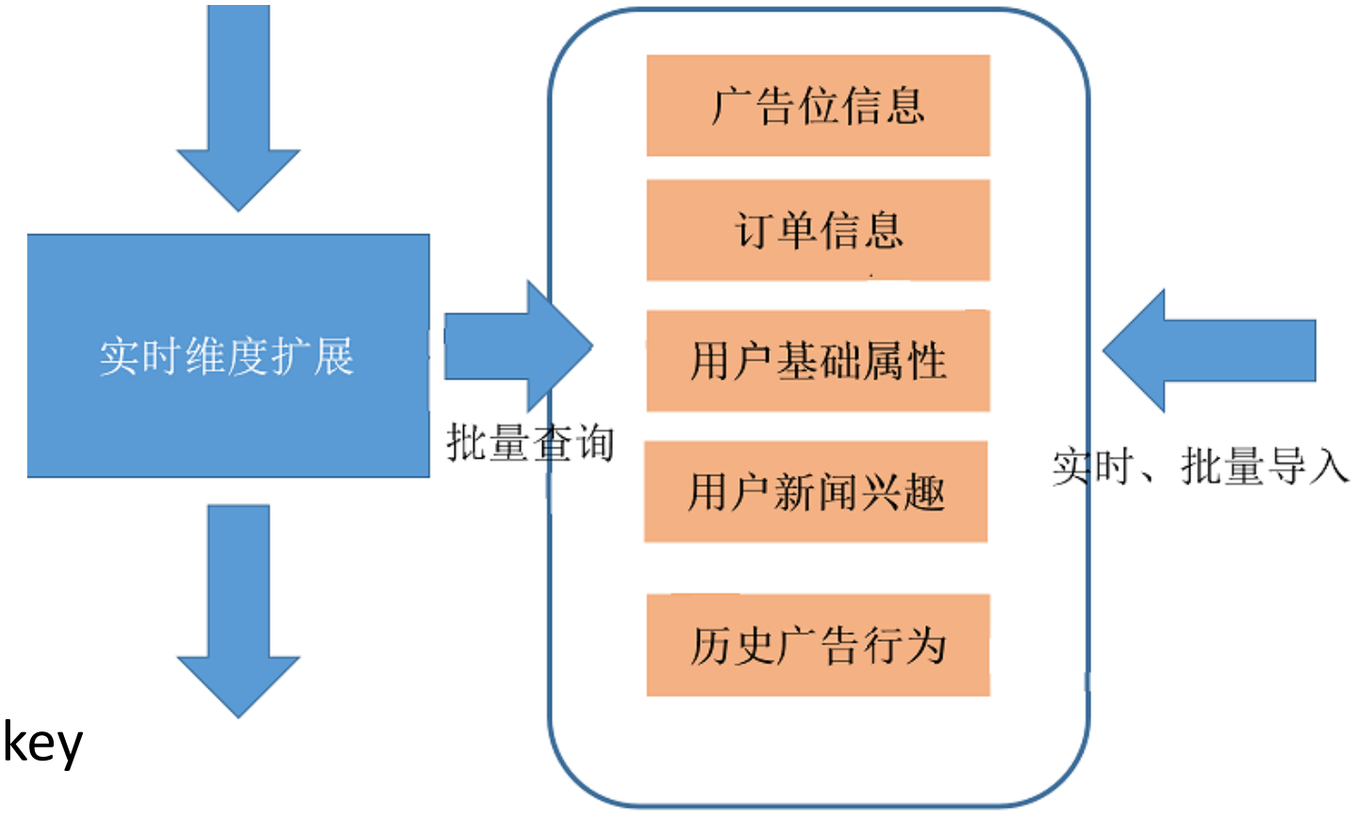


数据乱序问题

- ✓ 数据上报、处理并发机制
- ✓ 触发Join：检索驱动、曝光驱动
- ✓ 原子性保证：HBase rowkey 行级锁、以CheckAndPut方式写入。

实时维度扩展框架

- **维度信息库：**
  - ✓ 业务维度表，用户特征库
  - ✓ 支持实时、批量导入
  - ✓ 统一存储在Redis集群



- **问题：**
  - ✓ 维度间存在依赖关系
  - ✓ 如何满足高吞吐低延时

    平均一条数据需要查询40个key

- **实时维度扩展框架：**
  - ✓ 组件化设计： Transformer
  - ✓ 框架：Transformer调度、批量查询Key、放入Local Dict。
  - ✓ Transformer: 查询Local Dict，补全对应业务维度信息

广告数据查询引擎

- **典型查询case:**

```
select 广告位类型, 频道类型, 订单分类, 平台 ， sum(imp), sum(income)
from video_view_table
group by 广告位类型, 频道类型, 订单分类, 平台
where date between in (start_time,  end_time)
and  m_vsubprogram == '8001'
```

- **看似简单的问题：**

  却衍生出N多解决方案， **Lost in Solutions!**

- **数据规模和服务能力：**
  - ✓ 流量规模： 百万级 -> 百亿级
  - ✓ 聚合的维度：几个 -> 几十 、上百个
  - ✓ 查询响应：小时级 -> 分钟级、秒级

广告数据建模特点

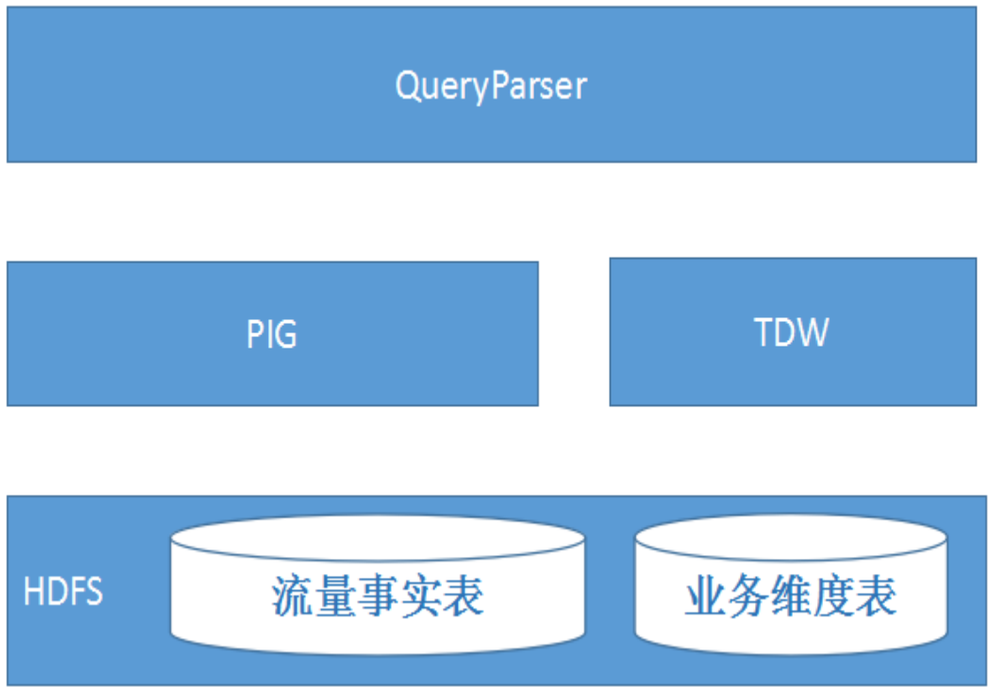
- **数据量级大，分析维度多**
  - ✓ 曝光级数据，单表 **2000亿+** records、历史总量**PB级**
  - ✓ 分析维度**100+**，其中常用维度达**50+**

- **服务需求多样化**
  - ✓ **接口类型**： Json，SQL
  - ✓ **查询类型**：报表型需求、分析型需求
  - ✓ **查询响应**：**秒级~分钟级**，并且越快越好

- **业务数据易变下的一致性**
  - ✓ 品牌合约广告下的**收入指标计算**
  - ✓ 模型数据修正频繁且易错

BASE: 基于MR的方案

- **数据存储层：HDFS**
  - ✓ 流量事实表:
    视频曝光模型、新闻曝光模型等
  - ✓ 业务维度表:
    广告位信息表、订单信息表等
- **查询计算层：**
  - ✓ QueryParser：
    将查询生成pig latin或者HiveSQL
  - ✓ PIG/TDW，基于星型模型Join



- **问题：**
  - ✓ 星型模型Join：
    Replicated Joins -> Memory Bound，对长跨度查询不友好
  - ✓ 查询响应慢，受制集群状态（十几分钟到小时级）
  - ✓ 无法支持交互式查询业务需求（秒级）

引入实时查询引擎

- **数据存储层：HDFS + Parquet列式存储**
  - ✓ **Pre Join:** 大宽表，当前基于常用维度
  - ✓ **预聚合cubes：空间换时间，**

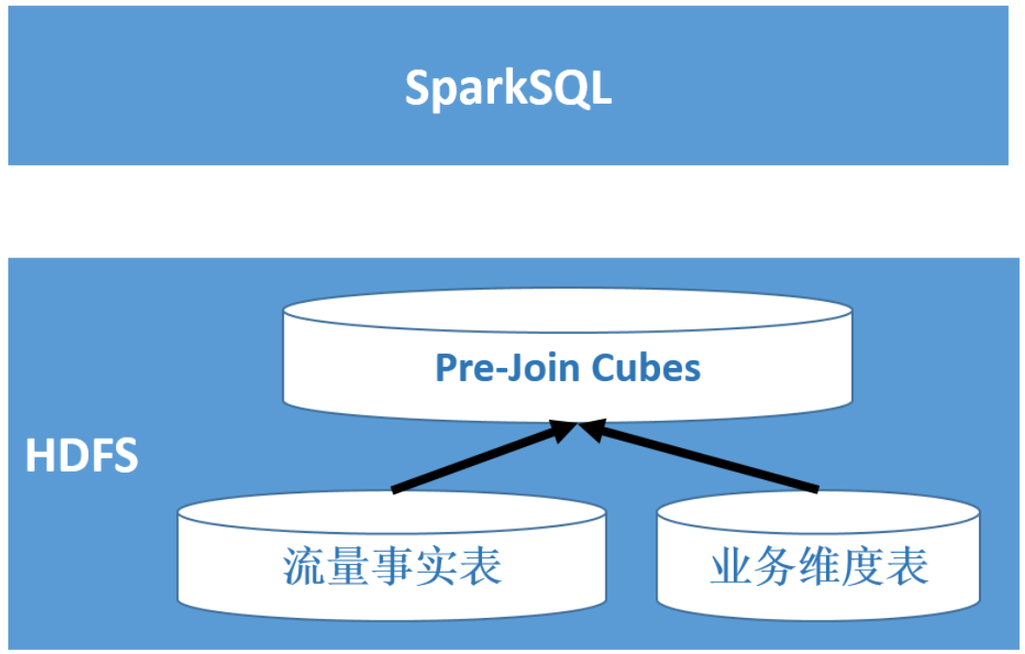
    人工定义：

    1. 报表型需求：通常查询维度固定。
    2. 长时间跨度查询：按时间聚合月表、年表

    智能生成： 基于查询日志，以覆盖率为目标，生成频繁项集组合。

    效果：1%成本，3倍查询性能提升

- **查询计算层：SparkSQL**
  - ✓ 多轮SQL迭代，减少数据落地。
  - ✓ 非Filter模式下，考验Reduce能力。
  - ✓ 兼容性强，便于扩展。



- **新的问题：**
  - ✓ Cubes管理，智能选择机制
  - ✓ Pre-Join下的数据一致性问题
  - ✓ 任务隔离机制

一次查询的优化流程

- **举个Query例子:**

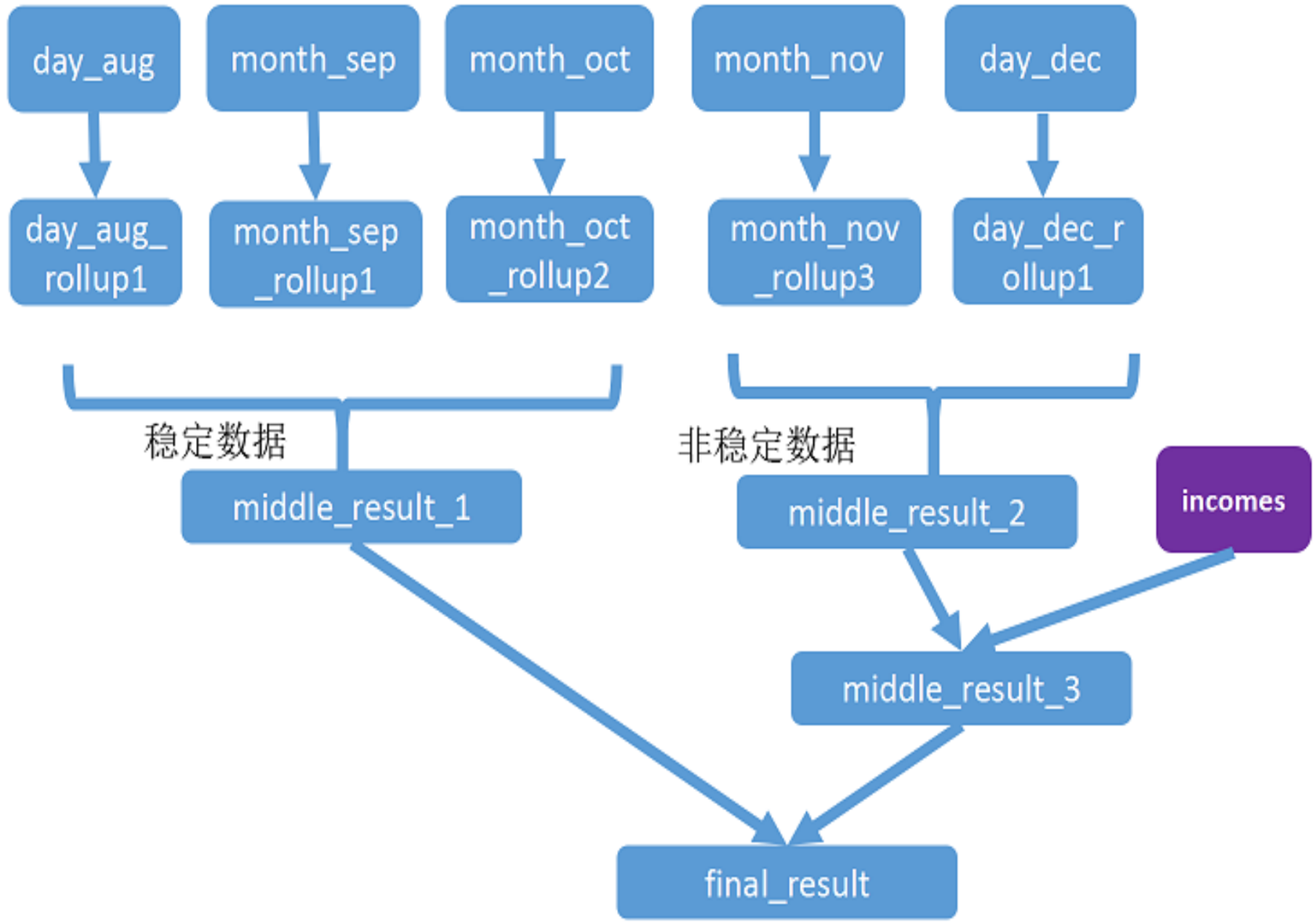
查询2016.08.17至2016.12.02，子频道为8001，不同广告位、订单、平台下的曝光量、收入总量

Step1: 时间范围选分区表

Step2: 选择中间表（cube）

Step3: 判定是否稳定数据

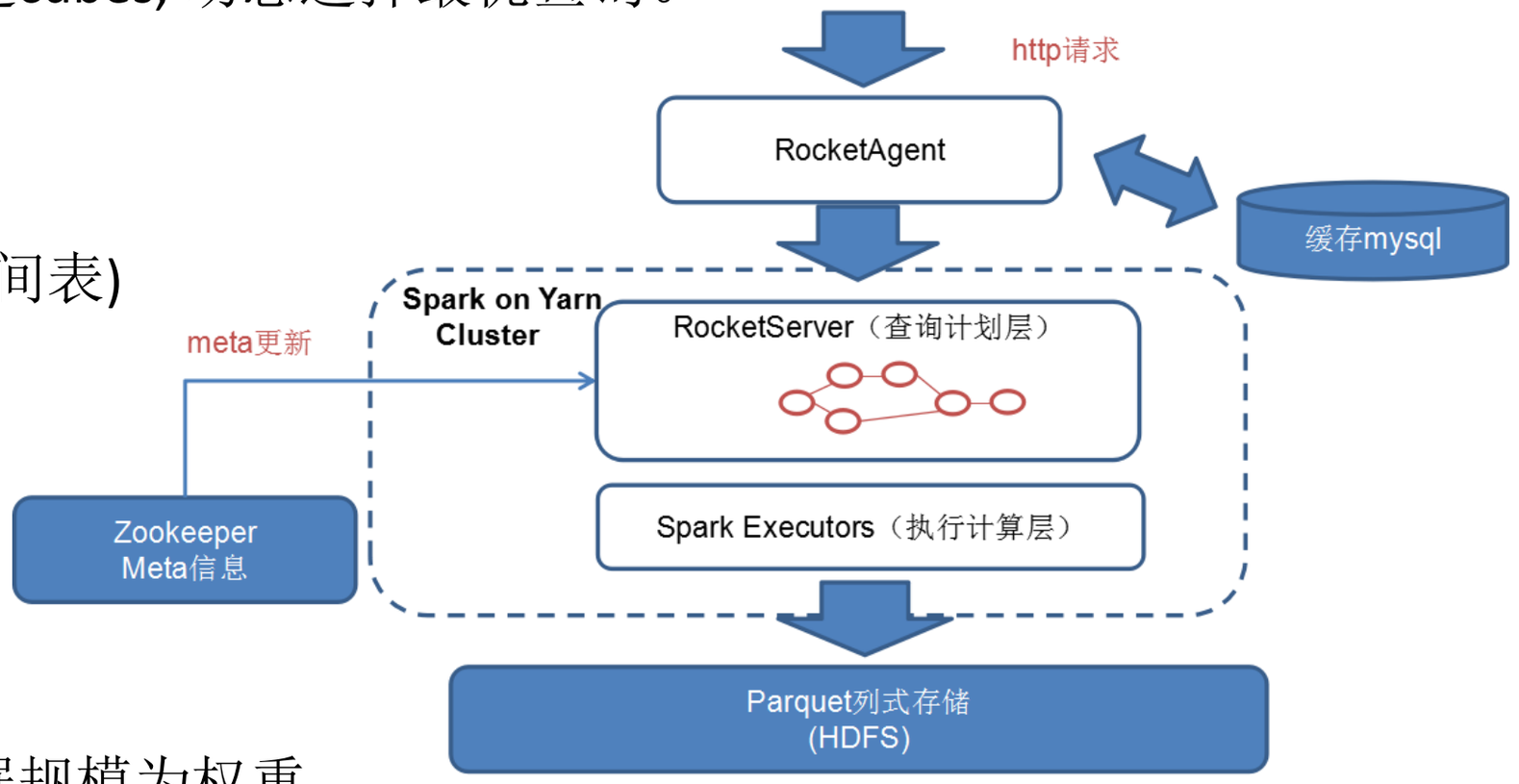
Step4: union, group by



系统实现

- **建设思路：**
  - ✓ 以SparkSQL作为基础查询层, parquet + HDFS作为存储层。
  - ✓ 自建元数据层，离线构建cubes, 动态选择最优查询。

- Meta：存储于Zookeeper
  - ✓ TableFamily(分区表，中间表)
  - ✓ ViewFamily(虚拟视图表)
  - ✓ BroadCastTable(缓存表)



- **在线查询：**
  - ✓ Cubes选择，以Cube数据规模为权重
  - ✓ 一致性：稳定期和非稳定期

    非稳定期数据：基于缓存BroadcastTable, 实时Join。
  - ✓ 隔离机制： fairscheduler pools + pool 内 FIFO、线程隔离

THANKS

ArchSummit 全球架构师峰会 2016

[北京站]

主办方 Geekbang 极客邦科技 InfoQ