Tencent 腾讯

腾讯看点基于Flink+ClickHouse的实时数据系统实践

汇报人：王展雄
腾讯看点-高级工程师
时　间：2020-10-12

可解决的痛点：实时数据分析系统

推荐： 10分钟前，我们上了一个推荐策略，想知道在不同人群推荐效果怎么样？

运营： 我想知道，现在在广东省的用户中，最火的广东地域内容是哪些？

审核： 我想看到，过去5分钟，按照举报数、曝光数降序的游戏类内容列表？

作者： 我想了解，今天到目前为止，我的内容收到了多少个点赞和评论？

老板： 我想了解，过去10分钟有多少用户在看点浏览过内容？

……

0-调研

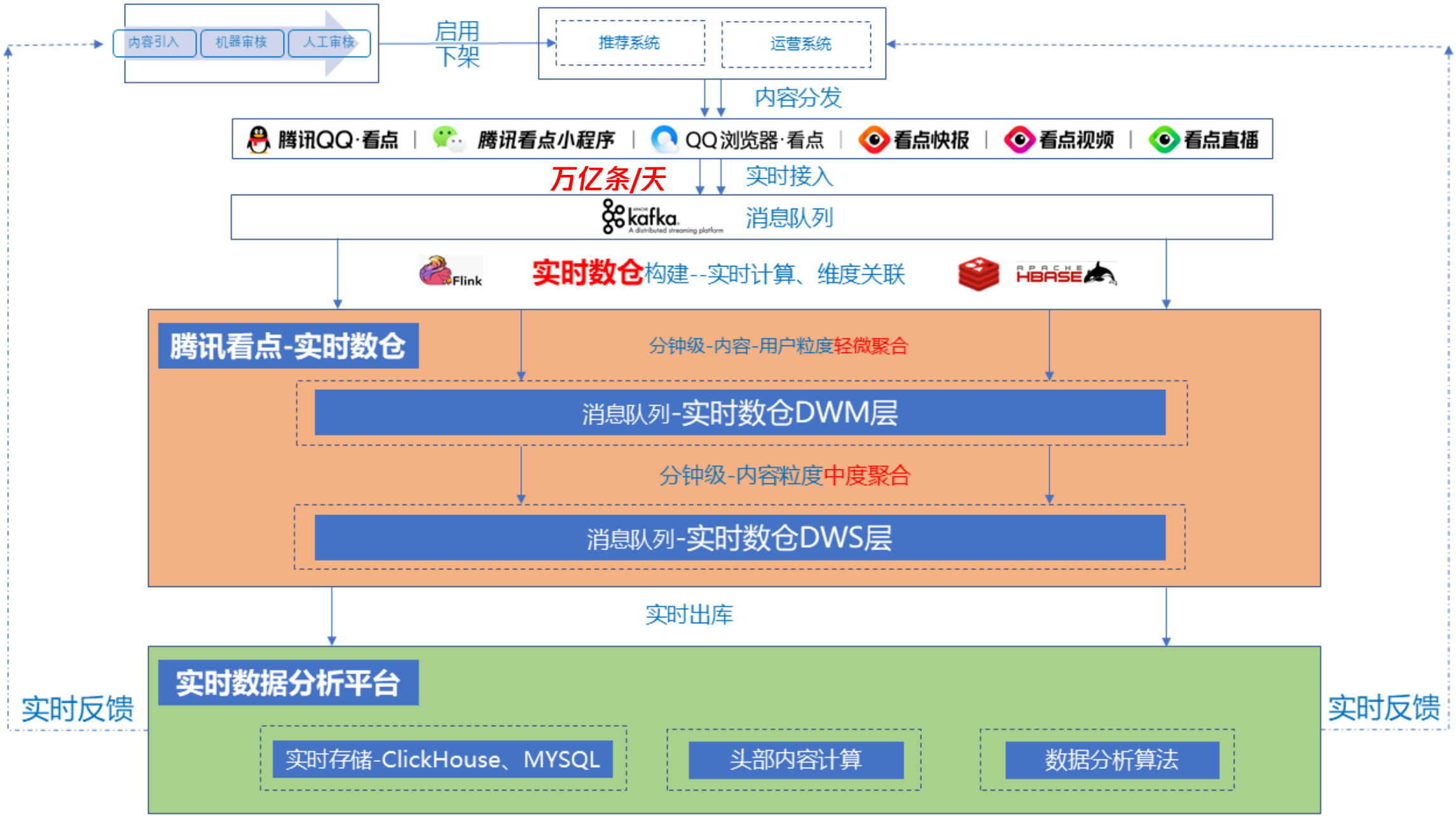
离线数据分析平台-无法满足

实时数据分析平台-准实时无法满足

方案选型

架构设计

项目背景：腾讯看点数据中心-实时数据分析系统



实时数据分析系统-方案选型

- 实时数仓：

大规模数据处理架构	业界使用	灵活性	容错性	成熟度	迁移成本	批/流处理代码
Lambda架构	高	高	高	高	低	2套
Kappa架构	低	低	低	低	高	1套

- 实时计算引擎：

实时计算引擎	准确性	容错机制	延时	吞吐量	易用性	业界使用
Flink	Exactly-once	Checkpoint轻量级快照	低	高	高	高
SparkStreaming	Exactly-once	RDD Checkpoint	中	高	中	中
Storm	At-least-once	Acker重试	低	低	低	低

- 实时存储引擎：

存储引擎	维度索引	高并发	预聚合	高性能查询	特性
ClickHouse	有	高	有	快	海量数据批量写入、实时多维聚合OLAP查询
Ckv/HBase	无	高	无	慢	KV读写、多维聚合查询效率极低
Tdsql/Tidb	有	低	无	慢	适合OLTP
ES	有	快	无	慢	全文检索适合日志查询，运营分析，实时聚合查询慢
Druid	有	高	单粒度	中	计算全局TopN只能是近似值

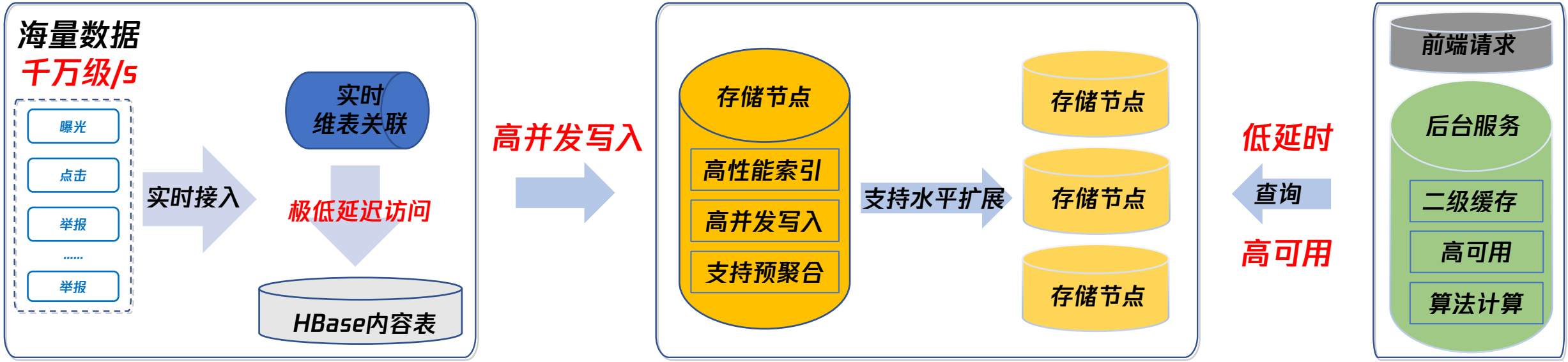
实时数据分析系统-设计目标与设计难点

设计目标：

1. 实时计算引擎（实时数仓构建）
2. 实时存储引擎
3. App层（后台服务和前端展示层）

难点：

- 千万级/s的海量数据如何实时接入，并且进行极低延迟的实时维表关联
- 高并发写入、高可用分布式和高性能索引查询



实时数据分析系统-架构设计

前端页面 Ant Design of React

页面请求

后台服务 腾讯自研RPC后台服务框架

后台请求 1分钟内存缓存

实时数仓

原始消息队列

队列拆分：MQ1、MQ2、MQ3

微队列：曝光、点击、互动、...

Flink 实时计算：Redis缓存、HBase内容表

消息队列数仓DWM层

Flink 实时计算：分钟-内容ID-C侧数据-B侧数据粒度聚合

消息队列数仓DWS层

实时存储

Apache ZooKeeper™

Replic1 Replic2

Hash

写

读

后台请求

ClickHouse集群

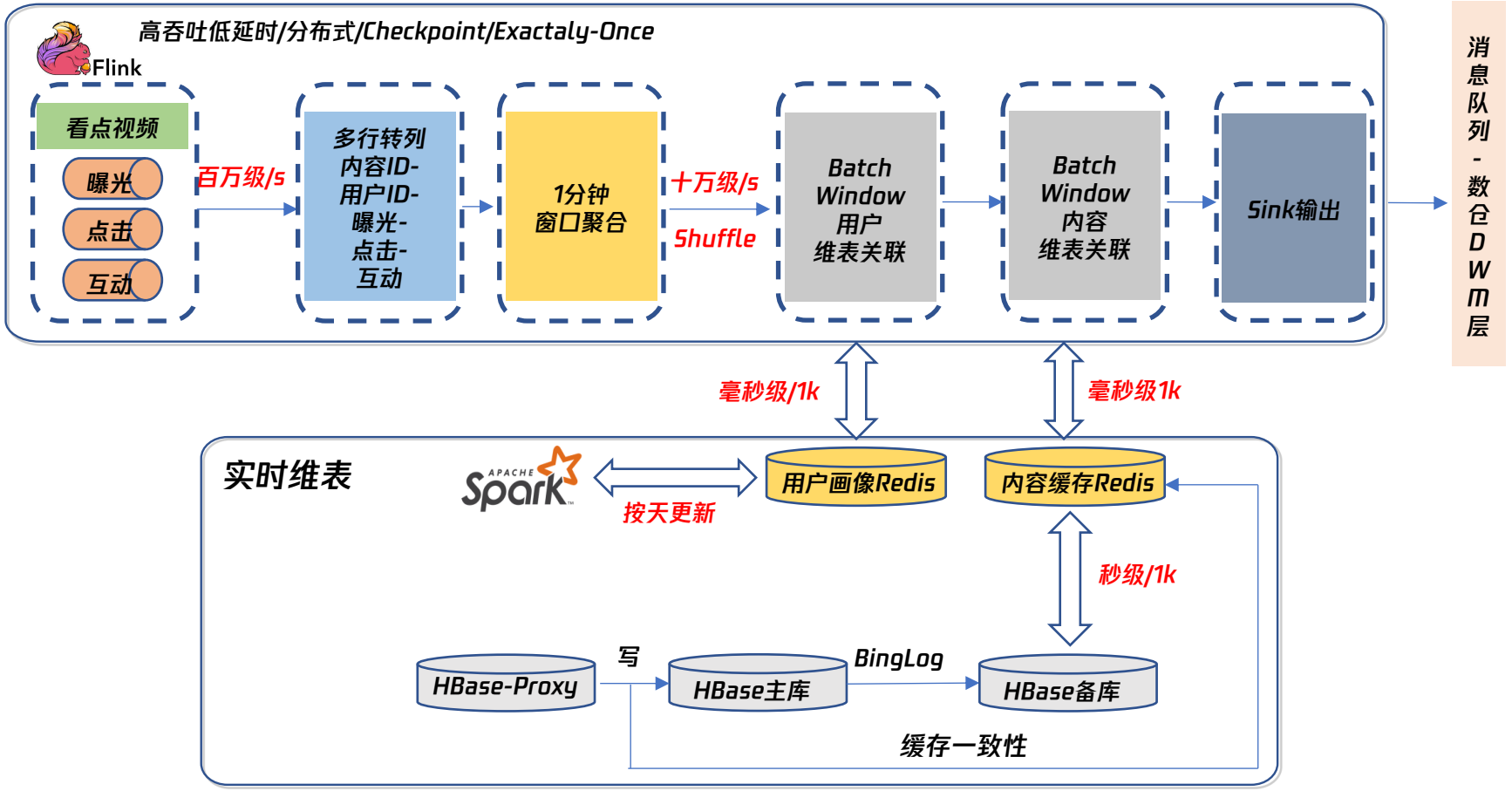
接入层	计算层	实时数仓存储层	实时写入层	OLAP存储层	接口层

01-实时计算

实时关联

实时数仓

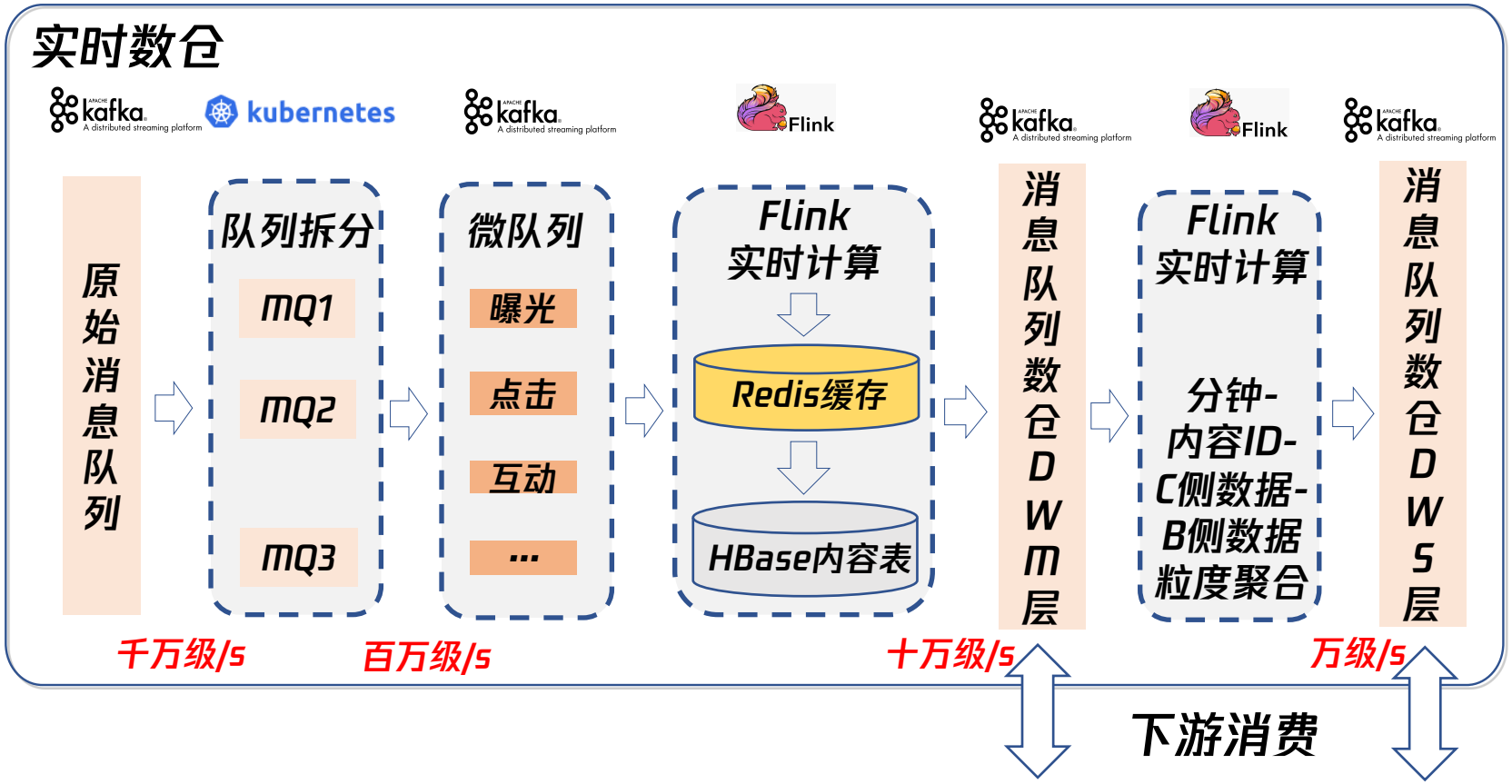
实时高性能维表关联



难点：1分钟的实时数据流(百万级/s)直接关联HBase需要小时级耗时	
解决方案：	耗时：
✓ Flink行转列/聚合	小时级->十几分钟
✓ Batch+Redis缓存	十几分钟->秒级
✓ 上游过滤一些不存在的赛道内容ID防止缓存穿透	秒级->秒级
✓ 消峰填谷，防止雪崩	秒级->数十秒

指标	优化前	优化后	节省
时间	小时级	数十秒	90%+
数据量	百亿级	十亿级	

实时数仓建设-向下游提供服务



50个消费者	建设实时数仓前	建设实时数仓后	节省
Flink(核心)	X1*50	X1	1个月节省上百万
Redis(内存)	X2*50	X2	减少98%
接入人力成本（天）	X3*50	X3	
接入实时数据门槛	高(涉及模块多、有门槛)	低（Flink SQL）	

02-实时存储

一. 分布式-高可用

- 方案选择
- 水平扩展

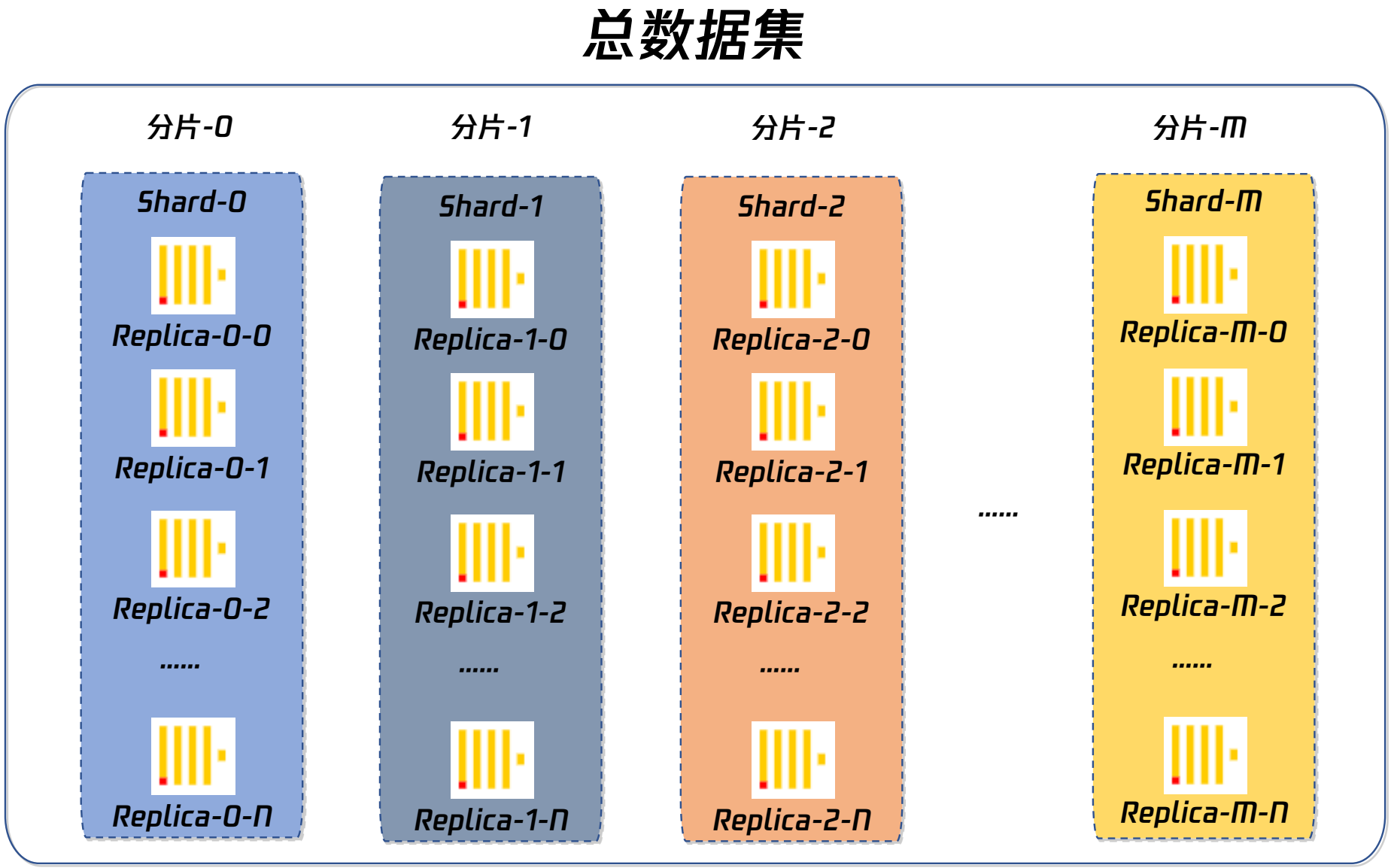
二. 海量数据-写入

- 磁盘做Raid加大磁盘IO
- Batch写入，减少Merge压力
- 分布式集群构建、分摊写入压力

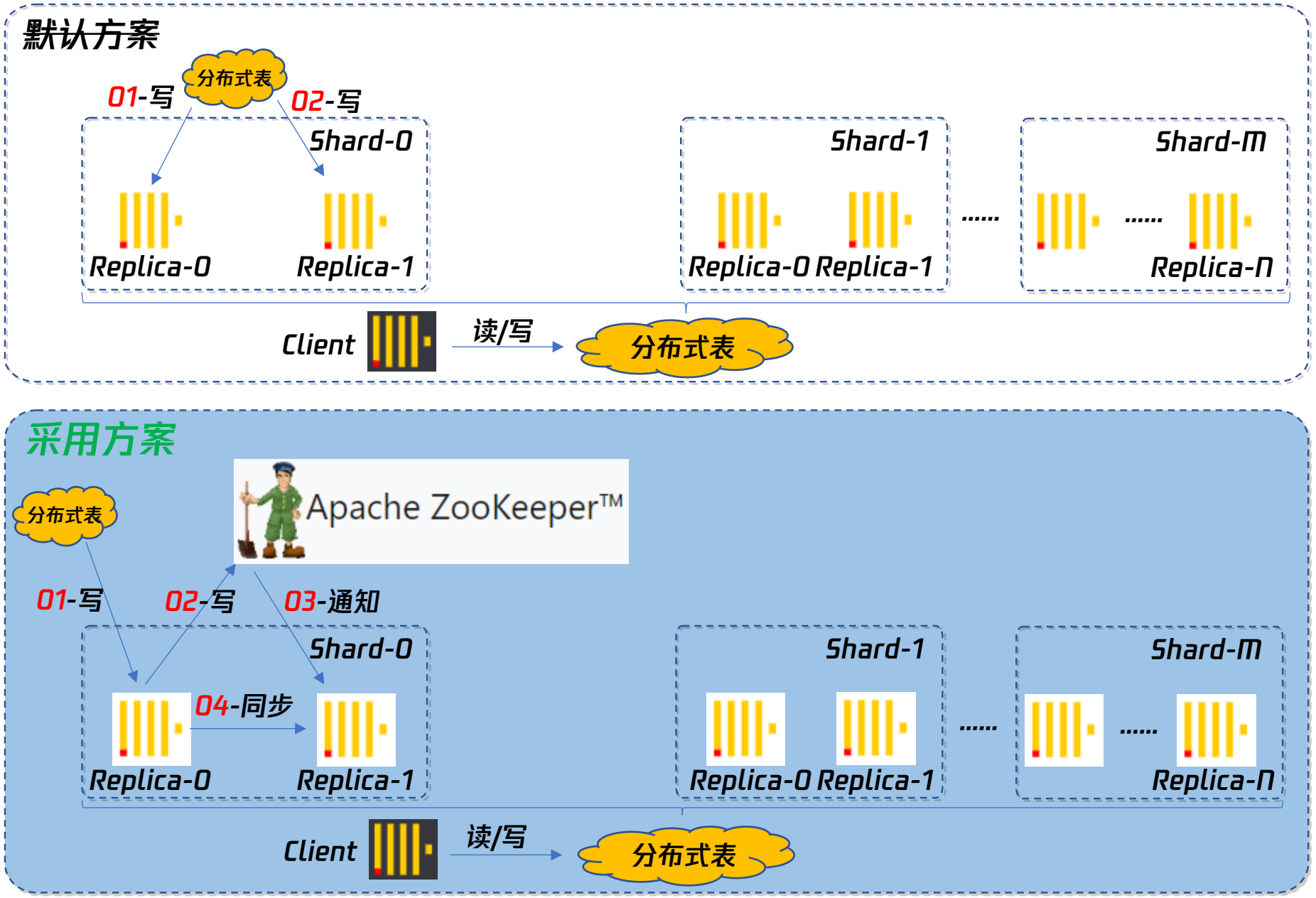
三. 高性能-查询

- 索引构建
- 物化视图构建
- 分布式集群构建并行查询

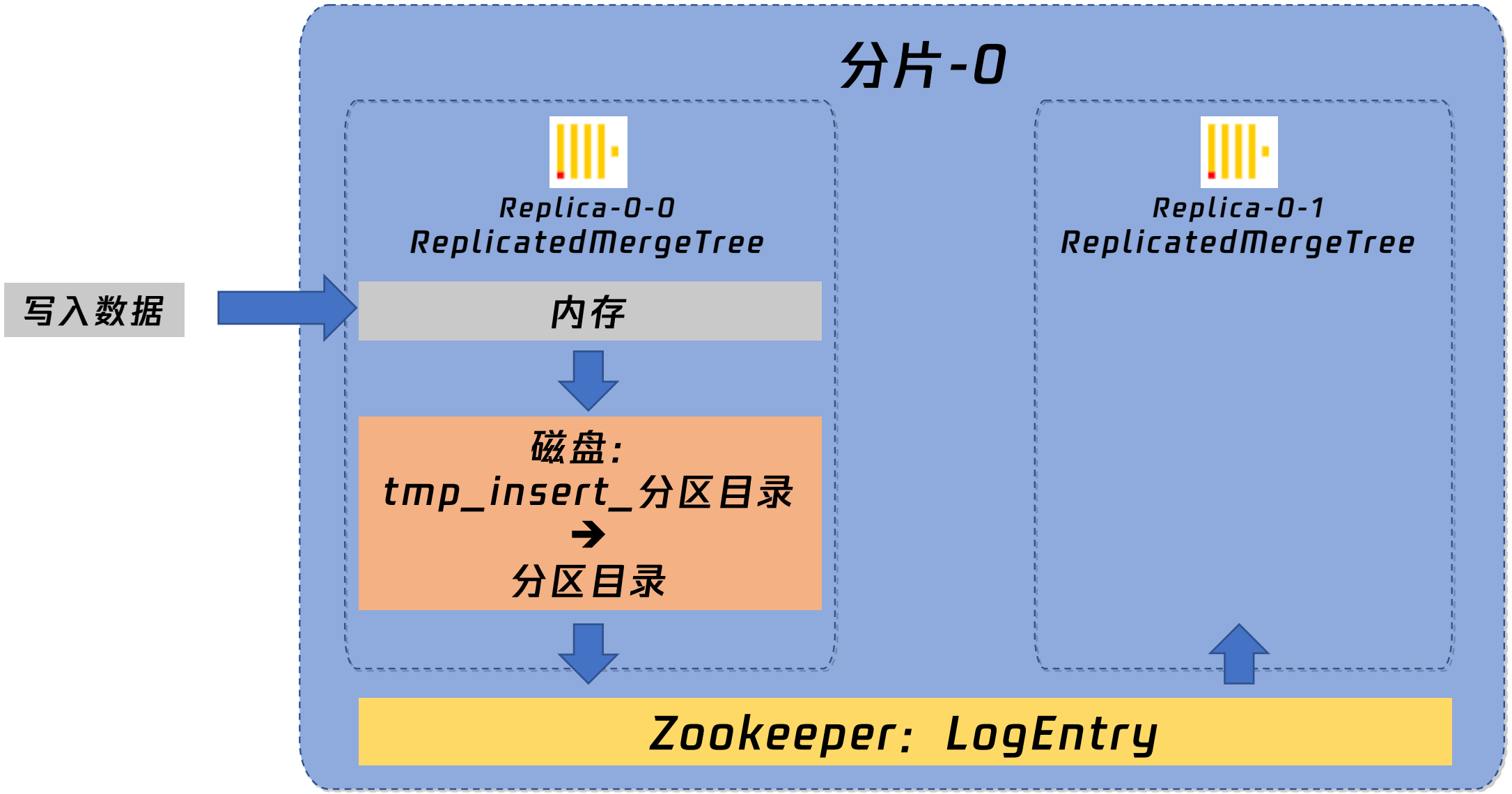
实时存储-分布式-高可用


总数据集
分片-0
分片-1
分片-2
分片-M
Shard-0
Replica-0-0
Replica-0-1
Replica-0-2
.....
Replica-0-N
Shard-1
Replica-1-0
Replica-1-1
Replica-1-2
.....
Replica-1-N
Shard-2
Replica-2-0
Replica-2-1
Replica-2-2
.....
Replica-2-N
.....
Shard-M
Replica-M-0
Replica-M-1
Replica-M-2
.....
Replica-M-N

实时存储-分布式-高可用


默认方案
分布式表
01-写
02-写
Shard-0
Shard-1
Shard-M
Replica-0
Replica-1
Replica-0 Replica-1
Replica-N
Client
读/写
分布式表
采用方案
分布式表
Apache ZooKeeper™
01-写
02-写
03-通知
04-同步
Shard-0
Shard-1
Shard-M
Replica-0
Replica-1
Replica-0 Replica-1
Replica-N
Client
读/写
分布式表

实时存储-分布式-高可用-原理



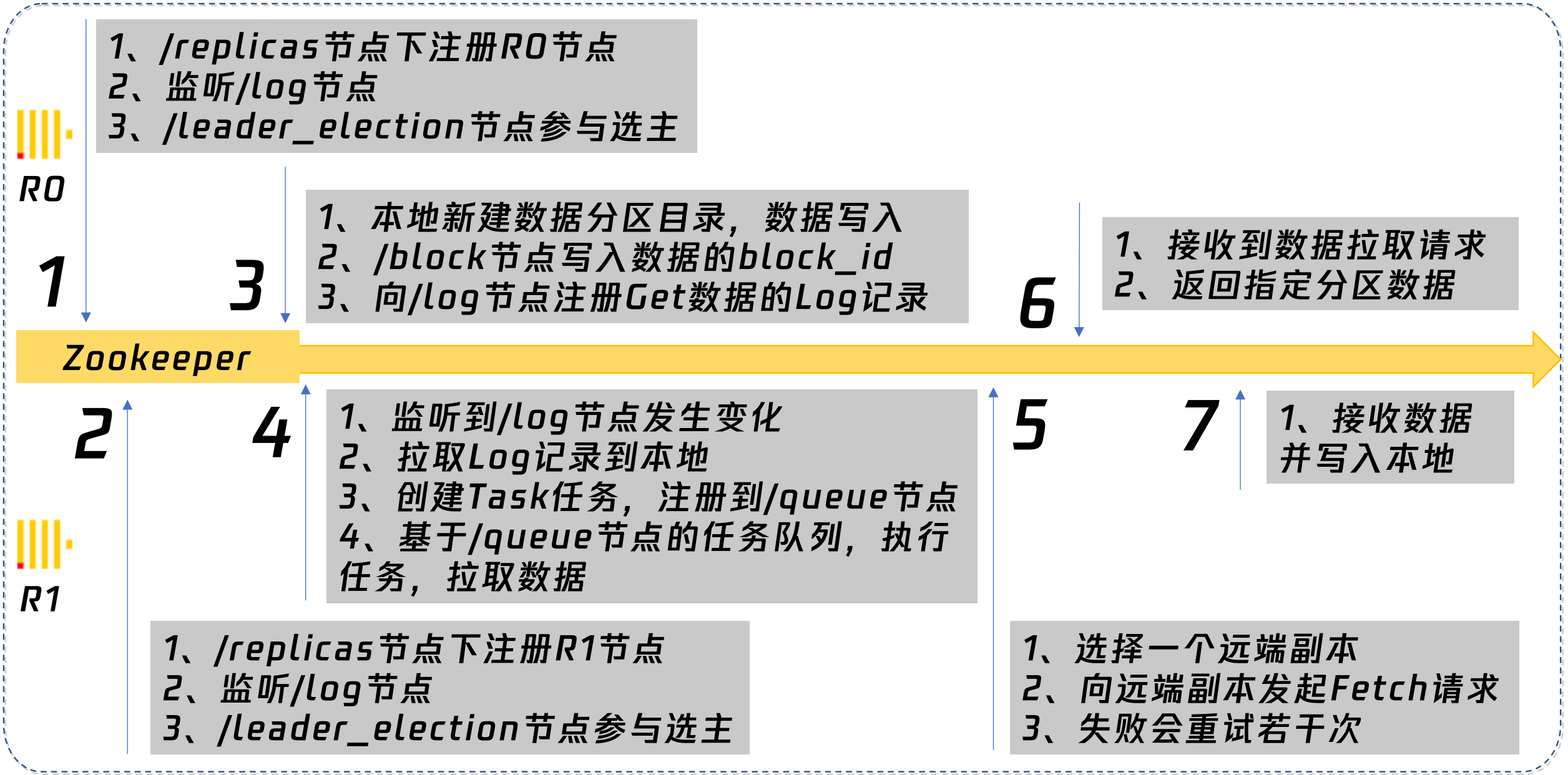
实时存储-分布式-高可用-原理

元数据相关：
1、/metadata节点，保存元数据信息，包括主键、分区键等
2、/columns节点，保存字段信息，包括列名和数据类型
3、/replicas节点，保存副本相关信息

校验相关：
4、/leader_election节点，用于主备选举
5、/blocks节点，防止数据块重复，以及数据同步

执行相关：
6、/log节点，记录了副本协同需要执行的Log记录，包含任务执行源信息
7、/queue节点，包含具体执行的操作任务

实时存储-分布式-高可用-原理



实时存储-海量数据-写入-问题1

```
|-- 20200924_1926431_1926431_0
|-- 20200924_1926432_1926432_0
|-- 20200924_1926433_1926433_0
|-- 20200924_1926434_1926434_0
|-- 20200924_1926434_1926438_1
|-- 20200924_1926434_1926452_2
|-- 20200924_1926435_1926435_0
|-- 20200924_1926436_1926436_0
|-- 20200924_1926437_1926437_0
|-- 20200924_1926438_1926438_0
|-- 20200924_1926439_1926439_0
|-- 20200924_1926439_1926443_1
|-- 20200924_1926440_1926440_0
|-- 20200924_1926441_1926441_0
|-- 20200924_1926442_1926442_0
|-- 20200924_1926443_1926443_0
|-- 20200924_1926444_1926444_0
|-- 20200924_1926444_1926448_1
|-- 20200924_1926445_1926445_0
|-- 20200924_1926446_1926446_0
|-- 20200924_1926447_1926447_0
|-- 20200924_1926448_1926448_0
|-- 20200924_1926449_1926449_0
|-- 20200924_1926450_1926450_0
|-- 20200924_1926451_1926451_0
|-- 20200924_1926452_1926452_0
|-- 20200924_1926453_1926453_0
|-- 20200924_1926454_1926454_0
|-- 20200924_1926455_1926455_0
`-- detached
```

MaxBlockNum

20201012_1_2_1

分区ID MinBlockNum 合并Level

20201012_1_1_0 20201012_2_2_0

合并

20201012_1_2_1

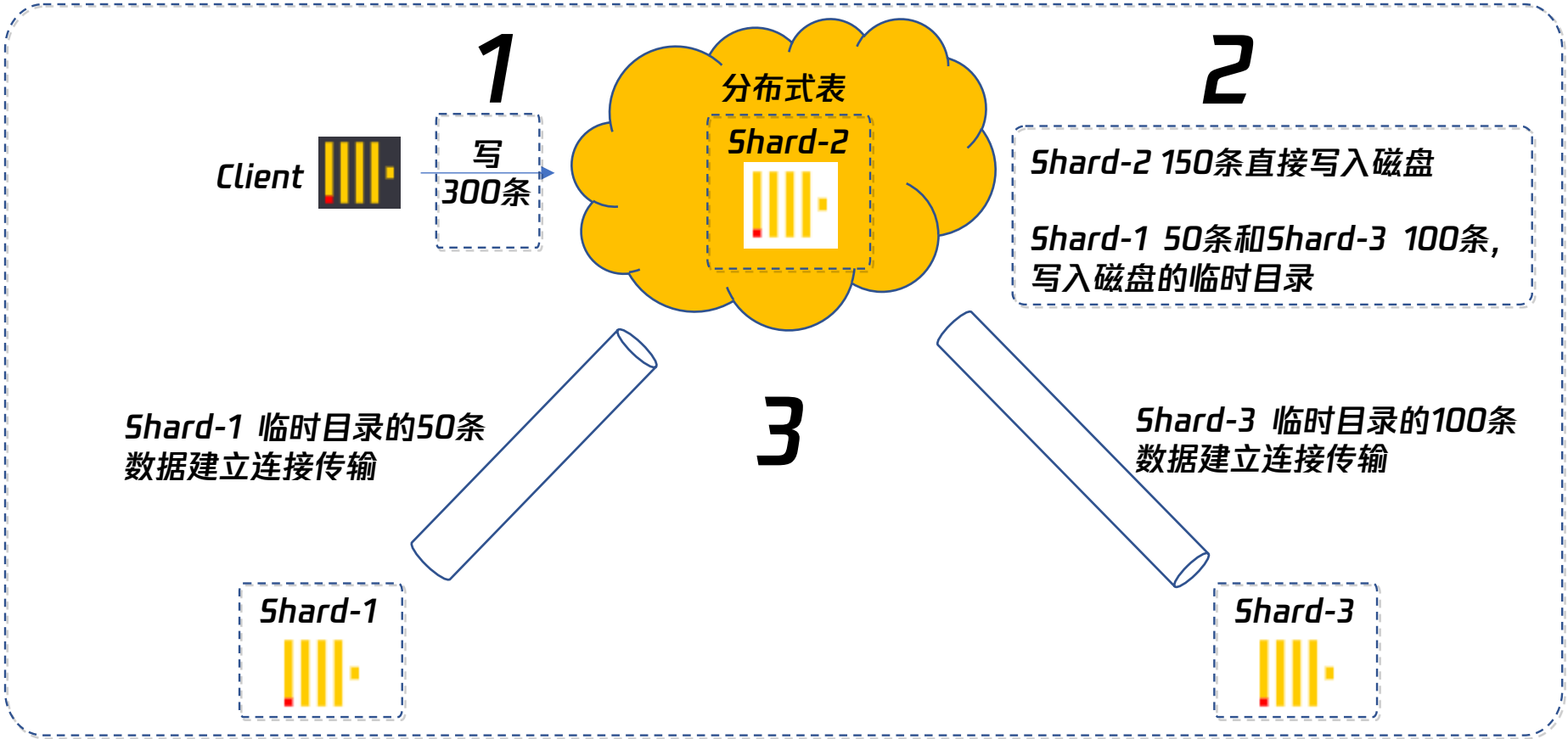
写入问题：

1、写入QPS太高

解决方案：

1、几十万行的Batch写入

实时存储-海量数据-写入-问题2



写入问题：

1、写入QPS太高、ZK崩溃

2、写入出现单点问题，磁盘打满

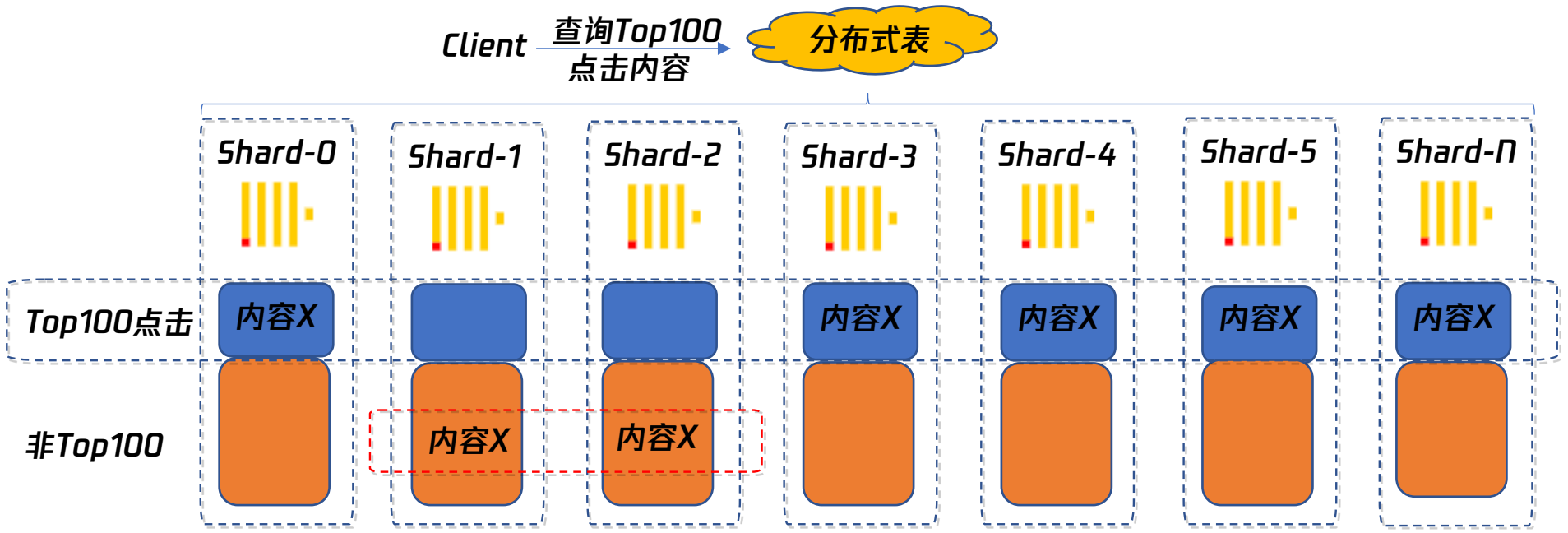
解决方案：

1、几十万行的Batch写入

2.1、对磁盘做Raid

2.2、在写入之前进行分表，直接写分布式表

实时存储-海量数据-写入-问题3




```
SELECT rowkey, sum(pv) AS pv
FROM table_all
WHERE f_date = '2020-10-12'
GROUP BY rowkey
LIMIT 100

Limit
 Expression
  Expression
   MergingAggregated
    Union
     ParallelAggregating
      Expression x 12
       Filter
        MergeTreeThread
     Remote x 7
```

写入问题：

1、写入QPS太高、ZK崩溃

2、写入出现单点问题，磁盘打满

3、 Local Top非全局Top的问题

解决方案：

1、几十万行的Batch写入

2.1、对磁盘做Raid
2.2、在写入之前进行分表，直接写分布式表

3、写入之前进行Hash路由，将同一个内容ID的记录路由到同一个分片上

实时存储-高性能-查询-优化1

```
Whole data:     [---------------------------------------------]
CounterID:      [aaaaaaaaaaaaaaaaabbbbcdeeeeeeeeeeeeefgggggggghhhhhhhhhiiiiiiiiiklllllll]
Date:           [11111122222223333123321111122222233321111111212222222311111223311122333]
Marks:           |     |     |     |     |     |     |     |     |     |     |
                a,1   a,2   a,3   b,3   e,2   e,3   g,1   h,2   i,1   i,3   l,3
Marks numbers:   0     1     2     3     4     5     6     7     8     9     10
```

CounterID IN ('a', 'h') 且 Date = 3，服务器读取marks范围在[1, 3)和[7, 8)之间的数据

CounterID in ('a', 'h')，服务器读取marks范围在[0, 3)和[6, 8)之间的数据

Date = 3，服务器读取marks范围在[1, 10)之间的数据

查询优化：

1、日期FDate、分钟TimeOneMinute、内容ID建立稀疏索引

实时存储-高性能-查询-优化2

```
CREATE MATERIALIZED VIEW 物化视图
ENGINE = SummingMergeTree()
PARTITION BY toYYYYMMDD(f_date)
ORDER BY ( f_date, rowkey )
AS
SELECT f_date, rowkey, sumState(pv) AS SumPv
FROM 原始流水表
GROUP BY f_date, rowkey
```

原始流水数据量：Billion级

相差1000倍+

物化视图数据量：Million级

查询优化：

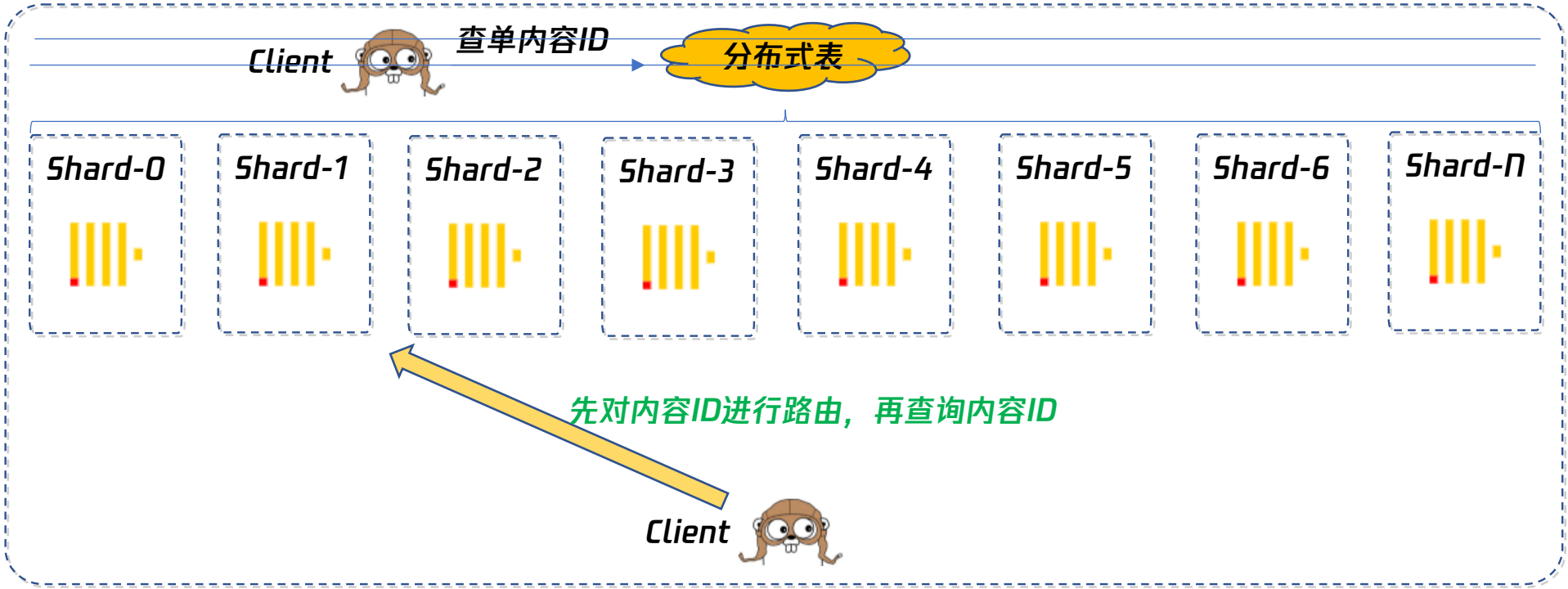
1、日期FDate、分钟TimeOneMinute、内容ID建立稀疏索引

2、针对不同的维度，建立对应的预聚合，物化视图，空间换时间

实时存储-高性能-查询

分布式表查询问题：

- 分布式表会把请求发送N个Shard上并行计算，然后数据汇总，再返回
- 但是只有一个Shard是包含内容ID的



我们的优化：

- 后台查询之前对内容ID进行路由，直接访问目标Shard，减少N-1/N的负载，大量缩短查询时间，提升用户体验

ClickHouse文件存储：[列].bin 和 [列].mrk

Marks标记序号	压缩块在.bin文件中的偏移量	解压后Marks标记的偏移量
0	0	0
1	0	8192
2	0	16384
3	0	24576
4	0	32768
5	0	40960
6	0	49152
7	0	57344
8	12016	0
9	12016	8192
.....	.....	.....

.mrk标记文件示意图

ClickHouse高性能原因:

1、多核CPU并行计算

2、SIMD并行计算加速

3、分布式水平扩展集群

4、稀疏索引、列式存储、数据压缩

5、聚合分析优化（Group by）

.....

腾讯看点数据中心-实时数据分析系统-系统成果

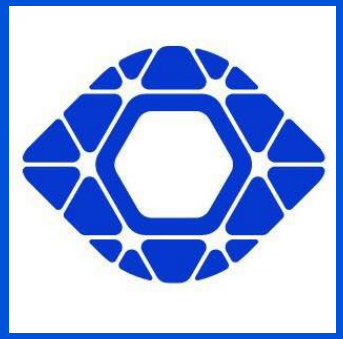
腾讯看点实时数据仓库：DWM层和DWS层

腾讯看点多维实时数据分析系统：亚秒级响应多维条件查询请求

- 未命中缓存情况下：
  - 过去30分钟内容的查询，99%的请求耗时在1秒内
  - 过去24小时内容的查询，90%的请求耗时在5秒内，99%的请求耗时在10秒内

Thanks

欢迎关注腾讯看点技术微信公众号



Tencent 腾讯

Q&A

欢迎关注腾讯看点技术微信公众号