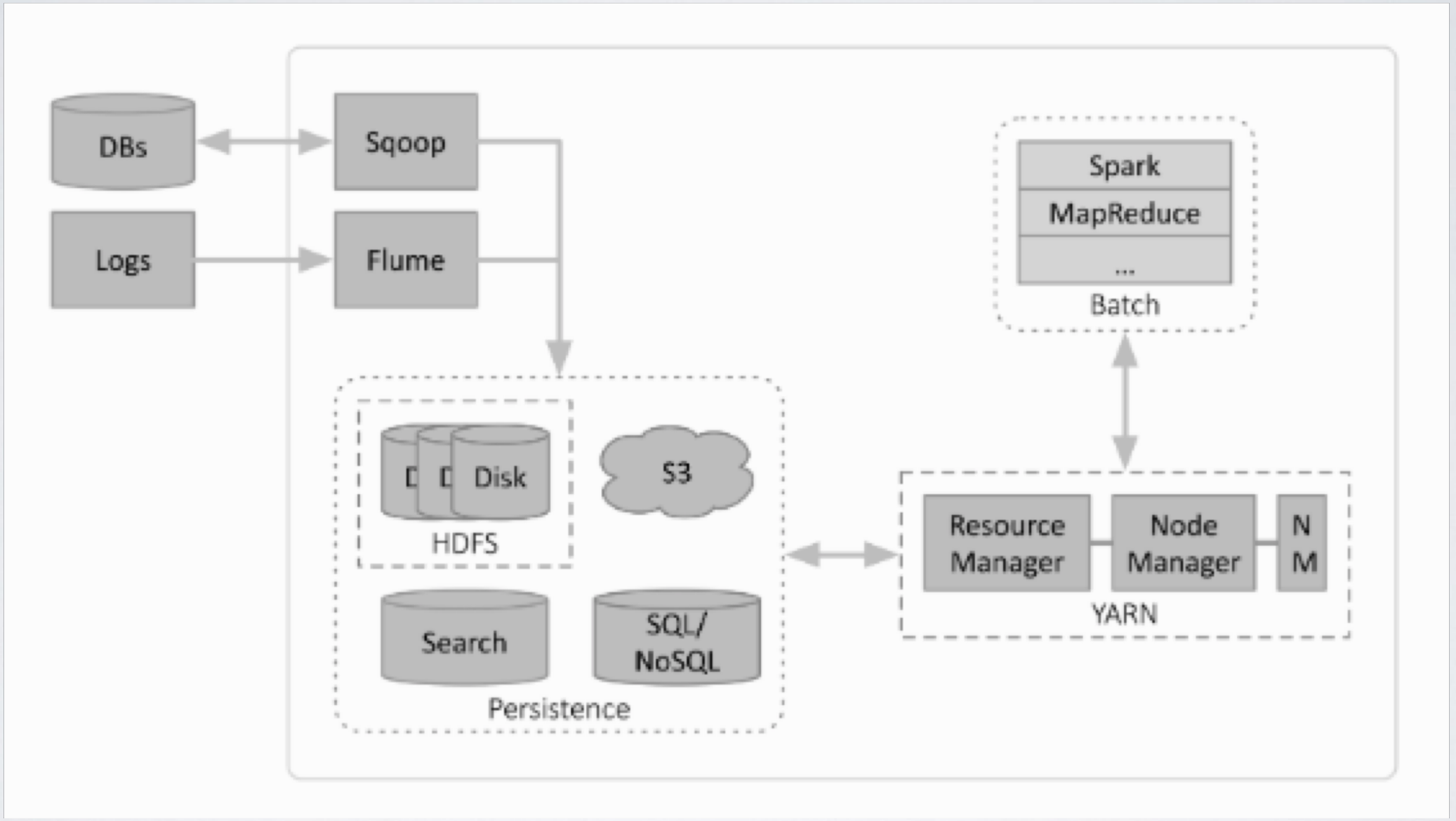
大数据流式计算介绍及应用

宦传建@大数据平台组

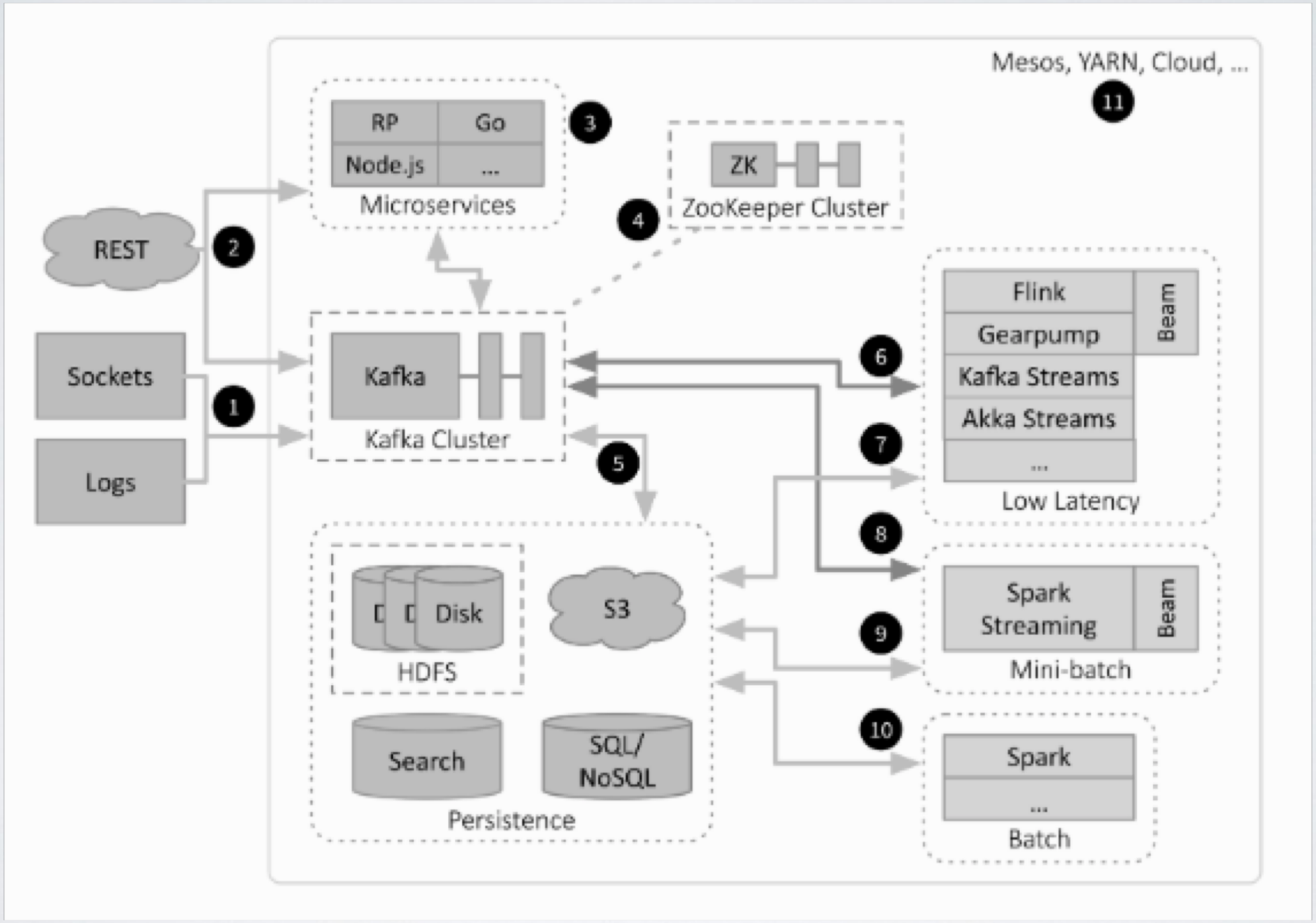
内容介绍

- 应用埋点服务架构介绍
- 埋点采集 - Flume
- 埋点 Pub/Sub - Kafka
- 埋点计算 - Spark Streaming
- 埋点计算实现 - Scala

批量计算架构


DBs
Logs
Sqoop
Flume
Spark
MapReduce
...
Batch
Disk
S3
HDFS
Search
SQL/
NoSQL
Persistence
Resource
Manager
Node
Manager
N
M
YARN

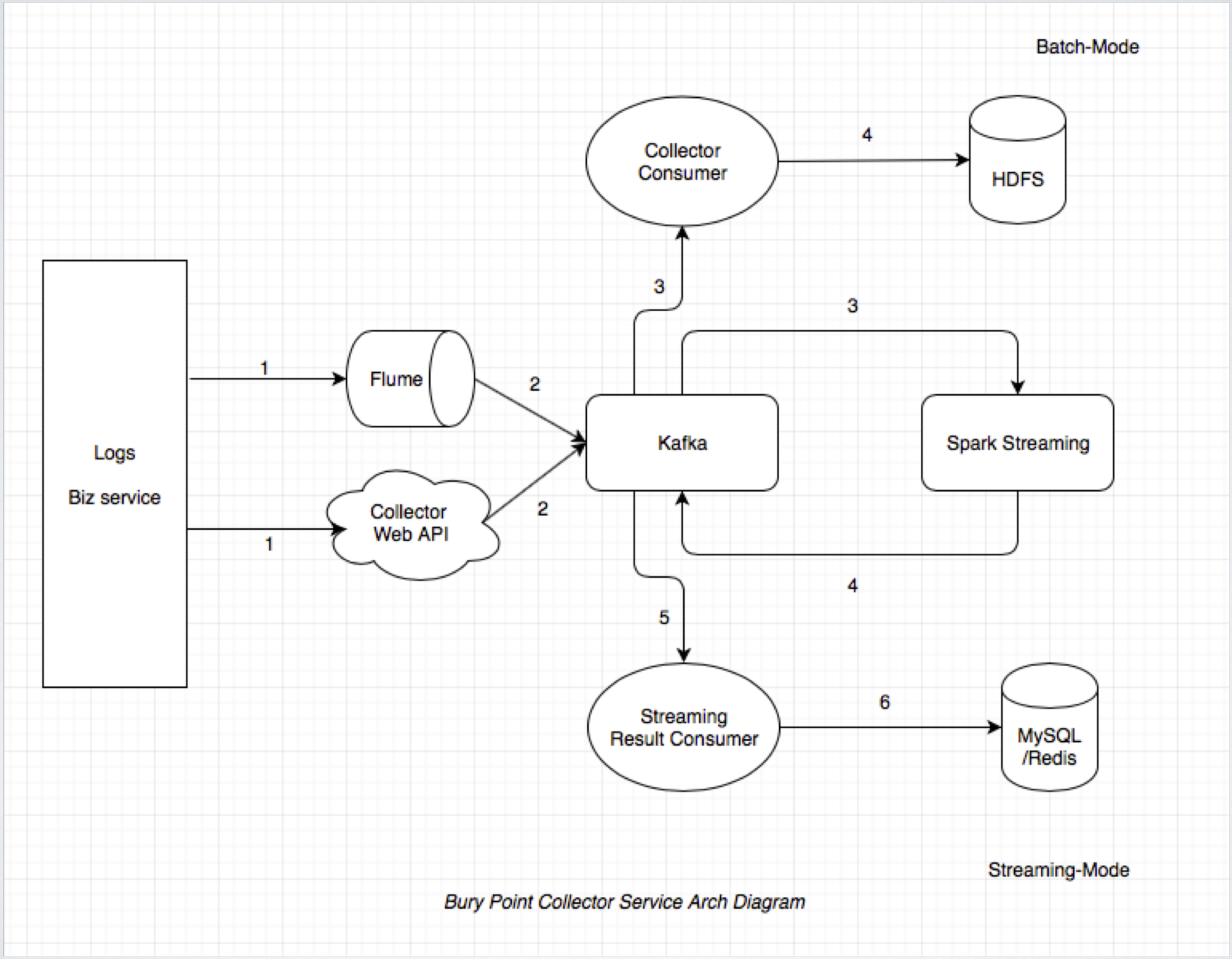
流式计算架构


Mesos, YARN, Cloud, ...
11
RP
Go
Node.js
...
Microservices
3
ZK
ZooKeeper Cluster
4
REST
2
Sockets
1
Logs
Kafka
Kafka Cluster
5
6
7
8
9
10
Flink
Gearpump
Kafka Streams
Akka Streams
...
Beam
Low Latency
Disk
S3
HDFS
Spark Streaming
Beam
Mini-batch
Search
SQL/ NoSQL
Persistence
Spark
...
Batch

Batch-Mode vs Streaming

Metric	Sizes and units: Batch	Sizes and units: Streaming
Data sizes per job	TB to PB	MB to TB (in flight)
Time between data arrival and processing	Many minutes to hours	Microseconds to minutes
Job execution times	Minutes to hours	Microseconds to minutes

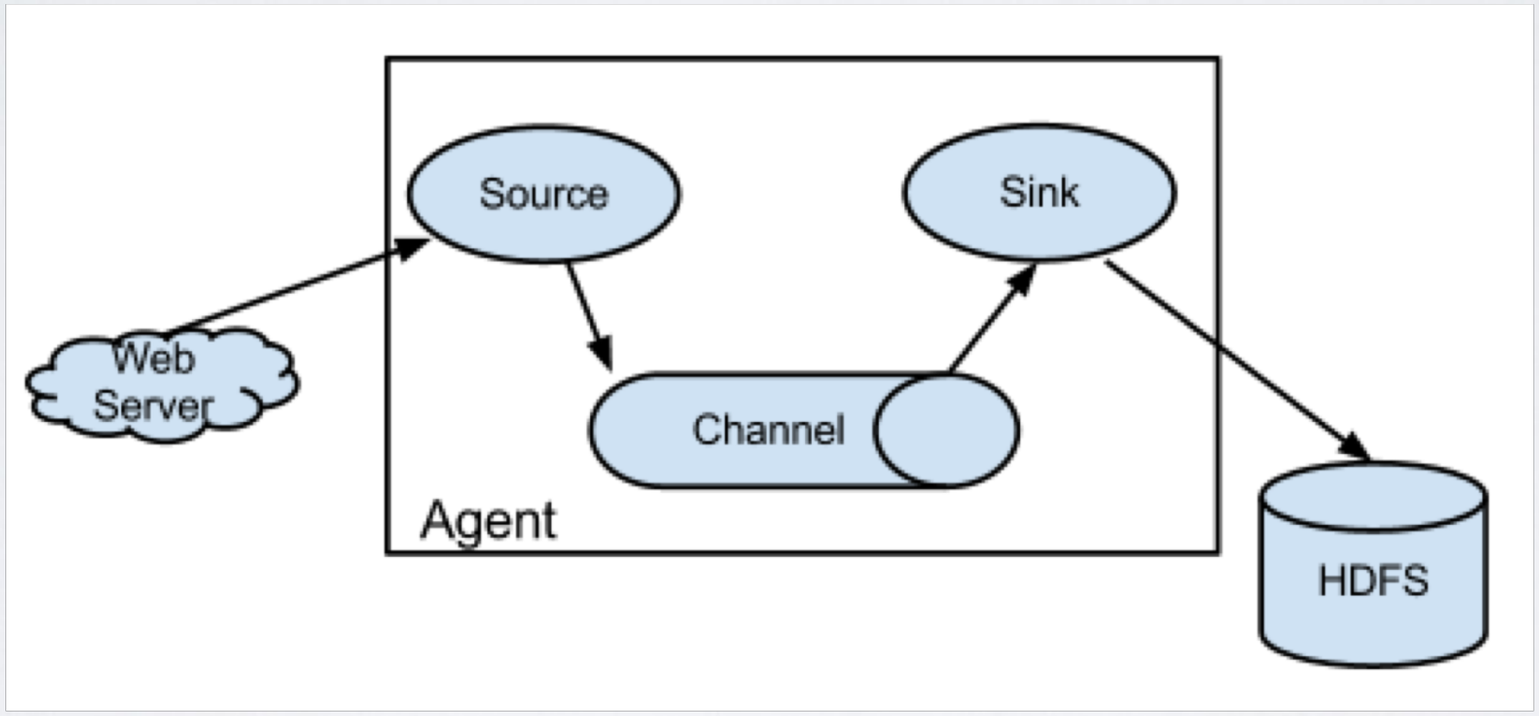
应用埋点服务架构



Bury Point Collector Service Arch Diagram

埋点采集 - Flume

分布式海量日志采集、聚合系统

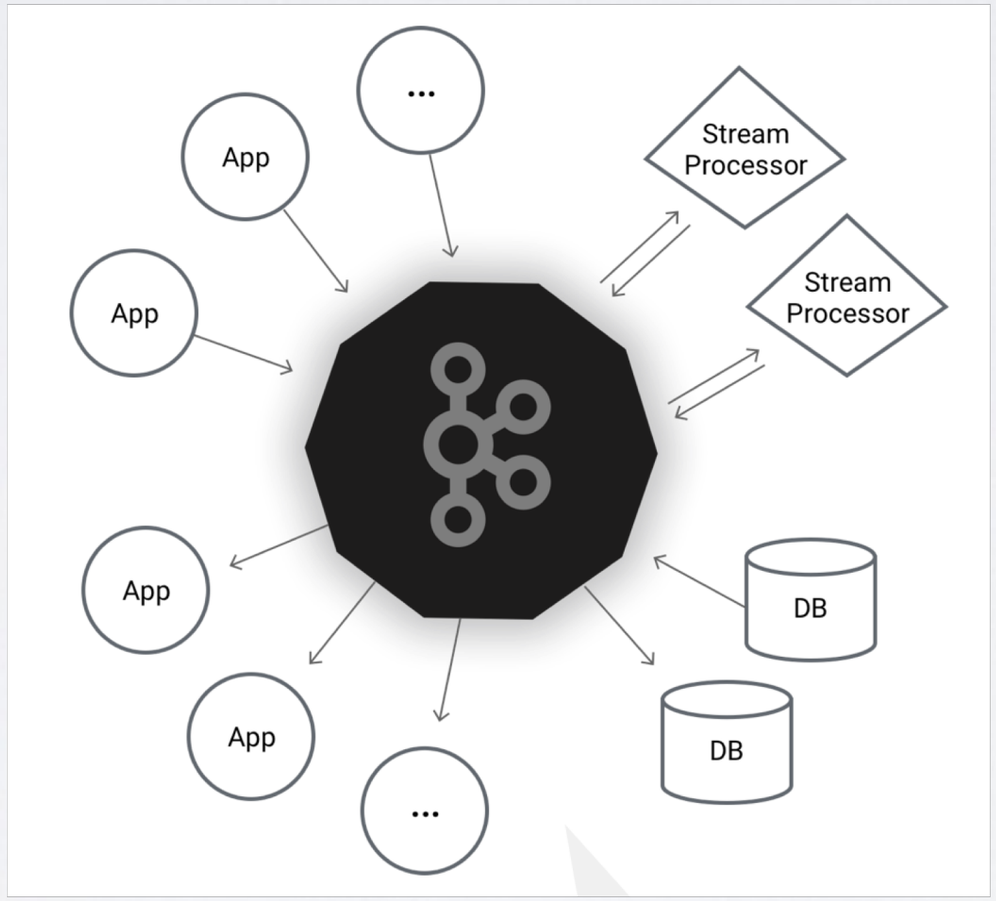


埋点采集 - Flume - 配置

```
[deploy[redacted] data-collector]$ cat flume-agent-config/actAgentProd01.conf
actAgentProd01.sources = actLogSource
actAgentProd01.sinks = kafkaSink
actAgentProd01.channels = actChannel

## Source definition
actAgentProd01.sources.actLogSource.type = TAILDIR
actAgentProd01.sources.actLogSource.positionFile = /opt/data-collector/flume/actAgentProd01-taildir_position.json
actAgentProd01.sources.actLogSource.filegroups = f1
actAgentProd01.sources.actLogSource.filegroups.f1 = /mnt/activityDataProd01/.*.log
actAgentProd01.sources.actLogSource.headers.f1.headerKey1 = value1
actAgentProd01.sources.actLogSource.fileHeader = true

## Regex interceptor to indicate kafka send message's key
actAgentProd01.sources.actLogSource.interceptors = i1
actAgentProd01.sources.actLogSource.interceptors.i1.type = regex_extractor
actAgentProd01.sources.actLogSource.interceptors.i1.regex = \"actId\":(.*?),
actAgentProd01.sources.actLogSource.interceptors.i1.serializers = s1
actAgentProd01.sources.actLogSource.interceptors.i1.serializers.s1.name = key

## Sink definition
actAgentProd01.sinks.kafkaSink.type = org.apache.flume.sink.kafka.KafkaSink
actAgentProd01.sinks.kafkaSink.kafka.topic = activity
actAgentProd01.sinks.kafkaSink.kafka.bootstrap.servers = [redacted]
actAgentProd01.sinks.kafkaSink.kafka.flumeBatchSize = 20
actAgentProd01.sinks.kafkaSink.kafka.producer.acks = -1
actAgentProd01.sinks.kafkaSink.kafka.producer.linger.ms = 1
actAgentProd01.sinks.kafkaSink.kafka.producer.compression.type = snappy

## Channel definition
actAgentProd01.channels.actChannel.type = file
actAgentProd01.channels.actChannel.checkpointDir = /opt/data-collector/flume/actAgentProd01-fileChannelCheckpoint
actAgentProd01.channels.actChannel.dataDirs = /opt/data-collector/flume/actAgentProd01-fileChannelData

## Connect
actAgentProd01.sources.actLogSource.channels = actChannel
actAgentProd01.sinks.kafkaSink.channel = actChannel
```

埋点 Pub/Sub - Kafka

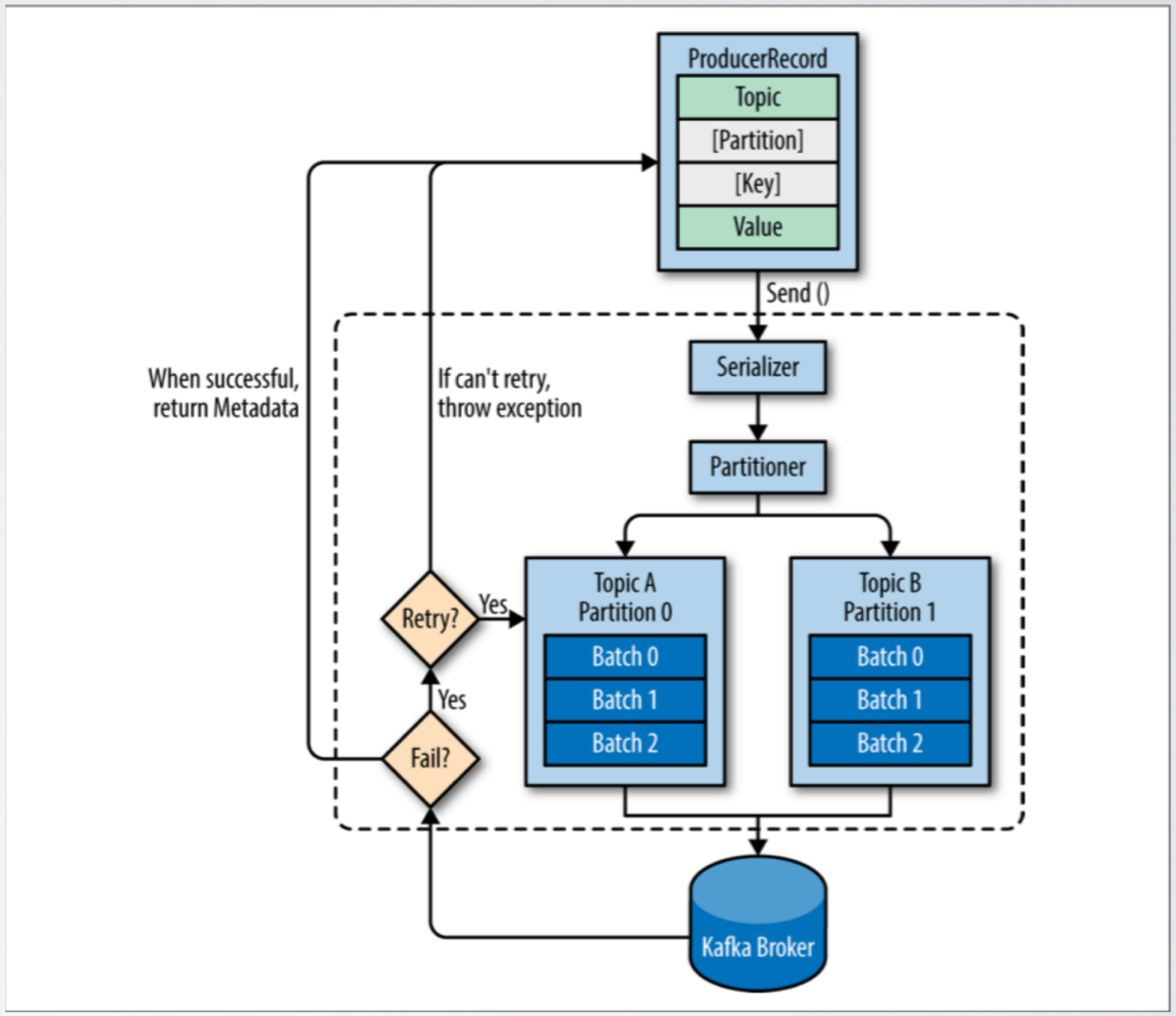
A distributed streaming platform



Kafka - Producer

- Kafka 对单个 topic 使用分区机制，生产者使用缓冲机制，收集尽可能多的数据，并进行压缩，通过牺牲一定的延时来换取更好的吞吐量，减少网络请求。

- 生产者可以通过指定 Key 达到相同的 Key 写入同一个分区，partitionNo = Hash(Key) % TopicPartitionNum



Kafka - Partition

- 分区是生产者、消费者最小的并行单位，创建一个 topic 时允许指定多个分区，topic 的分区数量只能增加，修改分区数，会发生 partition rebalance，操作要注意

- 为了容错，每个分区都会以副本的方式复制到多个 broker 上，形成一个 Leader 和多个 Follower，由 Leader 负责所有的读写请求，Follower 仅从 Leader 同步数据，出现故障时会重新选举。集群内多个 broker 会均摊 Leader 分布，所以整个集群对外是负载均衡的。

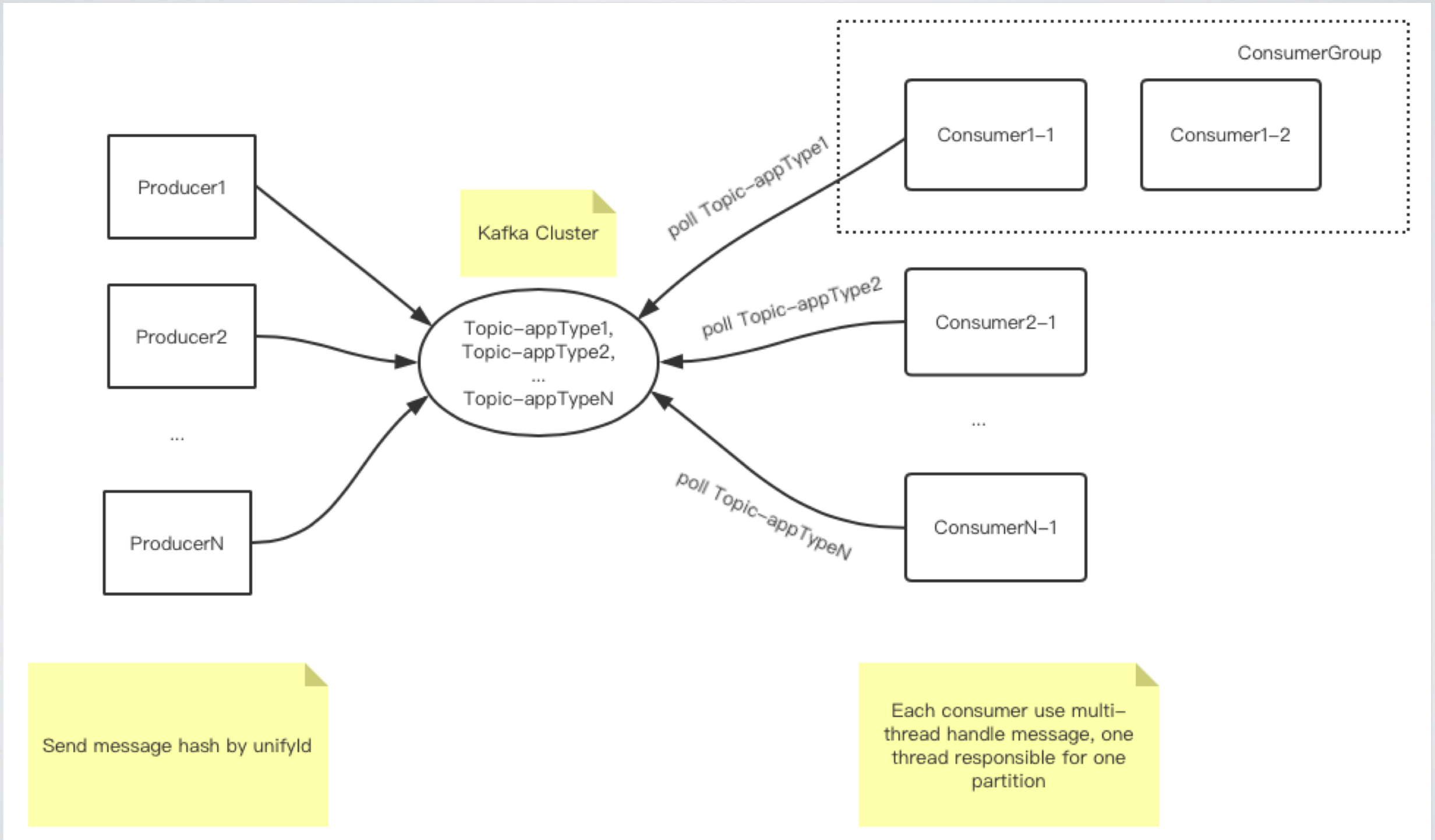
Kafka - Consumer

- Kafka 采用 poll 方式，这种方式的优势是让 broker 变成无状态的，不需要记录消息消费状态。

- 针对某个 topic 的多分区设计，单个分区只能被单个消费者消费，避免了锁，由消费者记录分区消费情况，每个分区记录一个整数值，标记当前已消费的偏移量，减少记录状态的数据量

- 通过修改分区偏移量重新消费旧消息

埋点 Pub/Sub 设计


ConsumerGroup
Producer1
Producer2
...
ProducerN
Kafka Cluster
Topic–appType1,
Topic–appType2,
...
Topic–appTypeN
poll Topic–appType1
poll Topic–appType2
poll Topic–appTypeN
Consumer1–1
Consumer1–2
Consumer2–1
...
ConsumerN–1
Send message hash by unifyId
Each consumer use multi–thread handle message, one thread responsible for one partition

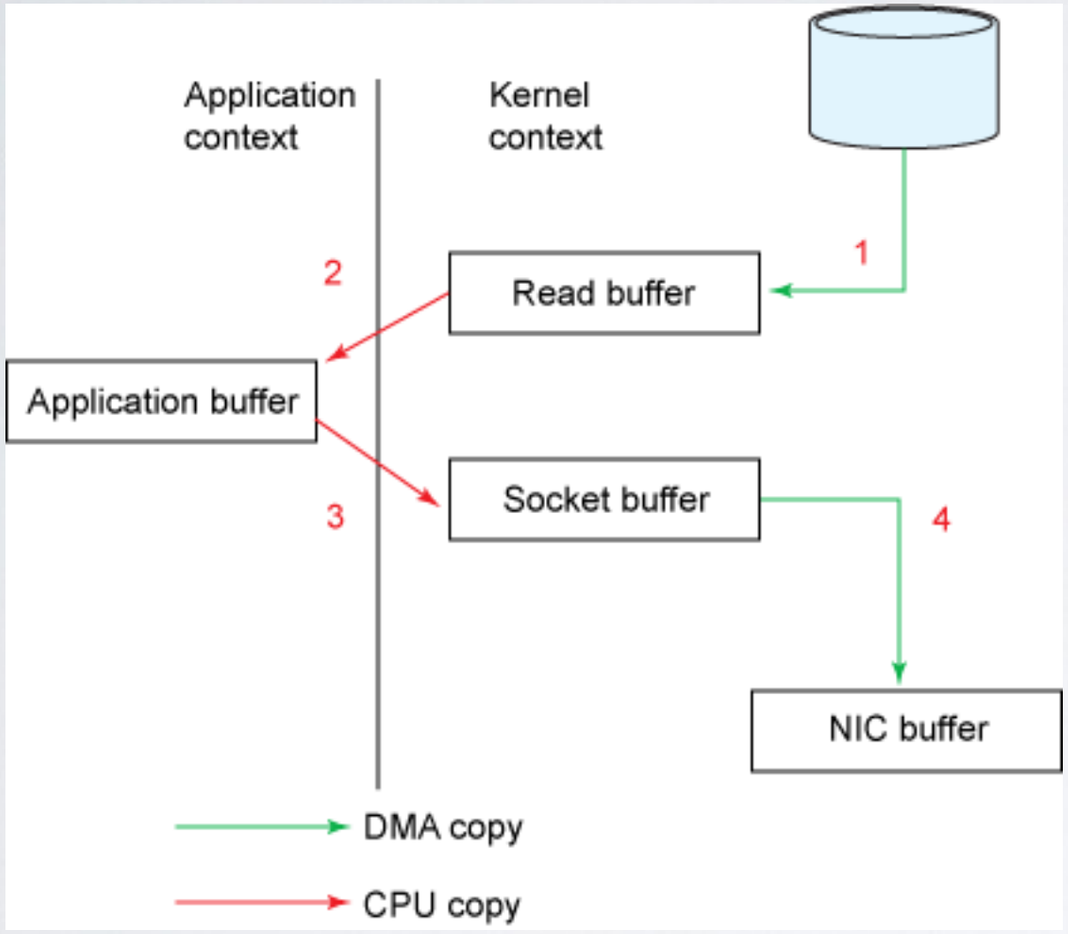
埋点 Pub/Sub 设计说明

- 按照埋点应用类型定义 topic，创建多个 partition，相同 topic 每个商户占用一个 partition
- 多个消费者负责消费某个 topic，消费者使用多线程处理，每个线程负责一个 partition

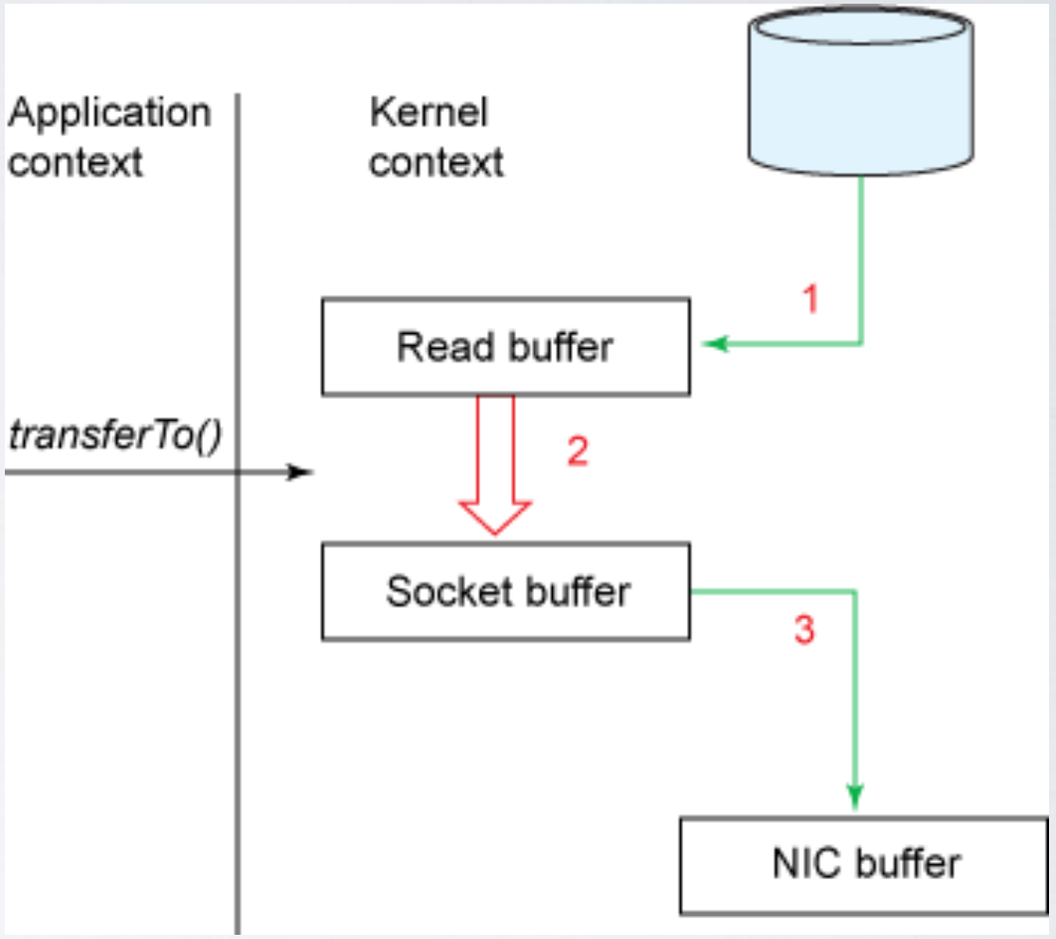
埋点 Pub/Sub 端服务启动

```
[deploy@testhadoop data-collector]$ ls -l data-collector-scripts
total 44
-rwxrwxr-x 1 deploy deploy  584 Jun 29 10:10 compile-collector.sh
-rwxrwxr-x 1 deploy deploy  703 May 22 23:11 genLog.sh
-rwxrwxr-x 1 deploy deploy  335 May 22 23:11 simpleGenLog.sh
-rwxrwxr-x 1 deploy deploy  606 Jun 28 19:00 start-collector-service.sh
-rwxrwxr-x 1 deploy deploy  763 May 24 10:48 start-flume.sh
-rwxrwxr-x 1 deploy deploy   91 May 22 23:11 start-kafka-manager.sh
-rwxrwxr-x 1 deploy deploy   93 May 22 23:11 start-kafka.sh
-rwxrwxr-x 1 deploy deploy  964 Jun 12 15:05 stop-collector-service.sh
-rwxrwxr-x 1 deploy deploy  335 May 22 23:11 stop-flume.sh
-rwxrwxr-x 1 deploy deploy  219 May 22 23:11 stop-kafka-manager.sh
-rwxrwxr-x 1 deploy deploy 1710 May 24 13:58 toolkit.sh
[deploy@testhadoop data-collector]$
[deploy@testhadoop data-collector]$
[deploy@testhadoop data-collector]$ ./data-collector-scripts/start-collector-service.sh
Usage: ./data-collector-scripts/start-collector-service.sh <httpPort> <env> <serviceType: consumer|producer> <topic> [-daemon]
[deploy@testhadoop data-collector]$
[deploy@testhadoop data-collector]$
[deploy@testhadoop data-collector]$ ./data-collector-scripts/start-collector-service.sh 8891 test consumer topicBuryPoint-1 -daemon
```

Kafka - Zero-copy



Traditional



Zero-copy

Kafka - Zero-copy

- 减少用户态和内核态上下文切换次数
- 减少重复的复制次数，减少 CPU 时间和 RAM 带宽，提升数据传输效率

Kafka - 追加写

- 在 Kafka 中，采用消息追加的方式来写入每个消息，每个消息读写时都会利用 Page Cache 的**预读**和**后写**特性，同时 partition 中都使用顺序读写，以此来提高 I/O 性能。

- 虽然 Kafk a能够根据偏移量查找到具体的某个消息，但是查找过程是**顺序查找**，因此如果数据很大的话，查找效率就很低。所以 Kafka 中采用了**分段**和**索引**的方式来解决查找效率问题。Kafka 把一个 patition 大文件又分成了多个小文件段，每个小文件段以偏移量命名，通过多个小文件段，不仅可以使用**二分搜索法**很快定位消息，同时也容易定期清除或删除已经消费完的文件，减少磁盘占用。

- 为了进一步提高查找效率，Kafka 为每个分段后的数据建立了索引文件，并通过索引文件**稀疏存储**来降低元数据占用大小。

埋点 Dashboard - 批处理

大转盘活动流量分析(测试版)

Refresh

埋点数据指标(1)

QUOTA_NAME	QUOTA_VALUE
今日新增用户数	49644
今日抽奖用户数	38512
总用户数	74959
总抽奖次数	188741

< 1 >

Updated: a month ago

埋点数据指标(2)

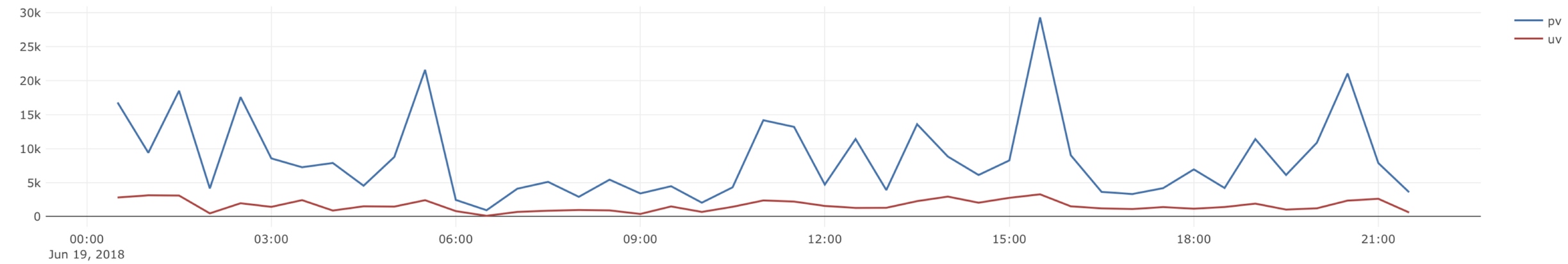
QUOTA_NAME	QUOTA_VALUE
日均pv	502760.02
日均uv	84076.02
日均抽奖数	167542.02
日均抽奖用户数	55843.02

< 1 >

Updated: a month ago

流量趋势分析(小时)

流量趋势分析(小时)



Updated: a month ago

休息一下

实时计算应用场景

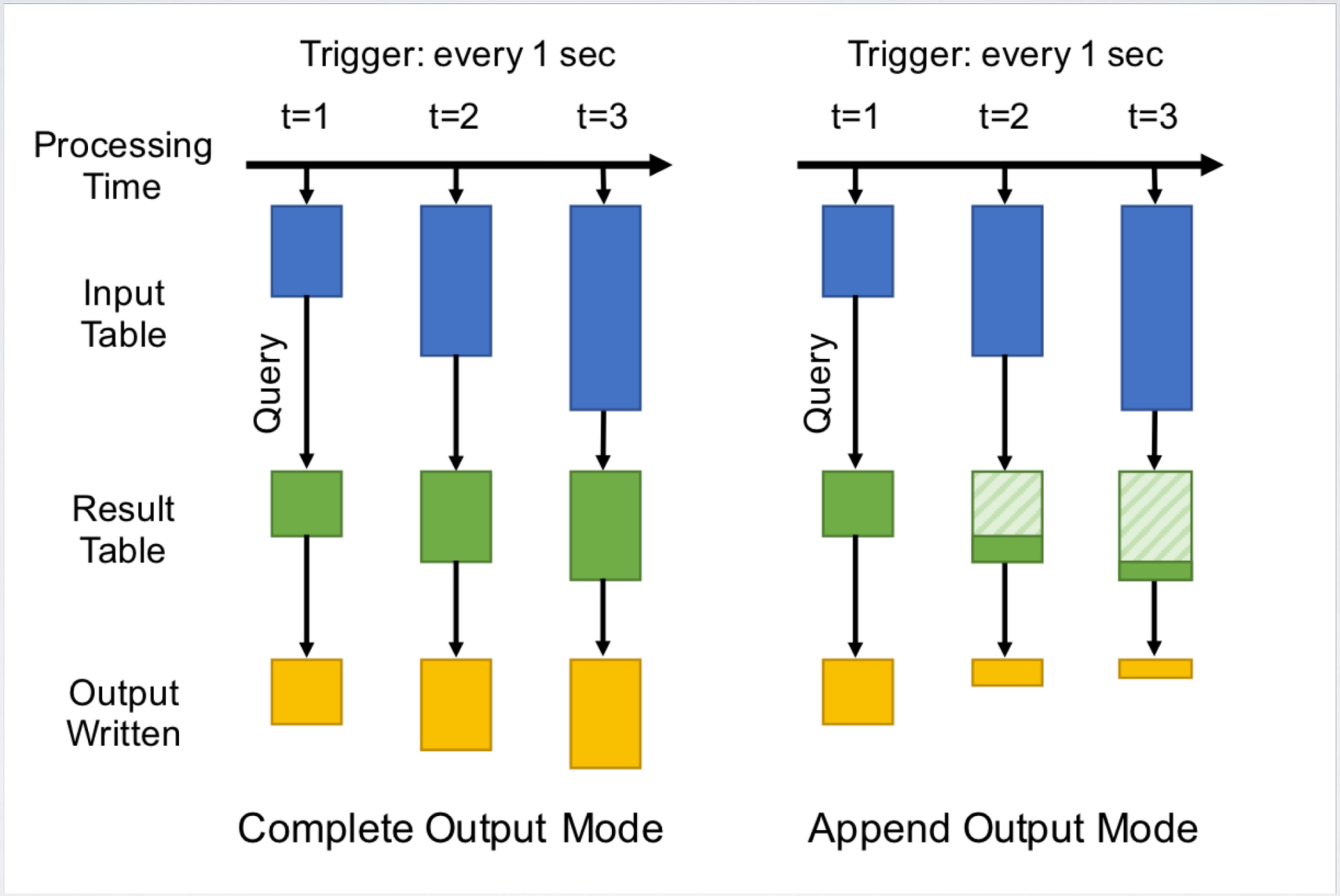
- 实时监控预警
- 实时数据报表
- 实时决策系统 (风控系统)
- 在线机器学习 (个性化实时推荐)
- …

埋点计算 - Spark Streaming

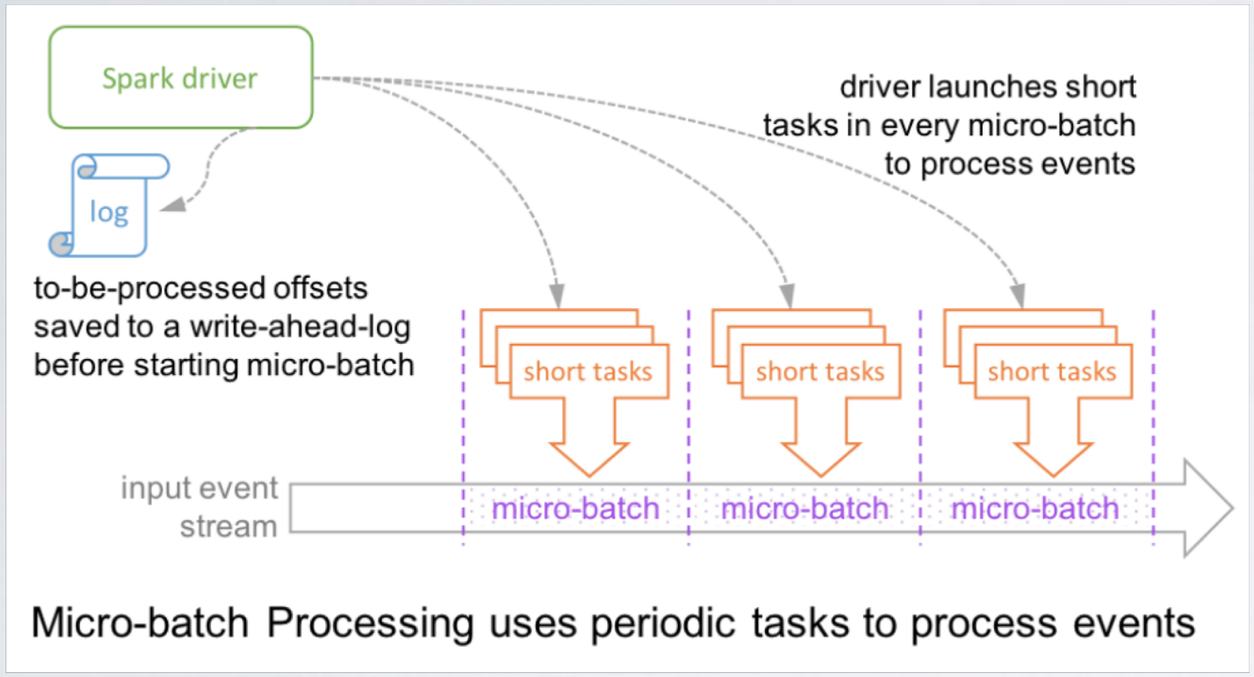
- DStream (Discretized Stream): Spark 早期推出的基于底层 RDD 的面向 micro-batch 计算方式，API 不支持 event-time，引擎无法执行上层优化，社区不推荐使用

- Structured Streaming: Spark 2.2 推出的基于 DataSet/DataFrame 的面向 micro-batch (伪实时) 计算方式，原生支持 event-time，提供上层结构化 API，引擎可自动优化，2.3 版本推出 continuous-process (真实时) 计算方式

Streaming - 增量计算模型

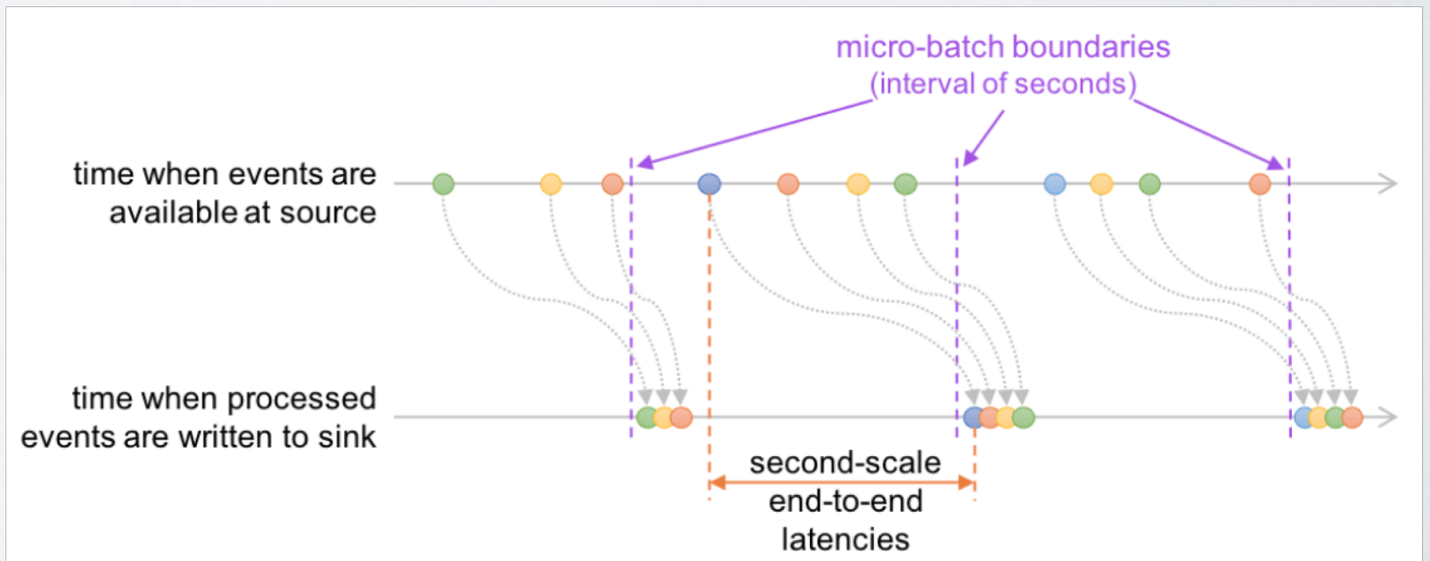


Streaming - Micro-Batch



Micro-batch Processing uses periodic tasks to process events

Structured Streaming 使用 micro-batch 方式可以充分利用已有的 Spark SQL 批处理引擎，可以做到秒级的处理延时 (官网号称 100ms)。

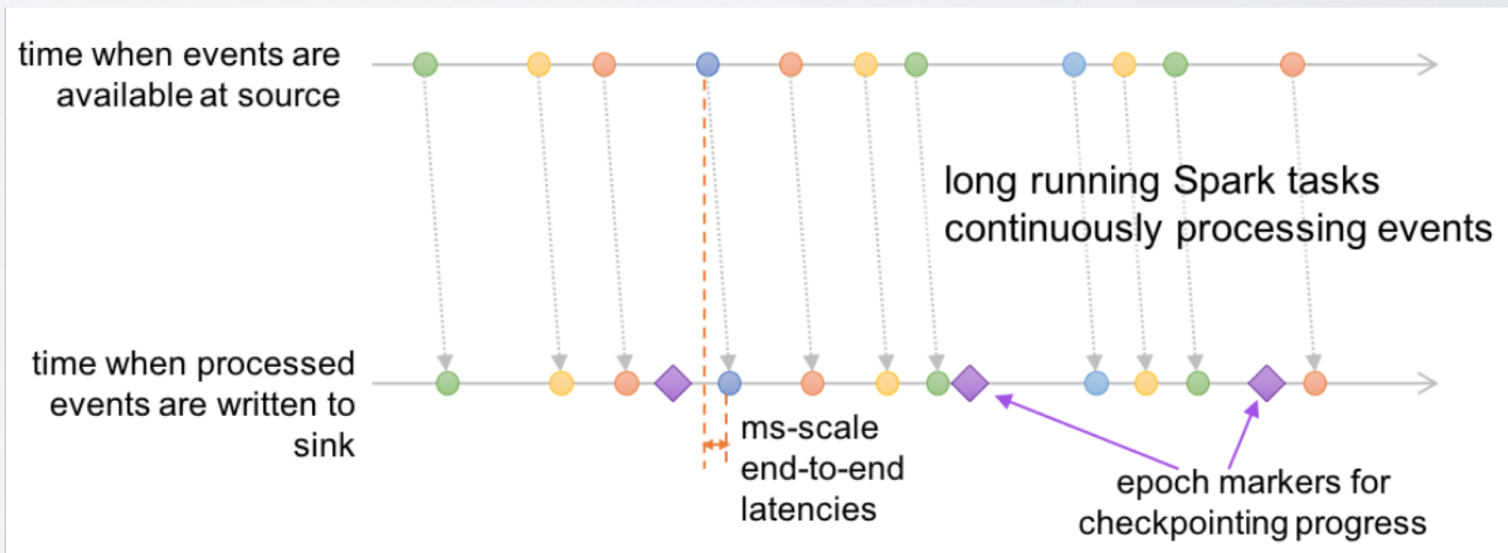


Second-scale end-to-end latencies with Micro-batch Processing

Streaming - Continuous Processing

Continuous Processing uses long-running tasks to continuously process events

Record-at-one-time, 可以实现 ms 延时，目前处于实验阶段



Millisecond-scale end-to-end latencies with Continuous Processing

Streaming - Source

```
val spark = SparkSession.builder.appName( name = "Burypoint streaming demo").getOrCreate()
import spark.implicits._

val buryPointRecords = spark.readStream
  .format( source = "kafka")
  .option("subscribe", "streaming-input")
  .option("kafka.bootstrap.servers", [redacted])
  .load

val buryPointSchema = (new StructType).add( name = "page", StringType)
  .add( name = "openid", StringType).add( name = "ctime", TimestampType)

val buryPointDF = buryPointRecords
  .selectExpr( exprs = "CAST(value AS STRING)")
  .select(from_json($"value", buryPointSchema).as( alias = "buryPointData"))
  .select( col = "buryPointData.*")
```

```scala
val pvDF = buryPointDF
  .withWatermark( eventTime = "ctime",  delayThreshold = "2 hours")
  .groupBy(window( timeColumn = $"ctime",  windowDuration = "1 minutes"), $"page")
  .count()
  .withColumnRenamed( existingName = "count",  newName = "pv")

val uvDF = buryPointDF
  .withWatermark( eventTime = "ctime",  delayThreshold = "2 hours")
  .groupBy(window( timeColumn = $"ctime",  windowDuration = "1 minutes"), $"page").
  agg(approx_count_distinct($"openid").as( alias = "uv"))
```

- Aggregation API
- Window Operation
- Watermark

Streaming - Sink

```
val pvQuery = pvDF.toJSON.as( alias = "value")
  .writeStream
  .queryName( queryName = "query_pv")
  .format( source = "kafka")
  .option("kafka.bootstrap.servers", '[redacted]')
  .option("topic", "streaming-output")
  .option("checkpointLocation", "/data/spark/test/streaming-demo/chkPointPv")
  .outputMode( outputMode = "update")
  .start()

val uvQuery = uvDF.toJSON.as( alias = "value")
  .writeStream
  .queryName( queryName = "query_uv")
  .format( source = "kafka")
  .option("kafka.bootstrap.servers", [redacted]
  .option("topic", "streaming-output")
  .option("checkpointLocation", "/data/spark/test/streaming-demo/chkPointUv")
  .outputMode( outputMode = "update")
  .start()

pvQuery.awaitTermination()
uvQuery.awaitTermination()
```

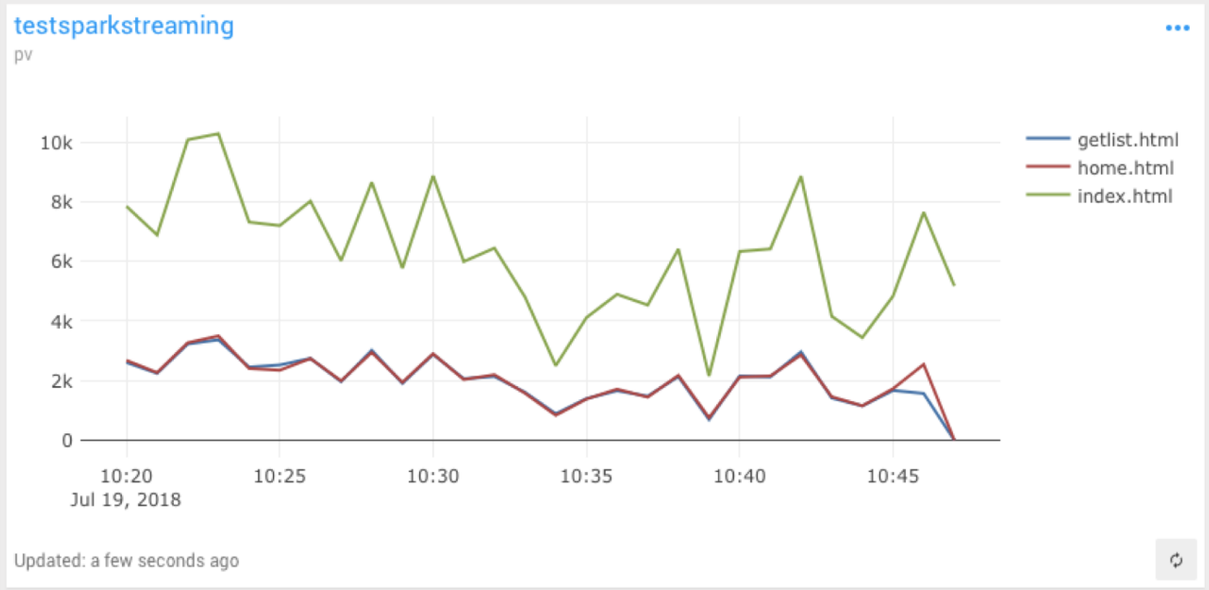
Streaming - Window

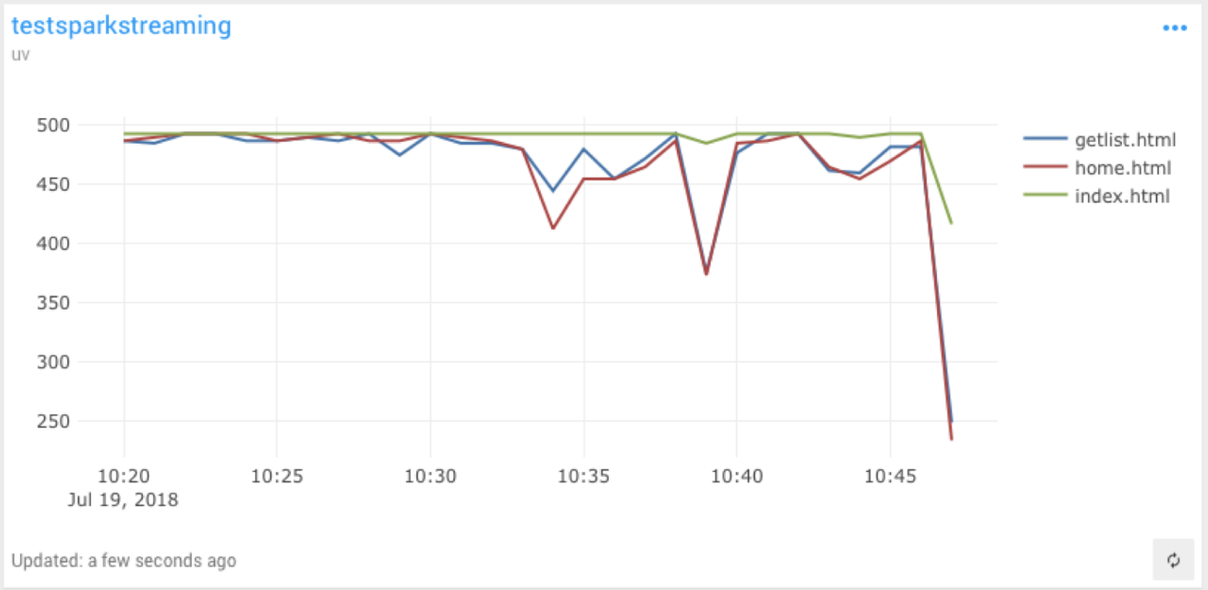
- Tumbling windows
- Sliding windows

Streaming - Join

- Streaming Join Static: 个性化实时推荐时，用户埋点数据需要关联历史数据 (订单数据等)
- Streaming Join Streaming

埋点 Dashboard - Streaming

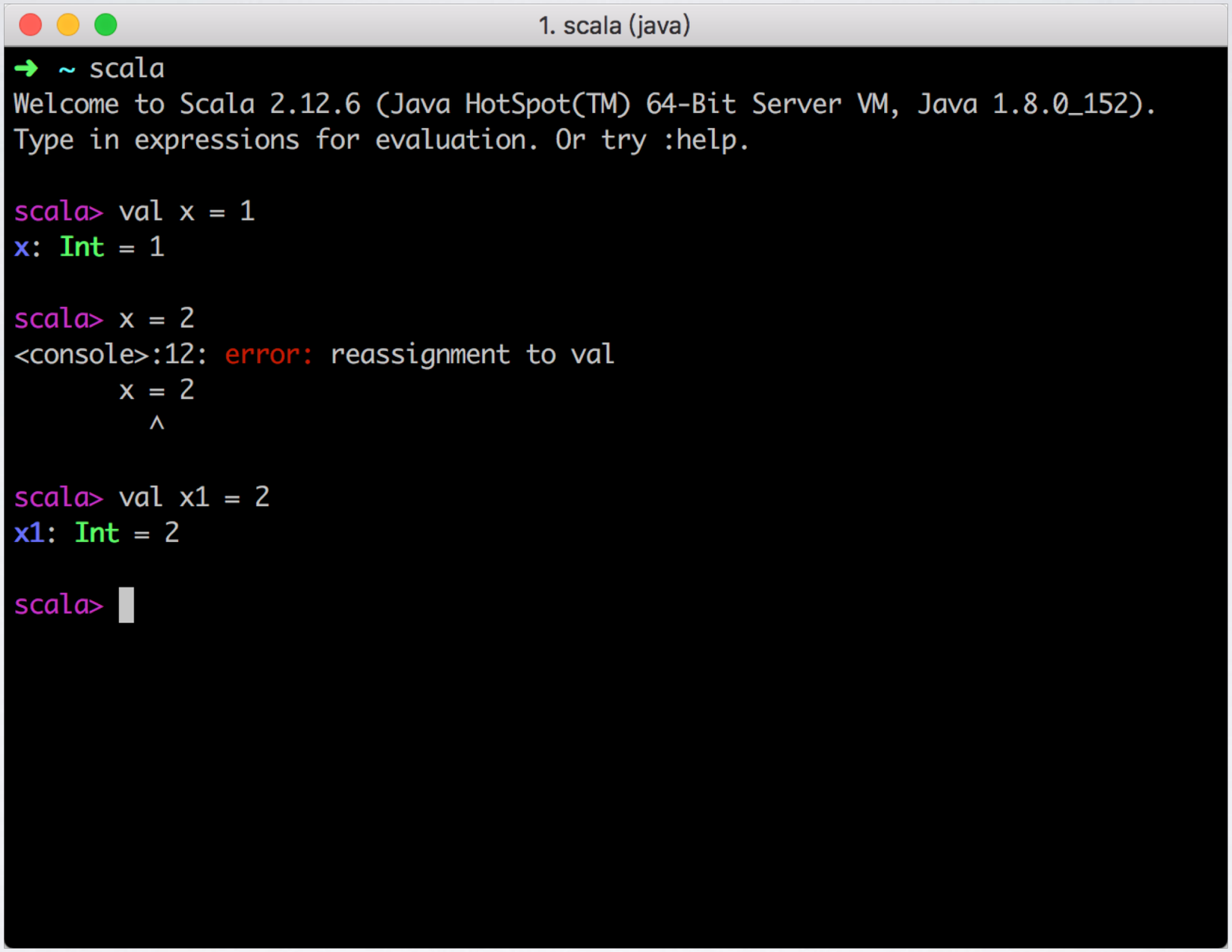




Scala

- 基于 JVM 的编程语言，直接利用 Java 社区资源
- 结合面向对象和函数式编程范式
- 编译型强类型语言，适合构建大型项目

Scala - FP - Immutable



Scala - FP - Pattern Match

```scala
abstract class Device

case class Phone(model: String) extends Device {
  def screenOff = "Turning screen off"
}
case class Computer(model: String) extends Device {
  def screenSaverOn = "Turning screen saver on..."
}

def goIdle(device: Device) = device match {
  case p: Phone => p.screenOff
  case c: Computer => c.screenSaverOn
}
```

Any Questions ?

谢谢大家 ~