互联网金融公司大数据实时数据仓库的

实践

胡夕

00 打个广告…

- Kafka中国社区QQ群： 162272557

- Kafka技术分享微信公众号

01 关于我们公司…

一家互联网金融公司

资产配置

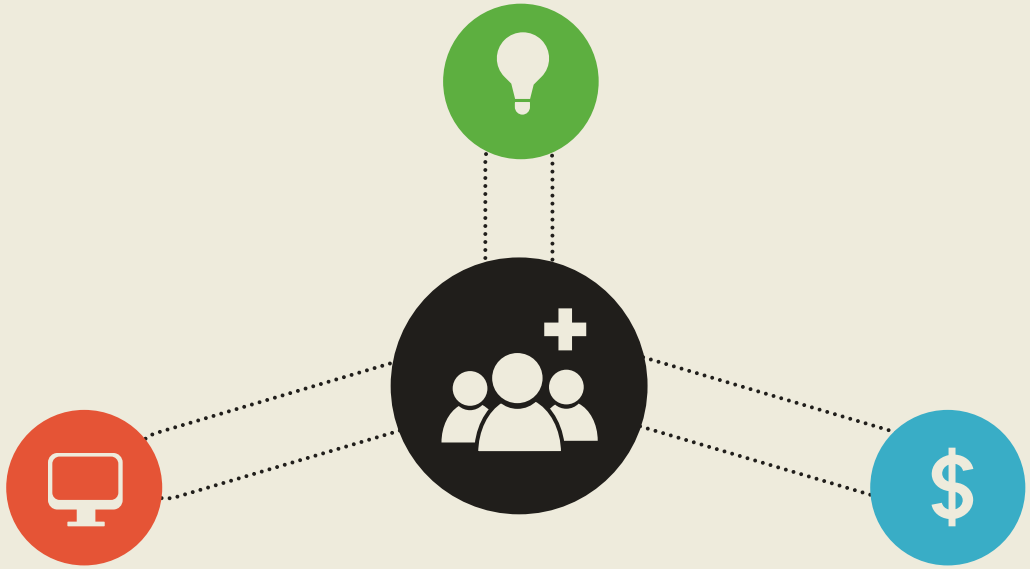
- 资产在不同资产类别之间的分配
- 不同等级的风险与收益的组合
- 面向广大C端用户
- 利用互联网手段提升传统资产配置效率

互联网借贷

- 适应个人和小微企业的融资需求
- 借助互联网技术和机器学习模型管控风险

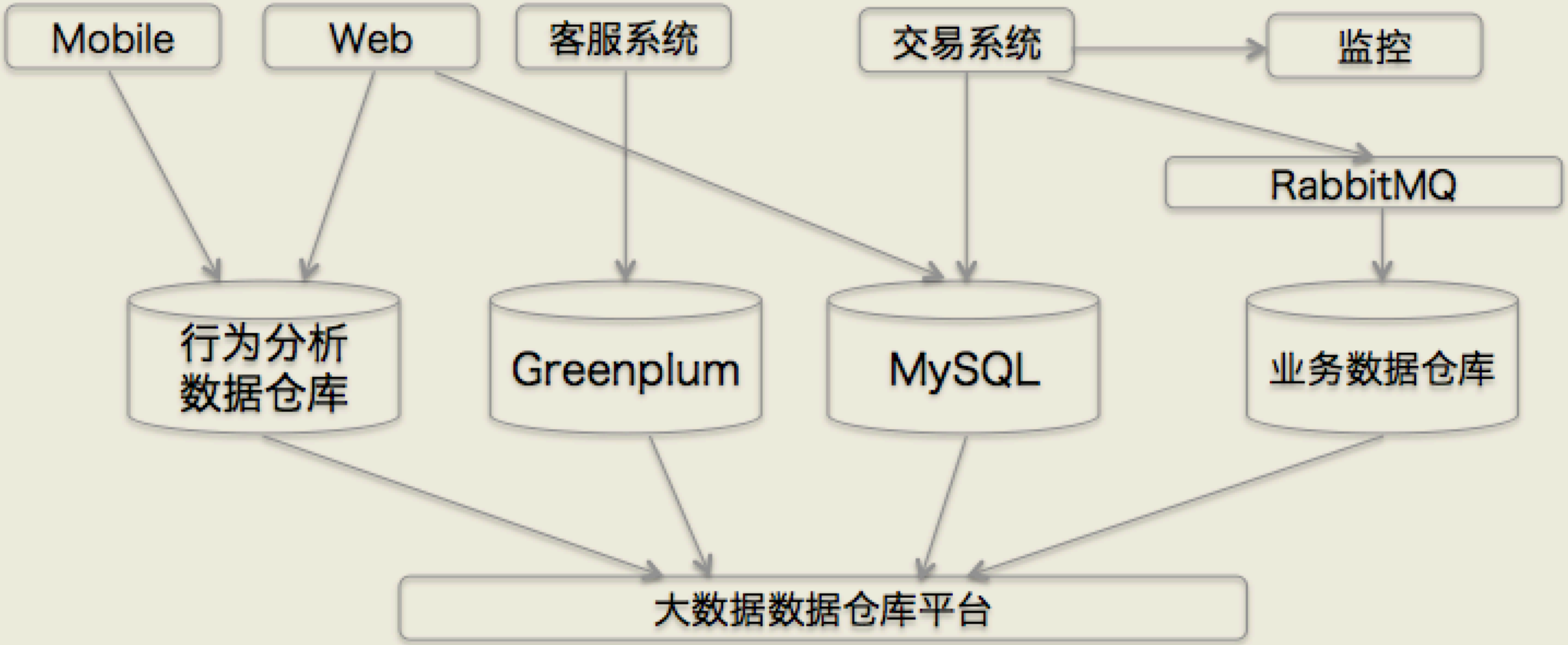
02 当前业务痛点

- 数据仓库数据源多样
  - 子系统众多
  - 数据整合难度大
  - 点对点的系统对接维护成本高
- 数据仓库数据量大
  - TB级别的风险数据存储
- 数据“落地” 周期长，非实时
  - 离线批处理



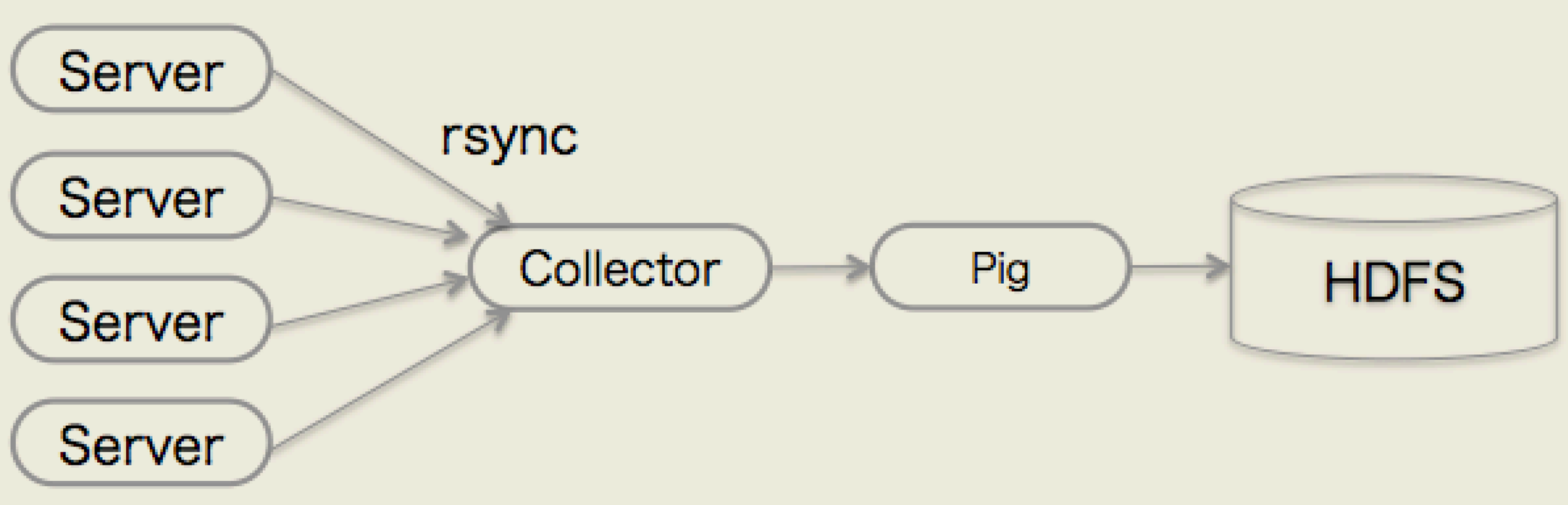
交易数据	风控数据	其他数据
• 购买 • 赎回 • 充值 • 提现	• 交易记录 • SNS记录 • 个人信用评估 • 电商记录	• 通道管理 • 流动性管控 • 计费数据 • 行为数据

03 之前的架构


Mobile
Web
客服系统
交易系统
监控
RabbitMQ
行为分析
数据仓库
Greenplum
MySQL
业务数据仓库
大数据数据仓库平台

04 之前的数据处理流程

- rsync收集
- apache pig + SHELL
- 基于MapReduce作业
- 非实时的离线批处理



05 以Kafka为中心重构业务系统

交易系统

捕获关键业务数据(购买、提现、充值和赎回)

监控系统

实时监控业务指标并及时告警

用研系统

用户调研数据收集与分析

客服系统

保留全量客服数据，提升服务水平

风控系统

为下游风控模型提供数据

征信系统

不断完善用户画像，更新信用评估模型

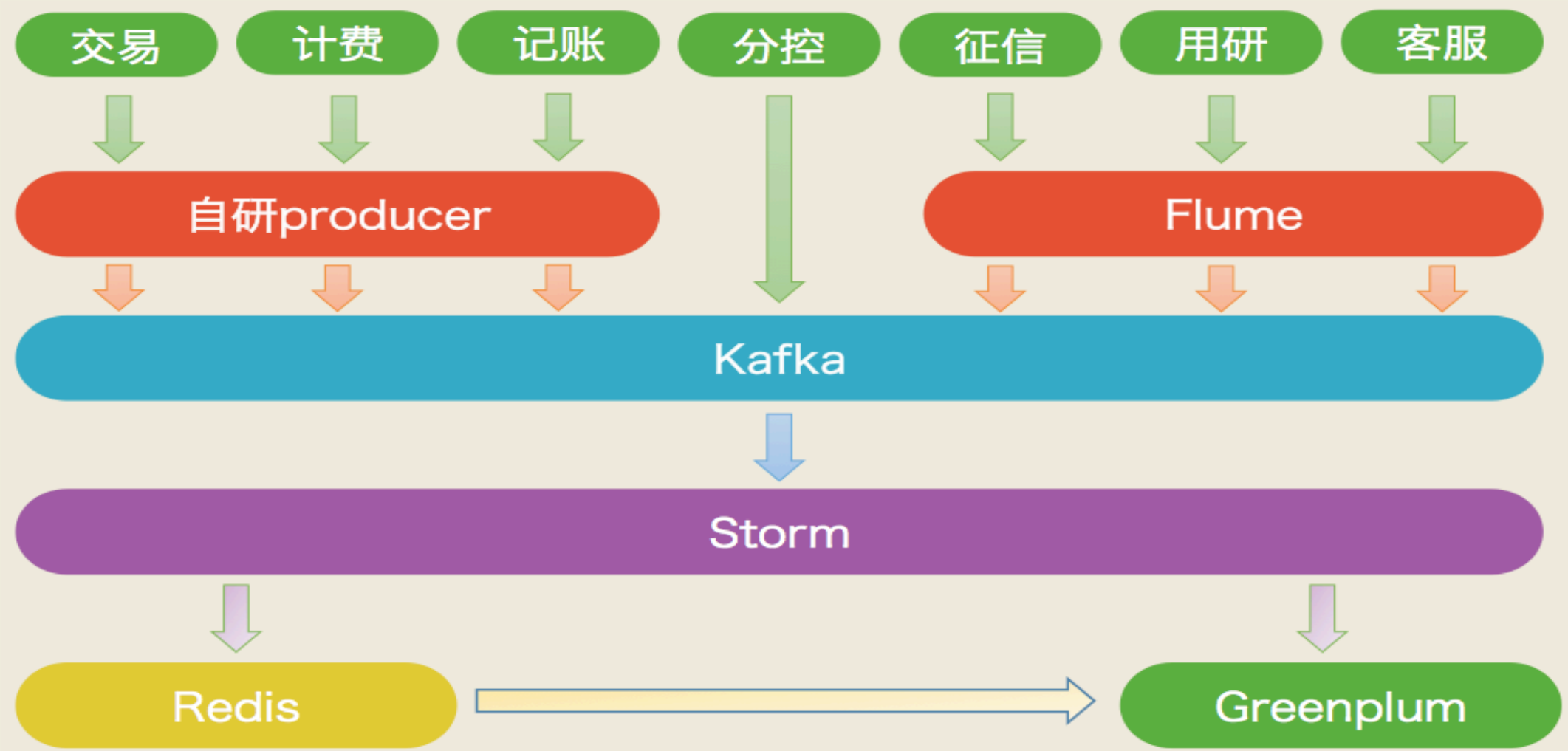
计费系统

准确及时地提供计费记录

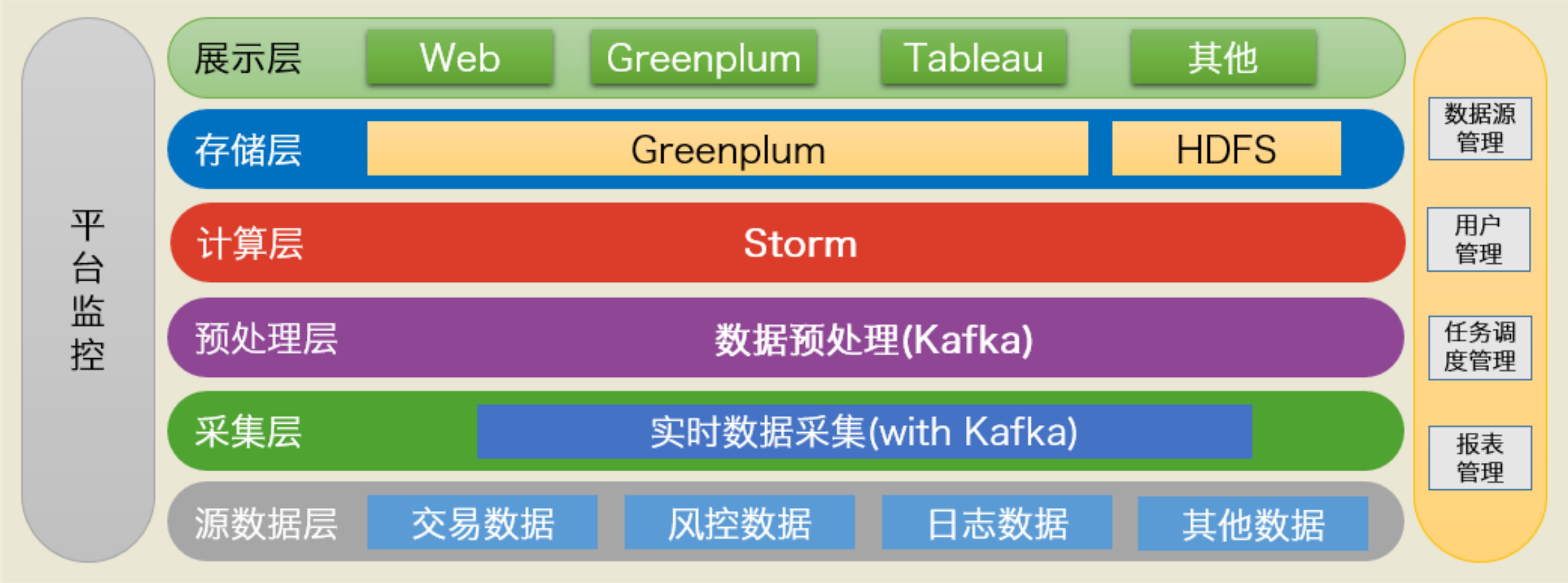
对账系统

降低人工开销，提升准确率

06 业务架构


交易
计费
记账
分控
征信
用研
客服
自研producer
Flume
Kafka
Storm
Redis
Greenplum

07 系统架构


平台监控
展示层
Web
Greenplum
Tableau
其他
存储层
Greenplum
HDFS
计算层
Storm
预处理层
数据预处理(Kafka)
采集层
实时数据采集(with Kafka)
源数据层
交易数据
风控数据
日志数据
其他数据
数据源管理
用户管理
任务调度管理
报表管理

08 Kafka部署情况

业务分级别部署

核心业务分区独立部署
非核心业务分区混合部署

2亿条消息/天

每日平均产生消息数2亿条

600 Bytes

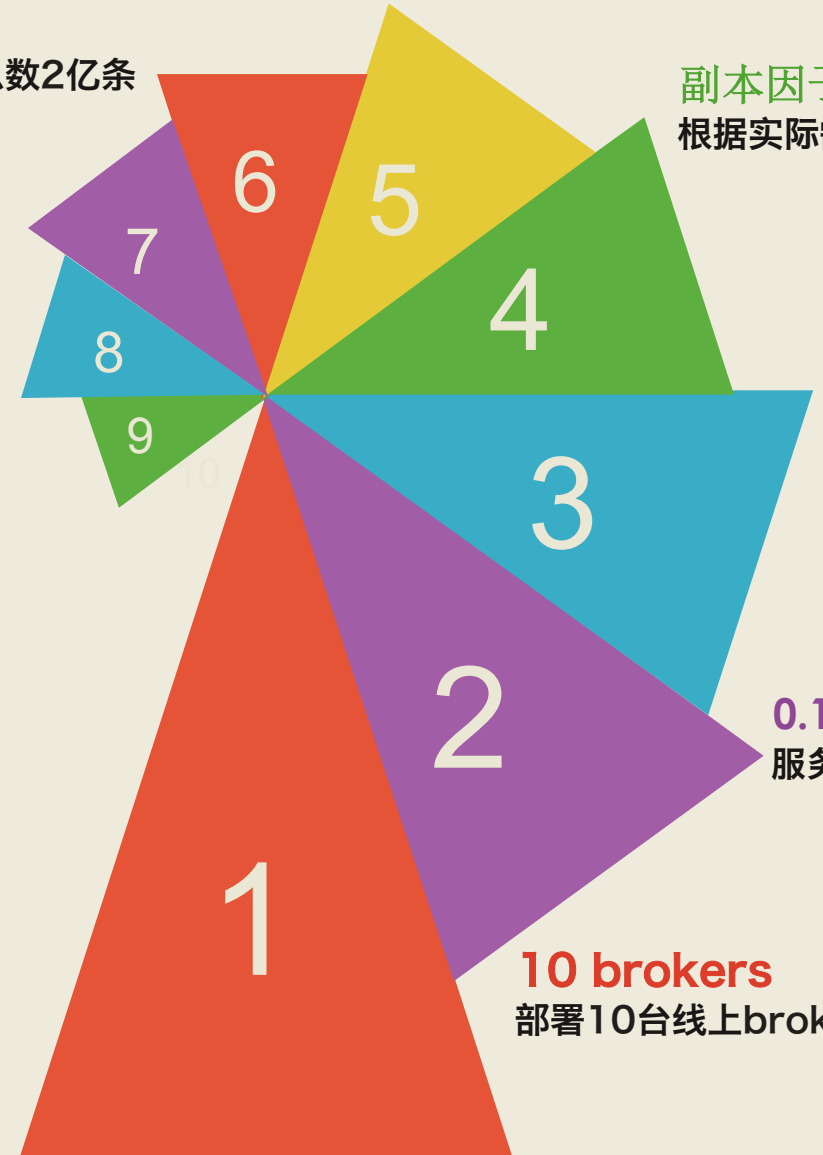
平均消息大小600字节

20Mbps

入站带宽

460Mbps

出站带宽



副本因子=2

根据实际需求设置合理备份数

8 topic，20+分区

根据实际业务划分topic

0.10.0.1

服务器版本升级到0.10.0.1版本

10 brokers

部署10台线上broker机器

09 Kafka部署规划

生产环境部署

业务划分

为每个业务方严格设定唯一的client.id，方便监控与DEBUG

业务等级

核心业务采用独立主备方式，避免其他业务干扰
非核心业务采用非独立主备混合部署方式，节省硬件资源

ZK集群分离

摒弃单机同时安装Zookeeper和Kafka的部署方式，单独部署ZK集群以最大化ZK写入性能

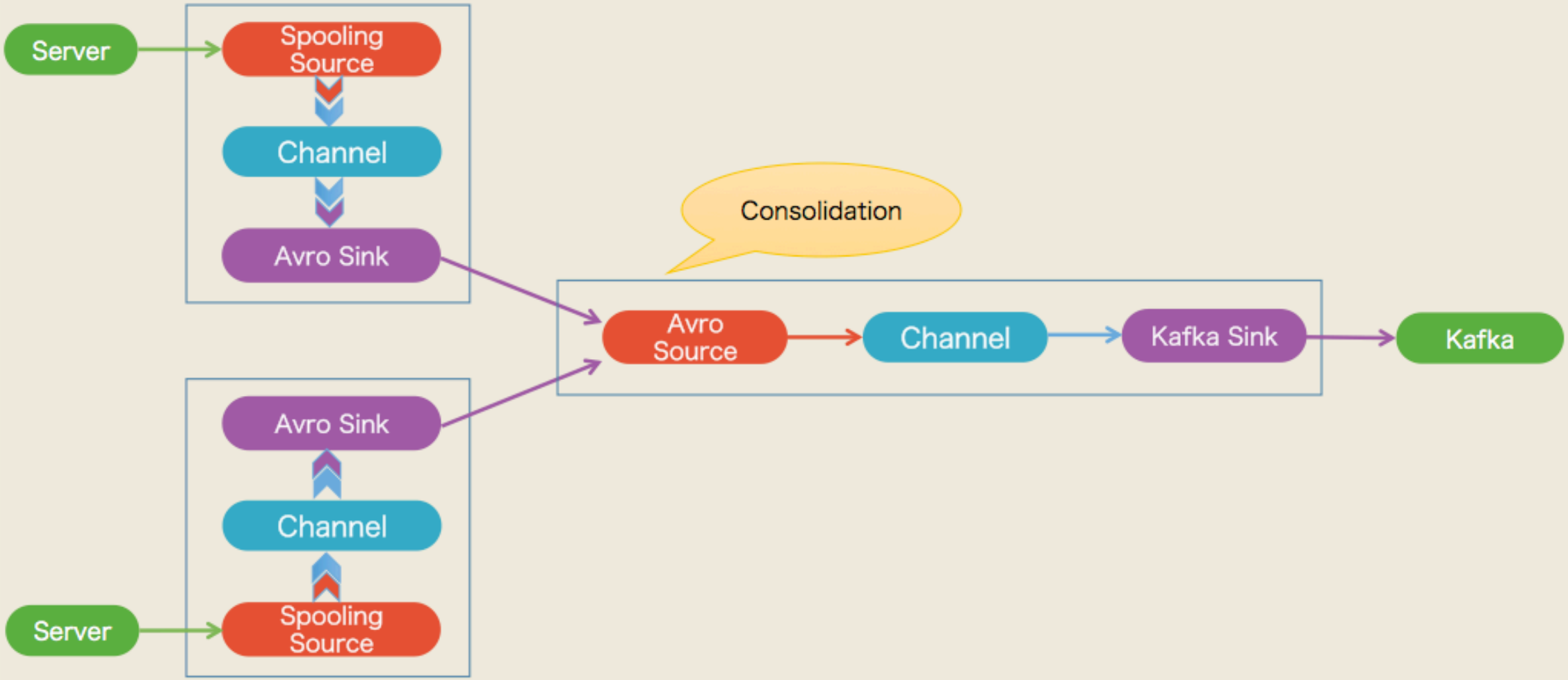
容量评估

制定满足性能目标的容量评估方案
预估各种硬件资源的1年内使用上限

资源规划

OS
CPU
RAM
磁盘
带宽
JVM
GC

10 Kafka + Flume

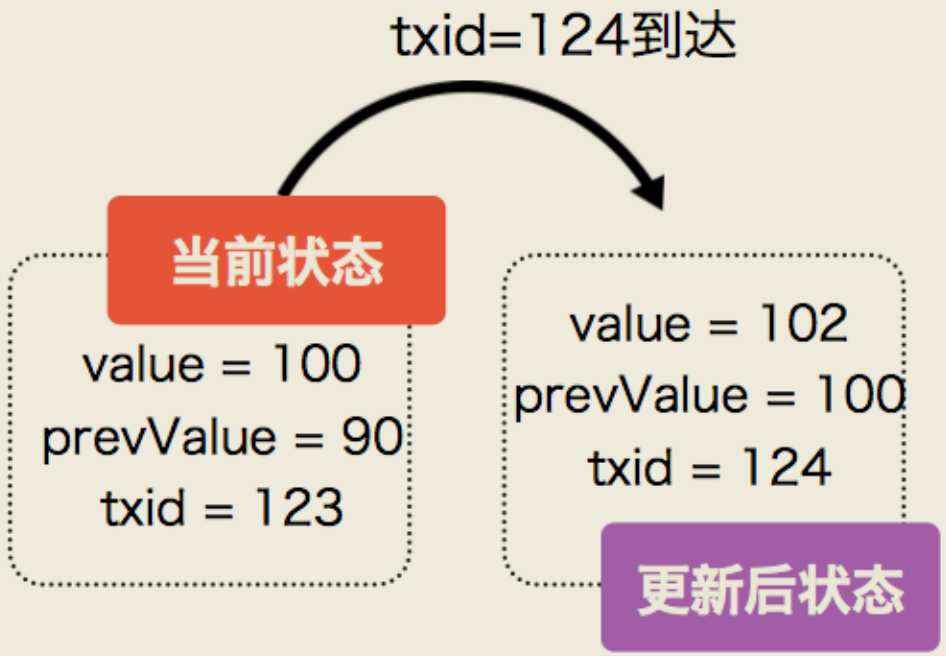


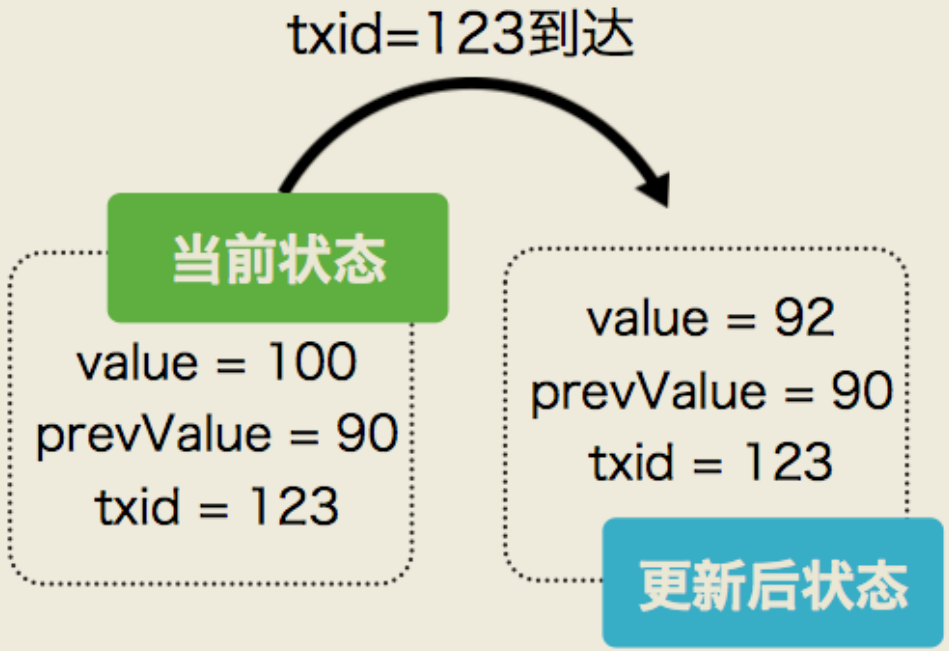
11 自研producer

```
Properties props = ProducerPropertyFactory.init("biz-client-id", brokerList);
QueuedKafkaProducer producer = new QueuedKafkaProducer<>(props);
//...
String eventBody = ...;
ByteBuffer valueBuffer = ByteBuffer.allocate(bizName.length() + eventId.length() + eventBody.length());
valueBuffer.put(bizName.getBytes("UTF-8"));
valueBuffer.put(eventId.getBytes("UTF-8"));
valueBuffer.put(eventBody.getBytes("UTF-8"));

ProducerRecord<String, String> record = new ProducerRecord<>(topic, eventKey, valueBuffer.array());

producer.send(new ProducerRecord<>(topic, eventKey, eventValue), new Callback() {
    @Override
    public void onCompletion(RecordMetadata metadata, Exception exception) {
        if (exception != null) {
            logger.warn("Sent event failed due to {}", exception);
            errorQueue.offer(record); // 由专门的错误队列handler单独处理
        } else {
            eventQueue.remove(record);
            producer.updateCheckpointFile(eventId);
            producer.updateSuccessfulSentEventCount();
        }
    }
});
```

12 Kafka + Storm Trident

- 实现精确一次(exactly-once processing)
  - 每个元组(tuple)只在一个batch中被处理一次
  - OpaqueTridentKafkaSpout容忍Kafka broker的fail
  - 使用额外空间保存状态实现精确一次的处理语义
  - 老版本Low-level consumer + ZK-based offset storing





13 Kafka + Storm Trident

- 示例代码

```
/**
 * Create an opaque trident kafka spout
 *
 * @param topic kafka topic
 * @return kafka spout
 */
protected static OpaqueTridentKafkaSpout buildKafkaSpout(String topic) {
    BrokerHosts zk = new ZkHosts(configReader.get( key: topic + ".kafka.zk.connect",  defaultValue: "localhost:2181"),
            configReader.get( key: topic + ".kafka.zk.path",  defaultValue: "/brokers"));
    TridentKafkaConfig config = new TridentKafkaConfig(zk, configReader.get( key: topic + ".kafka.topic", topic));
    boolean readFromStart = configReader.get( key: topic + ".storm.topology.spout.forceFromStart",  defaultValue: false);
    config.startOffsetTime = readFromStart ? OffsetRequest.EarliestTime() : OffsetRequest.LatestTime();
    config.scheme = new SchemeAsMultiScheme(new StringScheme());
    return new OpaqueTridentKafkaSpout(config);
}
```

```
Set<InetSocketAddress> nodes = initRedisService(redisConnectionStr);
JedisClusterConfig clusterConfig = new JedisClusterConfig.Builder().setNodes(nodes).build();
RedisClusterState.Factory factory = new RedisClusterState.Factory(clusterConfig);
stream.name("partitioned-redis-stream").partitionPersist(
        factory,
        new Fields("processed-event"),
        new RedisClusterStateUpdater(storeMapper).withExpire(redisEntryTTL),
        new Fields()
);
```

14 Kafka环境主要参数配置

Broker

log.retention.hours=72
num.network.threads=8
auto.leader.rebalance.enable=false
log.segment.bytes=2GB
replica.fetch.max.bytes=2MB
connections.max.idle.ms=-1
unclean.leader.election.enable=false

Producer

batch.size=300KB
linger.ms=50ms
compression.type=lz4
retries=Integer.MAX
设置client.id
max.in.flight.requests.per.connection=1
max.request.size=2MB

Consumer

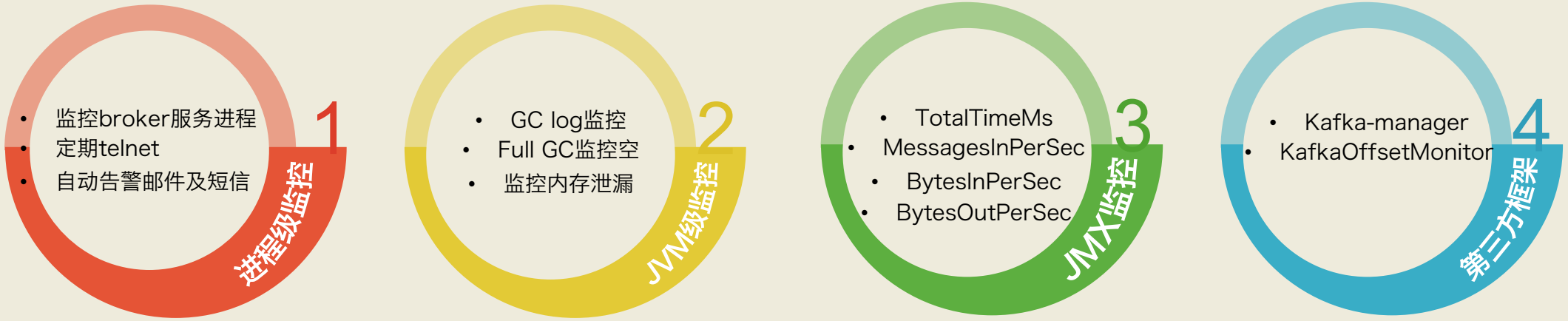
fetch.message.max.bytes=2MB

auto.commit.enable=false

OS & JVM

使用ext3/ext4
commit interval = 60s
sysctl vm.swappiness = 0
4GB JVM heap
使用CMS collector

15 Kafka集群监控



1. 缺乏统一的解决方案，需要自行组合不同级别指标构建监控方案
2. 不同信息散落在不同的监控组件中
   - 各种脚本，如kafka-topics/kafka-consumer-groups等
   - 服务器端helper类，如AdminClient/AdminUtils等
   - 客户端协议，如CreateTopic/DeleteTopics/DescribeGroups等
   - JMX指标(https://kafka.apache.org/documentation/#monitoring)
   - Zookeeper节点，如/brokers/topics，/brokers/ids等
3. 第三方监控框架更新不及时

16 实时数据仓库收集与分析

1 分钟级实时收集

消息全链路处理时间控制在5分钟内

2 精确一次消息处理

结合Storm Trident实现消息“精确一次”的处理语义

3 半小时级数据分析

借助Greenplum集群和定义良好的表分区实现半小时级的ad-hoc查询

4 实时监控告警

支持邮件/短信的实时业务级和IT级告警

17 踩过的“坑”(1)

- Zookeeper快照文件“打爆”磁盘
  - autopurge.purgeInterval = 24
  - autopurge.snapRetainCount = 5
- Zookeeper单台节点Socket最大连接数
  - maxClientCnxns = 200
- 副本在ISR中频繁进出
  - 发现是follower无法稳定追上leader
  - num.replica.fetchers = 8
- 大消息处理
  - broker端：调整replica.fetch.max.bytes / message.max.bytes
  - producer端：调整max.request.size
  - consumer端：调整fetch.message.max.bytes

18 踩过的“坑”(2)

- 过大的JVM heap拉长整体GC时间
  - 设置JVM heap不超过4GB

- 关闭自动preferred leader选举
  - 新leader的广播通知会被“积压”的客户端请求推迟
  - 高水位的截断有可能使PRODUCE请求丢失

- Controller偶发报错：Connection to *** was disconnected before the response was read
  - 出现时短暂推高PRODUCE请求处理时间
  - 倾向于认为是kafka-3916

19 数据仓库平台后续展望

替换storm-kafka

使用最新版本的storm-kafka-clients替换旧版本的consumer

完善现有监控方案

基于Kafka-manager进行二次定制开发

升级到Kafka 0.10.2版本

修复了GC阶段性阻塞broker的问题(详见kafka-4614)

使用Java 8

使用G1替代CMS，且社区已决定在0.11正式放弃对Java 7的支持

20 对社区的展望

完善监控方案

当前Kafka无统一监控解决方案，各个信息散落在不同的监控工具中

精确一次处理语义

主流流式处理框架必然要支持exactly-once的处理语义

Consumer Lag智能诊断

监控方案只能做到lag数查询，无法给出进一步诊断信息

带粘性的rebalance

当前是“牵一发而动全身”，效率很低，最好做到最小程度rebalance

谢　谢！