Kafka 技术介绍

一. Kafka 简介

Kafka 是一种分布式的，基于发布/订阅的消息系统。它最初由 LinkedIn 公司开发，之后成为 Apache 项目的一部分。

Apache Kafka 与传统消息系统相比，有以下不同：

- 它被设计为一个分布式系统，易于向外扩展；
- 它同时为发布和订阅提供高吞吐量；
- 它支持多订阅者，当失败时能自动平衡消费者；
- 它将消息持久化到磁盘，因此可用于批量消费，例如 ETL，以及实时应用程序。

二. Kafka 架构

Kafka 的架构包括以下组件：

- **话题**（Topic）是特定类型的**消息**流。**消息**是字节的有效负载（Payload），话题是消息的分类名或种子（Feed）名。
- **生产者**（Producer）是能够发布消息到话题的任何对象。
- 已发布的消息保存在一组服务器中，它们被称为**代理**（Broker）或 Kafka **集群**。
- **消费者**可以订阅一个或多个话题，并从 Broker 拉数据，从而消费这些已发布的消息。

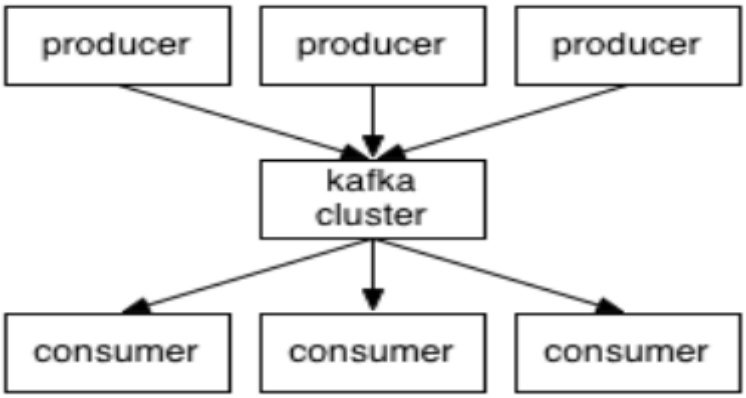

图 1：Kafka 生产者、消费者和代理环境

生产者可以选择自己喜欢的序列化方法对消息内容编码。为了提高效率，生产者可以在一个发布请求中发送一组消息。下面的代码演示了如何创建生产者并发送消息。

生产者示例代码：

```
producer = new Producer(…);
message = new Message(“test message str”.getBytes());
set = new MessageSet(message);
producer.send(“topic1”, set);
```

为了订阅话题，消费者首先为话题创建一个或多个消息流。发布到该话题的消息将被均衡地分发到这些流。每个消息流为不断产生的消息提供了迭代接口。然后消费者迭代流中的每一条消息，处理消息的有效负载。与传统迭代器不同，消息流迭代器永不停止。如果当前没有消息，迭代器将阻塞，直到有新的消息发布到该 话题。Kafka 同时支持点到点分发模型（Point-to-point delivery model），即多个消费者共同消费队列中某个消息的单个副本，以及发布-订阅模型（Publish-subscribe model），即多个消费者接收自己的消息副本。下面的代码演示了消费者如何使用消息。

消费者示例代码：

```
streams[] = Consumer.createMessageStreams(“topic1”, 1)
for (message : streams[0]) {
bytes = message.payload();
// do something with the bytes
}
```

Kafka 的整体架构如图 2 所示。因为 Kafka 内在就是分布式的，一个 Kafka 集群通常包括多个代理。为了均衡负载，将话题分成多个分区，每个代理存储一或多个分区。多个生产者和消费者能够同时生产和获取消息。

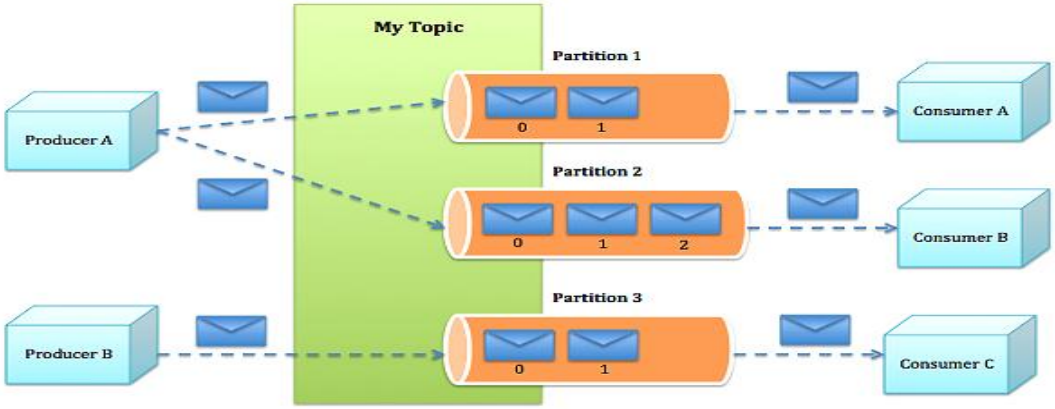

图 2：Kafka 架构

三．Kafka 存储

Kafka 的存储布局非常简单。话题的每个分区对应一个逻辑日志。物理上，一个日志为相同大小的一组分段文件。每次生产者发布消息到一个分区，代理 就将消息追加到最后一个段文件中。当发布的消息数量达到设定值或者经过一定的时间后，段文件真正写入磁盘中。写入完成后，消息公开给消费者。

与传统的消息系统不同，Kafka 系统中存储的消息没有明确的消息 Id。

消息通过日志中的逻辑偏移量来公开。这样就避免了维护配套密集寻址，用于映射消息 ID 到实际消息地址的随机存取索引结构的开销。消息 ID 是增量的，但不连续。要计算下一消息的 ID，可以在其逻辑偏移的基础上加上当前消息的长度。

消费者始终从特定分区顺序地获取消息，如果消费者知道特定消息的偏移量，也就说明消费者已经消费了之前的所有消息。消费者向代理发出异步拉请求，准备字节缓冲区用于消费。每个异步拉请求都包含要消费的消息偏移量。Kafka 利用 sendfile API 高效地从代理的日志段文件中分发字节给消费者。

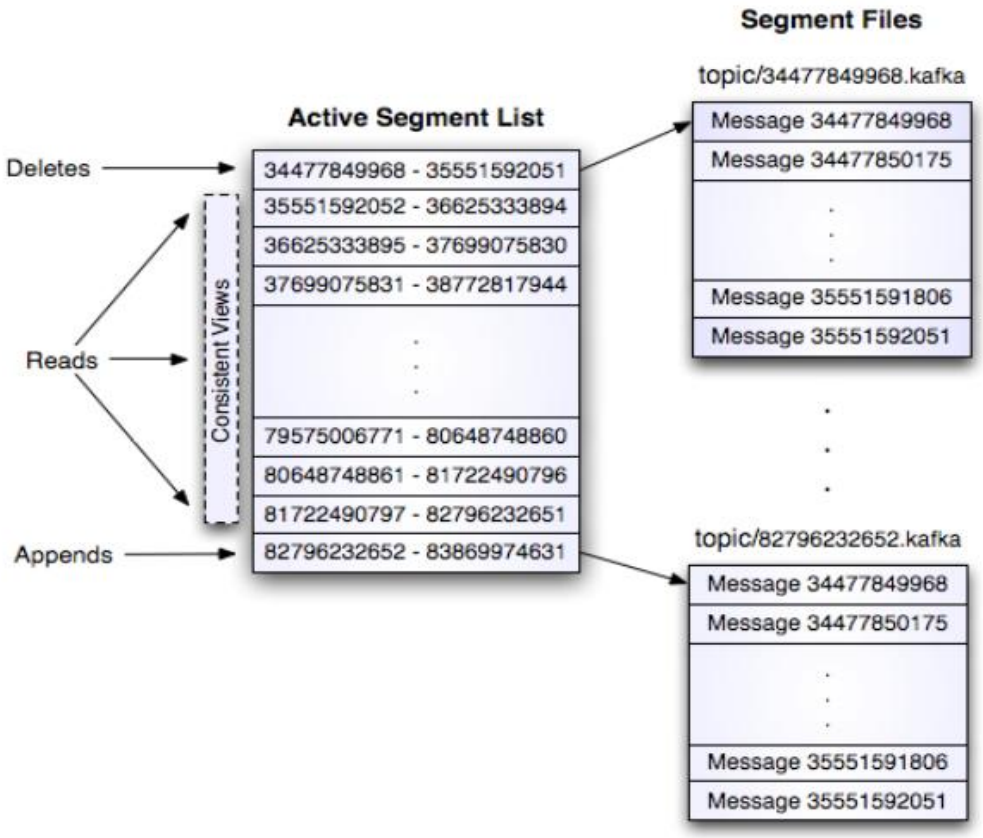

图 3：Kafka 存储架构

四．Kafka 代理

与其它消息系统不同，Kafka 代理是无状态的。这意味着消费者必须维护已消费的状态信息。这些信息由消费者自己维护，代理完全不管。这种设计非常微妙，它本身包含了创新。

- 从代理删除消息变得很棘手，因为代理并不知道消费者是否已经使用了该消息。Kafka 创新性地解决了这个问题，它将一个简单的基于时间的 SLA 应用于保留策略。当消息在代理中超过一定时间后，将会被自动删除。
- 这种创新设计有很大的好处，消费者可以故意倒回到老的偏移量再次消费数据。这违反了队列的常见约定，但被证明是许多消费者的基本特征。

五．ZooKeeper 与 Kafka

考虑一下有多个服务器的分布式系统，每台服务器都负责保存数据，在数据上执行操作。这样的潜在例子包括分布式搜索引擎、分布式构建系统或者已知的系统如 Apache Hadoop。 所有这些分布式系统的一个常见问题是，你如何在任一时间点确定哪些服务器活着并且在工作中。最重要的是，当面对这些分布式计算的难题，例如网络失败、带宽 限制、可变延迟连接、安全问题以及任何网络环境，甚至跨多个数据中心时可能发生的错误时，你如何可靠地做这些事。这些正是 Apache ZooKeeper 所 关注的问题，它是一个快速、高可用、容错、分布式的协调服务。你可以使用 ZooKeeper 构建可靠的、分布式的数据结构，用于群组成员、领导人选举、协 同工作流和配置服务，以及广义的分布式数据结构如锁、队列、屏障（Barrier）和锁存器（Latch）。许多知名且成功的项目依赖于 ZooKeeper，其中包括 HBase、Hadoop 2.0、Solr Cloud、Neo4J、Apache Blur（Incubating）和 Accumulo。

ZooKeeper 是一个分布式的、分层级的文件系统，能促进客户端间的松耦合，并提供最终一致的，类似于传统文件系统中文件和目录的 Znode 视 图。它提供了基本的操作，例如创建、删除和检查 Znode 是否存在。它提供了事件驱动模型，客户端能观察特定 Znode 的变化，例如现有 Znode 增加了 一个新的子节点。ZooKeeper 运行多个 ZooKeeper 服务器，称为 **Ensemble**，以获得高可用性。每个服务器都持有分布式文件系统的内存复本，为客户端的读取请求提供服务。

下图 4 展示了典型的 ZooKeeper ensemble，一台服务器作为 Leader，其它作为 Follower。当 Ensemble 启动时，先选出 Leader，然后所有 Follower 复 制 Leader 的状态。所有写请求都通过 Leader 路由，变更会广播给所有 Follower。变更广播被称为**原子广播**。

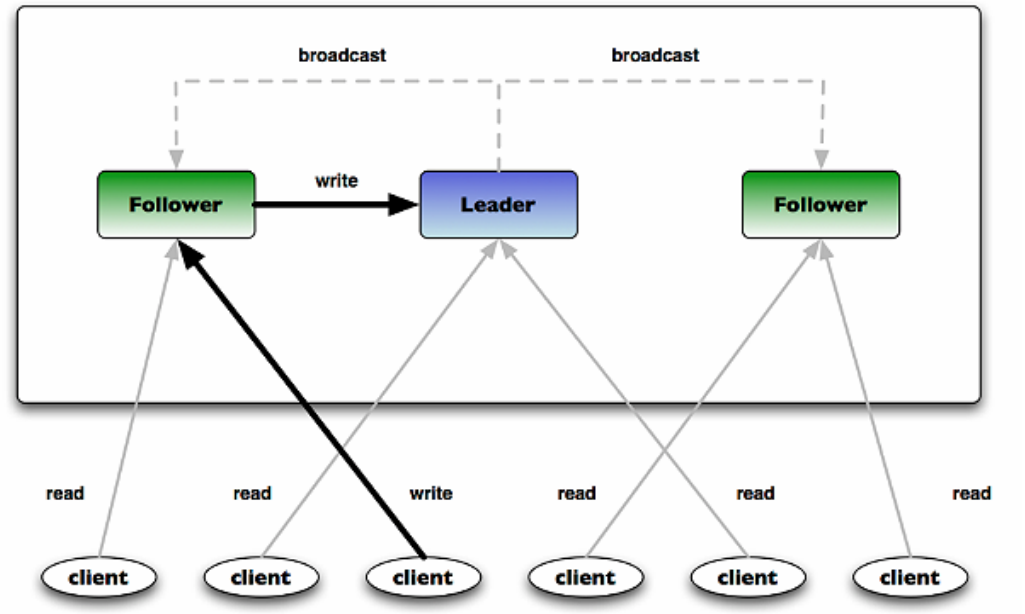

图 4：ZooKeeper Ensemble 架构

Kafka 中 ZooKeeper 的用途：正如 ZooKeeper 用于分布式系统的协调和促进，Kafka 使用 ZooKeeper 也是基于相同的原因。ZooKeeper 用于管理、协调 Kafka 代理。每个 Kafka 代理都通过 ZooKeeper 协调其它 Kafka 代理。当 Kafka 系统中新增了代理或者某个代理故障失效时，ZooKeeper 服务将通知生产者和消费者。生产者和消费者据此开始与其它代理 协调工作。Kafka 整体系统架构如图 5 所示。

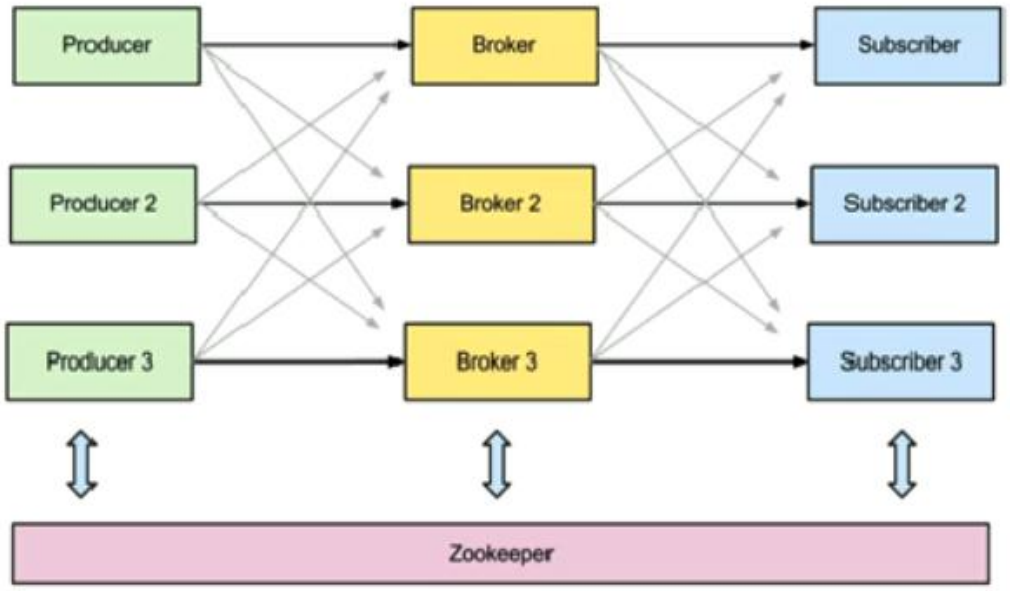

图 5：Kafka 分布式系统的总体架构

六．常用 Message Queue 对比

- RabbitMQ

  RabbitMQ 是使用 Erlang 编写的一个开源的消息队列，本身支持很多的协议：AMQP，XMPP, SMTP, STOMP，也正因如此，它非常重量级，更适合于企业级的开发。同时实现了 Broker 构架，这意味着消息在发送给客户端时先在中心队列排队。对路由，负 载均衡或者数据持久化都有很好的支持。

- Redis

  Redis 是一个基于 Key-Value 对的 NoSQL 数据库，开发维护很活跃。虽然它是一个 Key-Value 数据库存储系统，但它本身支持 MQ 功能，所以完全可以当做一个轻量级的队列服务来使用。对于 RabbitMQ 和 Redis 的入队和出队操作，各执行 100 万次，每 10 万次记录一次执行 时间。测试数据分为 128Bytes、512Bytes、1K 和 10K 四个不同大小的数据。实验表明：入队时，当数据比较小时 Redis 的性能要高于 RabbitMQ，而如果数据大小超过了 10K，Redis 则慢的无法忍受；出队时，无论数据大小，Redis 都表现出非常好的性能，而 RabbitMQ 的出队性能则远低于 Redis。

- ZeroMQ

  ZeroMQ 号称最快的消息队列系统，尤其针对大吞吐量的需求场景。ZeroMQ 能够实现 RabbitMQ 不擅长的高级/复杂的队列，但是开发人员需要自己组合多种技术框架，技术上的复杂度是对这 MQ 能够应用成功的挑战。ZeroMQ 具有一个独特的非中间件的模式，你不需要安装和运行一个消息服务 器或中间件，因为你的应用程序将扮演这个服务器角色。你只需要简单的引用 ZeroMQ 程序库，可以使用 NuGet 安装，然后你就可以愉快的在应用程序之间 发送消息了。但是 ZeroMQ 仅提供非持久性的队列，也就是说如果宕机，数据将会丢失。其中，Twitter 的 Storm 0.9.0 以前的版本中默认使用 ZeroMQ 作为数据流的传输（Storm 从 0.9 版本开始同时支持 ZeroMQ 和 Netty 作为传输模块）。

- ActiveMQ

  ActiveMQ 是 Apache 下的一个子项目。 类似于 ZeroMQ，它能够以代理人和点对点的技术实现队列。同时类似于 RabbitMQ，它少量代码就可以高效地实现高级应用场景。

- Kafka/Jafka

  Kafka 是 Apache 下的一个子项目，是一个高性能跨语言分布式发布/订阅消息队列系统，而 Jafka 是在 Kafka 之上孵化而来的，即 Kafka 的一个

升级版。具有以下特性：快速持久化，可以在 O(1)的系统开销下进行消息持久化；高吞吐，在一台普通的服务器上既可以达到 10W/s 的吞 吐速率；完全的分布式系统，Broker、Producer、Consumer 都原生自动支持分布式，自动实现负载均衡；支持 Hadoop 数据并行加载， 对于像 Hadoop 的一样的日志数据和离线分析系统，但又要求实时处理的限制，这是一个可行的解决方案。Kafka 通过 Hadoop 的并行加载机制统一了 在线和离线的消息处理。Apache Kafka 相对于 ActiveMQ 是一个非常轻量级的消息系统，除了性能非常好之外，还是一个工作良好的分布式系统。