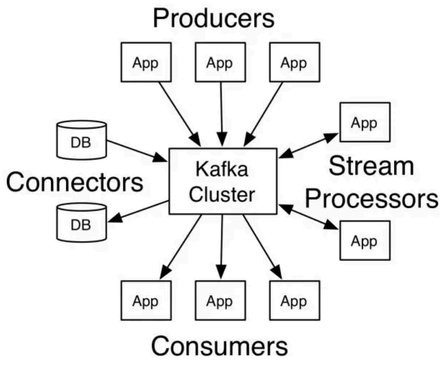
一. 简介

Kafka被用于两大类应用：
1. 在应用间构建实时的数据流通道
2. 构建传输或处理数据流的实时流式应用

几个概念
1. Kafka以集群模式运行在1或多台服务器上
2. Kafka以topics的形式存储数据流
3. 每一个记录包含一个key、一个value和一个timestamp

Kafka有4个核心API
1. Producer API：用于应用程序将数据流发送到一个或多个Kafka topics
2. Consumer API：用于应用程序订阅一个或多个topics并处理被发送到这些topics中的数据
3. Streams API：允许应用程序作为流处理器，处理来自一个或多个topics的数据并将处理结果发送到一个或多个topics中，有效的将输入流转化为输出流
4. Connector API：用于构建和运行将Kafka topics和现有应用或数据系统连接的可重用的produers和consumers。例如，如链接到关系数据库的连接器可能会捕获某个表所有的变更

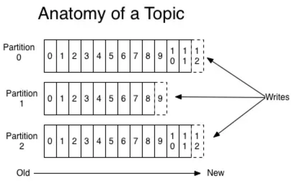


Kafka客户端和服务端之间的通信是建立在简单的、高效的、语言无关的TCP协议上的。此协议带有版本且向后兼容。我们为Kafka提供了Java客户端，但是客户端可以使用多种语言。

二. Topics and Logs

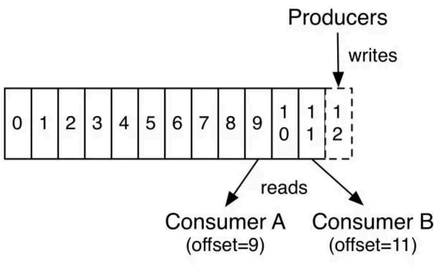
Topic是发布记录的类别。Kafka中的Topics一般是多订阅者的，也就是一个Topic可以有0个或多个Consumer订阅它的数据。

对于每个主题，Kafka会会维护一个如下所示的分区日志



每个分区是一个有序的，以不可变的记录顺序追加的Commit Log。分区中的每个记录都有一个连续的ID，称为Offset，唯一标识分区内的记录。

Kafka集群使用记录保存时间的配置来保存所有已发布的记录（无论他们是否被消费）。例如，配置策略为两天，那么在一条记录发布两天内，这条记录是可以被消费的，之后将被丢弃以腾出空间。Kafka的性能和数据量无关，所以存储长时间的数据并不会成为问题。



实际上唯一需要保存的元数据是消费者的消费进度，即消费日志的偏移量（Offset）。这个Offset是由Consumer控制的：通常消费者会在读取记录时以线性方式提升Offset，但是事实上，由于Offset由Consumer控制，因此它可以以任何顺序消费记录。例如一个Consumer可以通过重置Offset来处理过去的数据或者跳过部分数据。

这个特征意味着Kafka的Consumer可以消费“过去”和“将来”的数据而不对集群和其他Consumer不造成太大的影响。例如，可以使用命令行工具tail来获取Topic尾部的内容而不对已经在消费Consumer造成影响。

分区日志有几个目的。第一，使服务器能承载日志的大小，每个分区的日志必须可以被保存在单个服务器上，但是一个Topic可以拥有多个分区，那么它可以处理任意大小的数据量。第二，它们作为并行度的单位（更多的是这点的考虑）。

三. Distribution

分区日志分布在集群中服务器中，每个服务器处理一部分分区的数据和请求。每个分区可以配置分布的服务器，以实现容错。

每个分区拥有一个Leader节点，和零或多个Follower。Leader处理该分区所有的读写请求，Follower复制Leader数据。如果Leader节点宕机，将会有一个Follower节点自动的转化为Leader。每个节点成为其部分分区的Leader，并成为剩余分区的Follower，这样整个集群的负载将比较均衡。

四. Producers

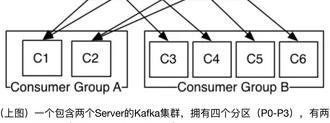
Producer发送数据到它选择的Topic。Producer负责决定将数据发送到Topic的那个分区上。这可以通过简单的循环方式来平衡负载，或则可以根据某些语义来决定分区（例如基于数据中一些关键字）。

五. Consumers

Consumer使用一个group name来标识自己的身份，每条被发送到一个Topic的消息都将被分发到属于同一个group的Consumer的一个实例中（group name相同的Consumer属于一个组，一个Topic的一条消息会被这个组中的一个Consumer实例消费）。Consumer实例可以在单独的进程中或者单独的机器上。

如果所有的Consumer实例都是属于一个group的，那么所有的消息将被均衡的分发给每个实例。

如果所有的Consumer都属于不同的group，那么每条消息将被广播给所有的Consumer。



（上图）一个包含两个Server的Kafka集群，拥有四个分区（P0-P3），有两个Consumer group：Group A和Group B。Group有C1、C2两个Consumer，GroupB有C3、C4、C5、C6四个Consumer。

更常见的是，Topic有少量的Consumer group，每一个都是“一个逻辑上的订阅者”。每个group包含多个Consumer实例，为了可伸缩性和容错性。这就是一个发布-订阅模式，只是订阅方是一个集群。

Kafka中消费的实现方式是“公平”的将分区分配给Consumer，每一个时刻分区都拥有它唯一的消费者。Consumer成员关系有Kafka程度动态维护。如果新的Consumer加入了分区，那么它会从这个分区其他的Consumer中分配走一部分分区：如果部分Consumer实例宕机，它的分区会被其他Consumer实例接管。

Kafka只保证同一个分区内记录的顺序，而不是同一个Topic的不同分区间数据的顺序。每个分区顺序结合按Key分配分区的能力，能满足大多数程序的需求。如果需要全局的顺序，可以使用只有一个分区的Topic，这意味着每个group只能有一个Consumer实例（因为一个分区同一时刻只能被一份Consumer消费－－多加的Consumer只能用于容错）。

六. Guarantees

Kafka高级API中提供一些能力

被一个Producer发送到特定Topic分区的消息将按照他们的发送顺序被添加到日志中。这意味着，如果M1、M2是被同一个Producer发送出来的，且M1先发送，那么M1拥有更小的Offset，在日志中的位置更靠前。

Consumer按照消息的存储顺序在日志文件中查找消息。

对于复制配置参数为N的Topic，我们能容忍N-1的服务器故障，而不会丢失已经Commit的数据。有关这些保证更详细的信息，参见文档的设计部分。

七. Kafka as a Messaging System

Kafka的流模式和传统的消息系统有什么区别?

消息传统上有两种模式：队列和发布-订阅。在队列中，一群Consumer从一个Server读取数据，每条消息被其中一个Consumer读取。在发布-订阅中，消息被广播给所有的Consumer。这两种模式有各自的优缺点。队列模式的优点是你可以在多个消费者实例上分配数据处理，从而允许你对程序进行“伸缩”。确定是队列不是多用户的，一旦消息被一个Consumer读取就不会再给其他Consumer。发布订阅模式允许广播数据到多个Consumer，那么就没办法对单个Consumer进行伸缩。

Kafka的Consumer group包含两个概念。与队列一样，消费组允许通过一些进程来划分处理（每个进程处理一部分）。与发布订阅一样，Kafka允许广播消息到不同的Consumer group。

Kafka模式的优势是每个Topic都拥有队列和发布-订阅两种模式。

Kafka比传统的消息系统有更强的顺序保证。

传统的消息系统在服务器上按顺序保存消息，如果多个Consumer从队列中消费消息，服务器按照存储的顺序输出消息。然后服务器虽然按照顺序输出消息，但是消息将被异步的传递给Consumer，所以他们将以不确定的顺序到达Consumer。这意味着在并行消费中将丢失消息顺序。传统消息系统通常采用“唯一消费者”的概念只让一个Consumer进行消费，但这就丢失了并行处理的能力。

Kafka做的更好一些。通过提供分区的概念，Kafka能提供消费集群顺序和负载的平衡。这是通过将分区分配个一个Consumer group中唯一的一个Consumer而实现的，一个分区只会被一个分组中的一个Consumer进行消费。通过这么实现，能让一个Consumer消费一个分区并按照顺序处理消息。因为存在多个分区，所有可以在多个Consumer实例上实现负载均衡。注意，一个分组内的Consumer实例数不能超过分区数。

八. Kafka as a Storage System

任何将发送消息和消费结构的消息队列都有效的用作一个消息的存储系统。不同的是Kafka是一个更好的存储系统。被写入到Kafka的数据将被写入磁盘并复制以保证容错。Kafka允许Producer等待确定，以保证Producer可以确认消息被成功持久化并复制完成。

Kafka使用的存储结构，使其提供相同的能力，无论是存储50KB或者50TB持久化数据。因为允许客户端控制读取的位置，可以将Kafka视为高性能，低延迟的日志存储、复制、传播的分布式系统。

九. Kafka for Stream Processing

仅仅是读写和存储流数据是不够的，Kafka的目标是对流失数据的实时处理。在Kafka中，Stream Producer从输入的Topic中读取数据，执行一些操作，生成输出流到输出的Topic中。

例如，零售的应用程序将收到销售和出货的输入流，并输出根据该数据计算的重排序和价格调整后的数据流。可以使用Producer和Consumer实现简单的处理。对于更复杂的转换，Kafka提供的完成的Stream API，允许构建将流中数据聚合或将流连接到一起的应用。