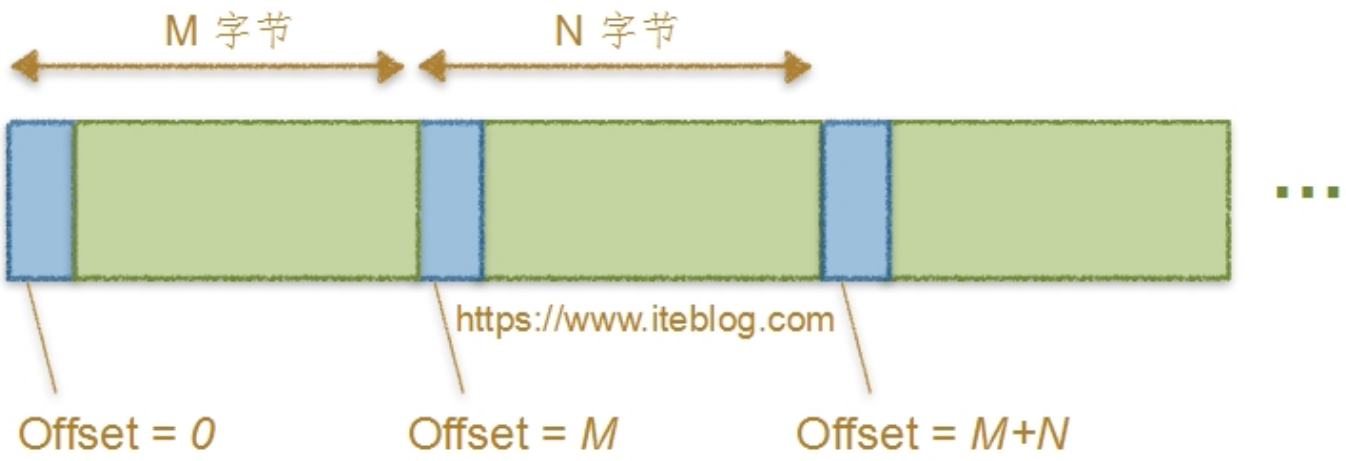
图解Apache Kafka消息偏移量的演变(0.7.x~0.10.x)

我在《Apache Kafka消息格式的演变(0.7.x~0.10.x)》文章中介绍了 Kafka 几个版本的消息格式。仔细的同学肯定看到了在 MessageSet 中的 Message 都有一个 Offset 与之一一对应，本文将探讨 Kafka 各个版本对消息中偏移量的处理。同样是从 Kafka 0.7.x 开始介绍，并依次介绍到 Kafka 0.10.x，由于 Kafka 0.11.x 正在开发中，而且消息格式已经和之前版本大不一样，所以这里不打算介绍。

Kafka 0.7.x

我在《Apache Kafka消息格式的演变(0.7.x~0.10.x)》文章中介绍 MessageSet 格式的时候就说Offset字段存储的是消息存储到磁盘之后的物理偏移量；注意，这里是物理偏移量，什么意思呢？看下面的图大家应该就会明白：

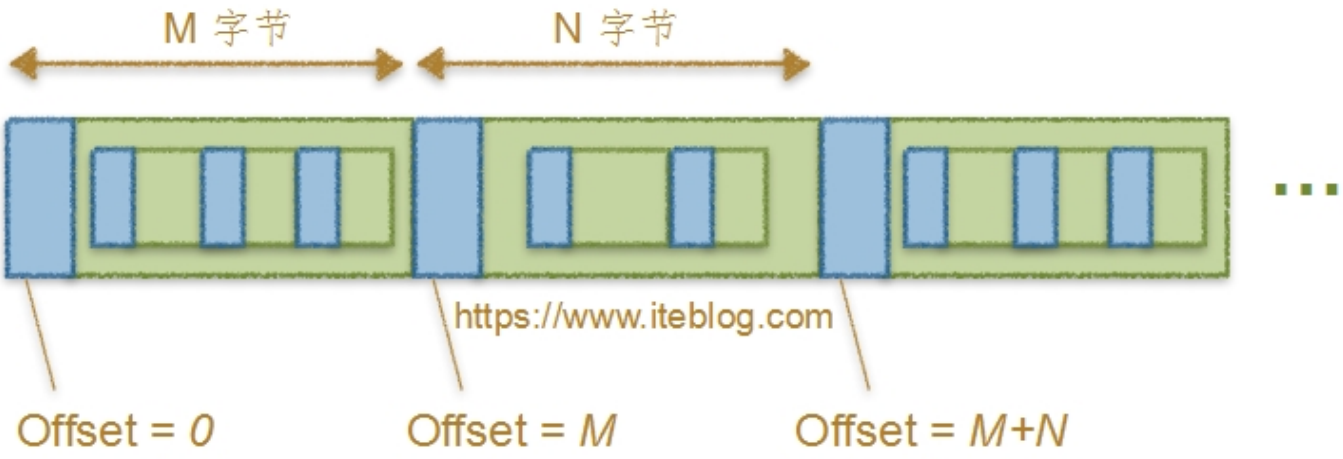


图一

从上图可以看出，每条消息存在磁盘的偏移量是其距离文件开头的绝对偏移量。比如上面第一条消息的偏移量是0；第二条消息的偏移量是第一条消息的总长度；第三条消息是其前两条消息总长度；以此类推。这种方式存储消息的偏移量很好理解，处理起来也很方便。

大家需要注意，消息存储到磁盘的偏移量是由 Broker 处理完成的，原因很简单，因为只有 Broker 端才知道现在 Log 的最新偏移量； Producer 端是无法获取的。这个逻辑同样适用于 Kafka 0.8.x 、Kafka 0.9.x 以及 Kafka 0.10.x。

上面仅仅是非压缩消息的偏移量处理，我们来看看这个版本压缩消息的偏移量处理是咋样的，如下图所示：



图二

正如上图所示，压缩消息内部的子消息并不设置偏移量，外部的消息偏移量设置规则和非压缩消息逻辑一致。

优缺点

这种设计存在几个问题：

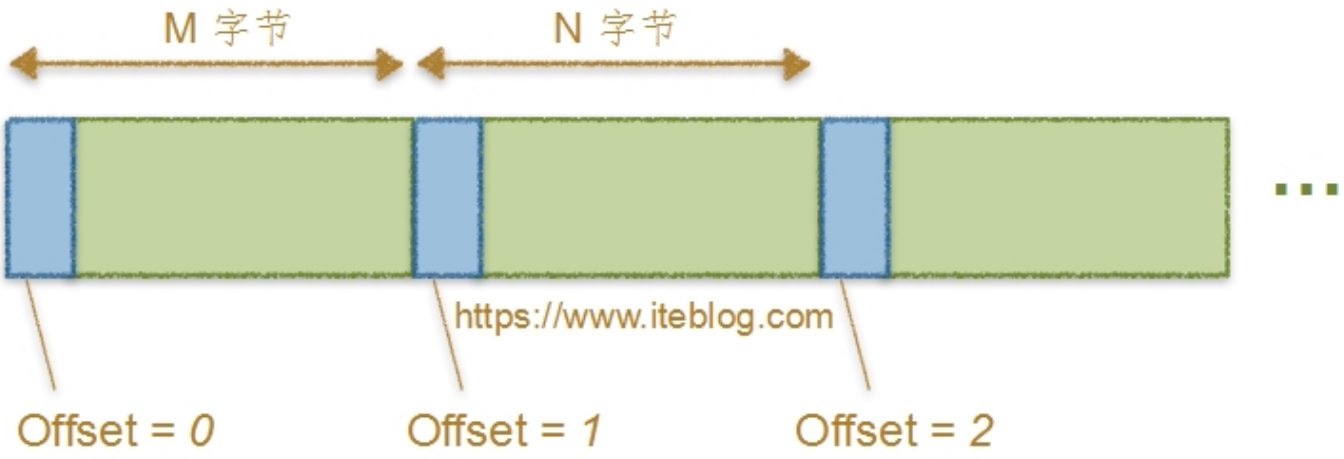
- 很难对压缩消息内部的消息进行checkpoint；
- 很难对压缩消息内部的消息进行定位操作；
- log compaction 不好做。

但是这样的设计也有好处：

- Broker 处理来自 Producer 的消息速度非常快
- CPU 的利用率一般
- 一般网络是这里的主要瓶颈。

Kafka 0.8.x

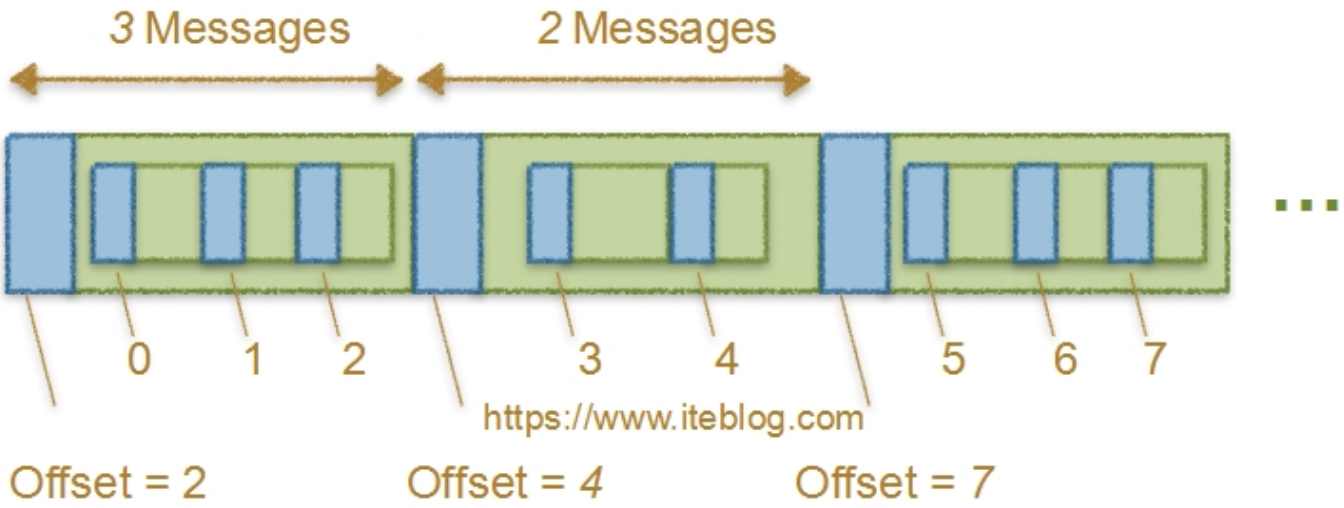
针对 Kafka 0.7.x 版本消息偏移量存在的多种问题，这个版本对其进行了解决，这个版本消息偏移量处理结果如下：



图三

我们先不管这个消息格式到底有何变化（真想知道的话，请参见《Apache Kafka消息格式的演变(0.7.x~0.10.x)》），上图很明显的一个变化就是偏移量的指已经不是消息的物理偏移量了，而是一个绝对偏移量，这个偏移量从0开始。第一条消息的绝对偏移量是0；第二条消息的绝对偏移量是1；依次类推。同样，这个偏移量的计算也是由 Broker 处理的。

压缩消息偏移量的处理逻辑就比这个复杂多了，先来看看一张结果图吧：

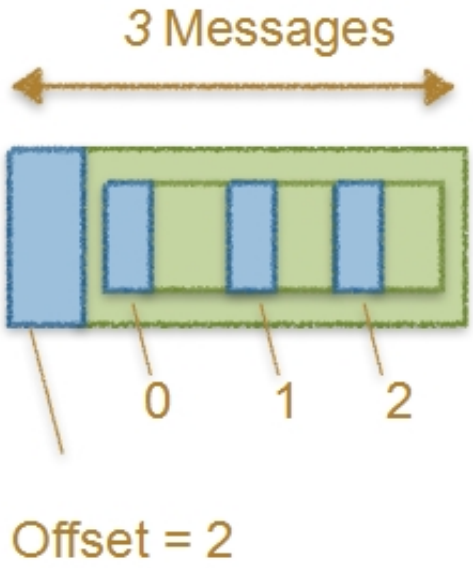


图四

这个图相对于 Kafka 0.7.x压缩消息的最明显变化就是，压缩消息内部的消息也有偏移量了！对于压缩消息的偏移量处理相对于 Kafka 0.7.x 复杂多了，下面我们将详细介绍 Kafka 是如何处理的：

Producer端对于压缩消息偏移量处理

Producer 端会对压缩消息中内部的消息设置一个相对偏移量。从0开始，依次到n-1，这里的n代表压缩消息的条数。处理的效果如下：

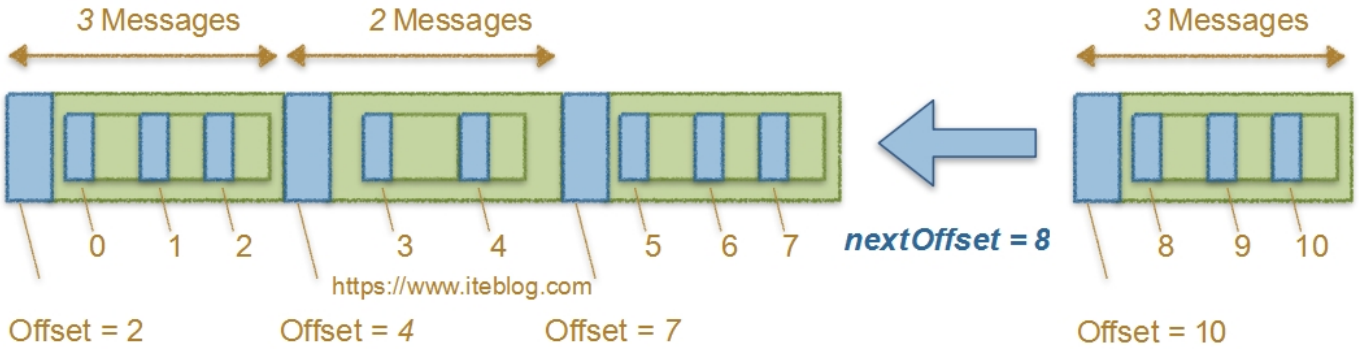


图五

偏移量设置好之后，Producer 端会将整个 MessageSet 进行压缩，然后发送到Broker。

Broker端对于压缩消息偏移量处理

Broker 端接收到 Producer 发送过来的压缩消息，忽略掉 Producer 端对压缩消息偏移量的而处理，其会先解压接收到的压缩消息，然后根据 nextOffset 依次设置压缩消息内部消息的偏移量，最后整个压缩消息的偏移量为最后一条内部消息的绝对偏移量。举个例子，比如图四最后一条消息的偏移量是7，那么 nextOffset 应该为 8；现在 Broker 接收到图五的消息，最后的处理如下：



图六

偏移量设置完之后， Broker 需要重新压缩刚刚解压好的消息，最后会将这条消息追加到 Log

文件中。

Client端对于压缩消息偏移量处理

Client 端如果请求压缩的消息，Broker 端会直接将整个压缩的消息发送到
Client，Client会自动将压缩的消息解压，解压的过程对我们编程的人来说是无感知的。

问题：为什么整个压缩消息的偏移量为最后一条内部消息的绝对偏移量呢？

这样设计其实是有目地的，由于 FetchRequest 协议中的 offset 是要求 Broker 提供大于等于这个 offset 的消息，因此 Broker 会检查log，找到符合条件的，然后传输出去。那么由于FetchRequest中的offset位置的消息可位于一个compressed
message中，所以broker需要确定一个compressed Message是否需要被包含在respone中。

- 如果我们将整个压缩消息的偏移量为第一条内部消息的绝对偏移量。那么，我们对于这个Message是否应包含在response中，无法给出是或否的回答。比如 FetchRequest 中指明的开始读取的offset是14，而一个compressed Message的offset是13，那么这个Message中可能包含offset为14的消息，也可能不包含。
- 如果我们将整个压缩消息的偏移量为最后一条内部消息的绝对偏移量。那么，可以根据这个offset确定这个Message应不应该包含在response中。比如 FetchRequest 中指明的开始读取的offset是14，那么如果一个compressed Message的offset是13，那它就不该被包含在response中。而当我们顺序排除这种不符合条件的Message，就可以找到第一个应该被包含在response中的Message（压缩或者未压缩），从它开始读取。

在第一种情况下（最小offset），我们尽管可以通过连续的两个Message确定第一个Message的offset范围，但是这样在读取时需要在读取第二个Message的offset之后跳回到第一个Message, 这通常会使得最近一次读(也就读第二个offset)的文件系统的缓存失效。而且逻辑比第二种情况更复杂。在第二种情况下，broker只需要找到第一个其offset大于或等于目标offset的Message，从它可以读取即可，而且也通常能利用到文件系统缓存，因为offset和消息内容有可能在同一个缓存块中。

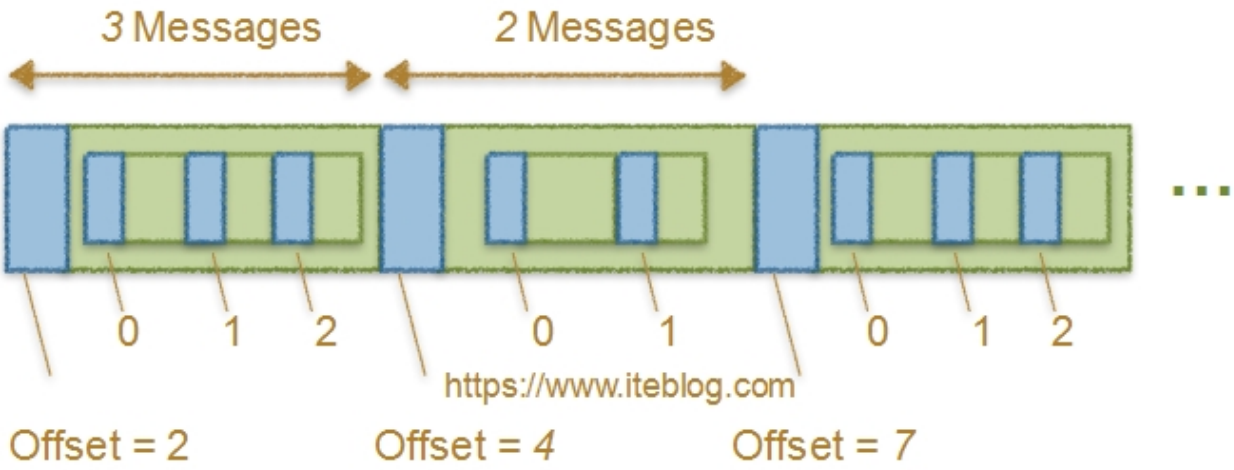
优缺点

这个版本的压缩消息中内部的消息也有偏移量了，这样就可以对内部消息进行定位处理。而且log compaction实现起来很方便。但是这个版本的消息偏移量也有个很明显的问题，就是对于每条压缩的消息，Broker 端都需要对其进行解压，设置好相关的偏移量之后，再进行压缩，这些都会占用很多的CPU资源。

Kafka 0.10.x

Kafka 0.10.x 对于非压缩的消息偏移量处理和 Kafka 0.8.x
一致，这里就不再介绍了。这里主要介绍 Kafka 0.10.x 对压缩消息偏移量处理逻辑。和 Kafka 0.8.x
处理内部消息偏移量逻
辑不一样，这个版本对于内部消息偏移量使用

的是相对偏移量，从0开始，依次到n-1，这里的n代表压缩消息的条数。所以 Kafka 0.10.x 压缩消息处理完偏移量之后看起来像下面的结果：



图七

从上图可以看出，相对于 Kafka 0.8.x 仅仅是内部消息偏移量变成了相对偏移量，整个压缩消息的偏移量处理逻辑和 Kafka 0.8.x 一致。下面我们将详细介绍 Kafka 是如何处理的：

Producer端对于压缩消息偏移量处理

这个逻辑和 Kafka 0.8.x 处理逻辑一致，不再介绍。有一点需要注意，Kafka 0.10.x 会将消息的 magic 值设置为 1，用于区分其他版本的消息，后面会介绍这样设置的用处。

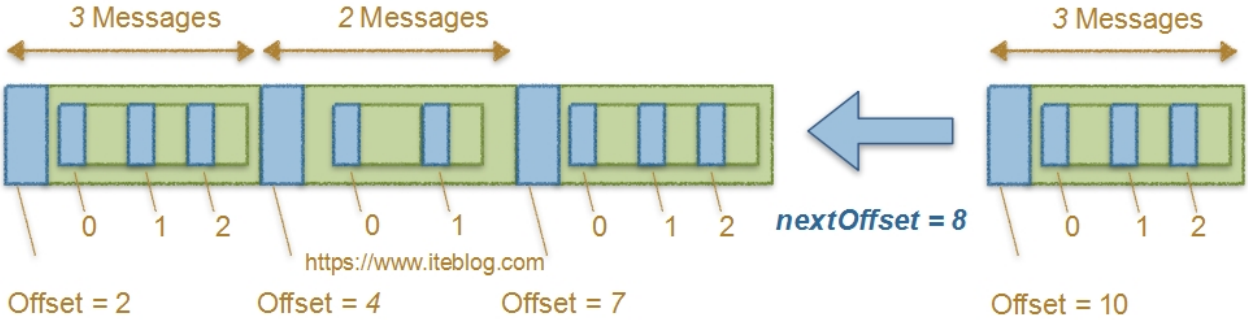
Broker端对于压缩消息偏移量处理

Broker 端接收到 Producer 发送过来的压缩消息，其也是先解压接收到的压缩消息，然后做一堆的判断，比如 消息的 magic 值是否大于0，压缩消息内部的消息偏移量值是否连续（0,1,2,3这样的）等，如果符合这些条件（inPlaceAssignment = true），那么 Broker 会直接处理整个压缩消息外部的偏移量，内部消息的偏移量不需要设置，因为这个在 Producer 端已经设置好了；并不需要再次压缩消息，最后会将这条消息追加到 Log 文件中。

如果 inPlaceAssignment = false，这时候会直接操作解压后的消息，并给压缩消息内部消息设置偏移量，最后设置整个压缩消息的偏移量；这时候会忽略掉 Producer 端为压缩消息设置的偏移量，包括内部消息和整个压缩消息的偏移量。整个处理逻辑分为两种情况：

- 如果接收到的消息不是由 Kafka 0.10.x 版本Producer客户端发送过来的，那么消息的 magic 值会等于0，这时候 Broker 设置偏移量逻辑和 Kafka 0.8.x 处理逻辑一致，也就是不管内部消息还是整个压缩消息的偏移量都是使用绝对偏移量；

- 如果接收到的消息是由 Kafka 0.10.x 版本Producer客户端发送过来的，那么消息的 magic 值会等于1，这时候 Broker
  会将压缩消息内部的消息偏移量设置成相对的，从0开始，依次到 n-1
  ，最后整个压缩消息的偏移量为 nextOffset + n -
  1，其中n为压缩消息的条数。处理结果如下：



图八

偏移量设置完之后，对于inPlaceAssignment = false，不管是由什么版本发送过来的消息，Broker 需要重新压缩刚刚解压好的消息，最后会将这条消息追加到 Log 文件中。

Client端对于压缩消息偏移量处理

对不同版本的 Client 请求， Broker 会做出不同的判断：对于非 Kafka 0.10.x 版本的 Consumer，Broker 端消息的发送不会使用零拷贝技术；而如果是 Kafka 0.10.x 版本的 Consumer，Broker 端消息的发送才会使用零拷贝技术