Kafka消息时间戳及压缩消息对时间戳的处理

《Apache Kafka消息格式的演变(0.7.x~0.10.x)》
《图解Apache Kafka消息偏移量的演变(0.7.x~0.10.x)》
《Kafka消息时间戳及压缩消息对时间戳的处理》

本博客的《Apache Kafka消息格式的演变(0.7.x~0.10.x)》文章中介绍了 Kafka 各个版本的格式变化。其中 Kafka 0.10.x 消息的一大变化是引入了消息时间戳的字段。本文将介绍 Kafka 消息引入时间戳的必要性以及压缩消息对时间戳的处理。

引入时间戳的必要性

理解消息引入时间戳的原因，首先我们需要知道 0.10.x 之前版本的 Kafka 存在的几个问题：

- 日志保存（Log retention）策略存在的问题：我们都知道，Kafka 会将接收到的消息持久化到磁盘；但是不可能只存不删，所以 Kafka 提供了机制会定期删除过期的日志。我们可以配置log.retention.hours、log.retention.minutes或log.retention.ms中的一个实现；但是这种删除策略是依据log segment的最后修改时间（last modification time）实现的。如果我们的集群执行过分区副本的重新分配（replica reassignment）操作，这时候新的log segment文件最后修改时间就是现在了。这时候我们就无法通过这种机制及时删除过期的日志。（关于log segment的删除原理请参见《Kafka日志删除源码分析》）
- 日志切分(log rolling)策略：和上面的原理类似。当前日志段文件会根据规则对当前日志进行切分——即，创建一个新的日志段文件，并设置其为当前激活(active)日志段。其中有一条规则就是基于时间的(log.roll.hours，默认是7天)，即当前日志段文件的最新一次修改发生于7天前的话，就创建一个新的日志段文件，并设置为active日志段。所以，它也有同样的问题，即最近修改时间不是固定的，一旦发生分区副本重分配，该值就会发生变更，导致日志无法执行切分。
- 流式处理(Kafka streaming)：Kafka 0.10.x 开始引入流处理（streaming processing），这种处理需要消息时间戳的概念。

基于上面的三种问题，引入时间戳势在必行。

Kafka消息时间戳

Kafka 0.10.x 消息的格式如下：

Message Format(Kafka 0.10.x)

Crc	Magic	Attributes	Timestamp	Key length	Key	Value length	Value

MessageSet Format

Offset	Size	Message	...	Offset	Size	Message

???????Spark?Hadoop??Hbase??????????????????iteblog_hadoop

目前 Kafka 消息支持两种类型的时间戳：CreateTime 和 LogAppendTime。分别表示 Producer 端消息的创建时间；以及 Broker 将这条消息写入到 Log Segment 的时间。这个时间就存在上图的 Timestamp 字段中（占8个字节）；同时attributes（占1个字节）字段的第 4 bit 位（从右往左数）代表消息的类型：0表示 CreateTime，1表示LogAppendTime。

Broker增加了两个参数：message.timestamp.type 和 max.message.time.difference.ms，其含义如下：

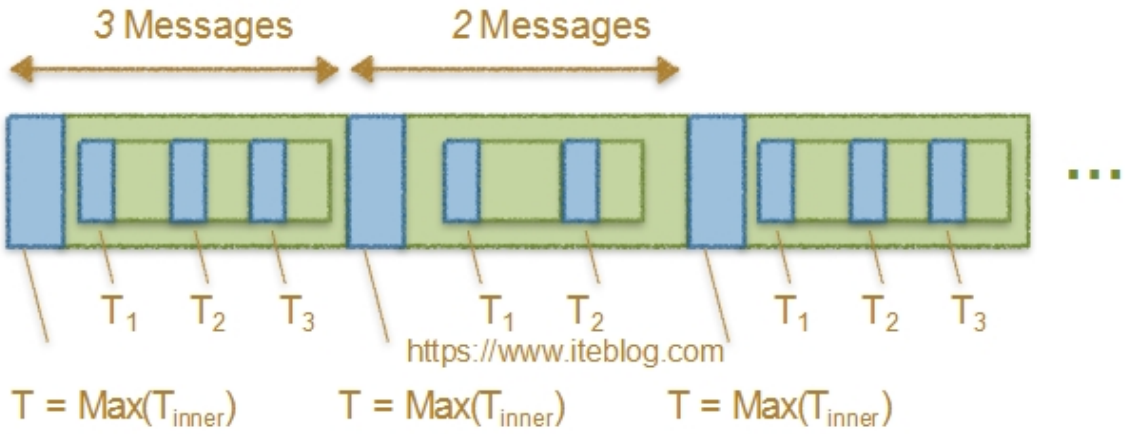
- message.timestamp.type： 这个参数在全局范围内设置 topic 的时间戳的类型。可以配置为 CreateTime 或者 LogAppendTime。当然，我们也可以在创建主题的时候单独设置时间戳类型。
- max.message.time.difference.ms ：如果 message.timestamp.type = CreateTime，Broker 仅仅接收那些消息时间戳和当前时间在 max.message.time.difference.ms 范围内的消息，超过了 max.message.time.difference.ms 配置范围的消息一律被 Broker 拒绝。

压缩消息时间戳的处理

在《图解Apache Kafka消息偏移量的演变(0.7.x~0.10.x)》文章中我介绍了压缩消息如何处理偏移量。现在我们来说说压缩消息对时间戳的处理。为了方便说明，我们把整个压缩消息称为Wrapper messages；压缩消息内部的消息为 Inner messages。压缩消息内部消息最大的 Create time 为 Max(T_{inner})。Kafka 对不同的时间戳类型有不同的处理方法，主要如下：

CreateTime

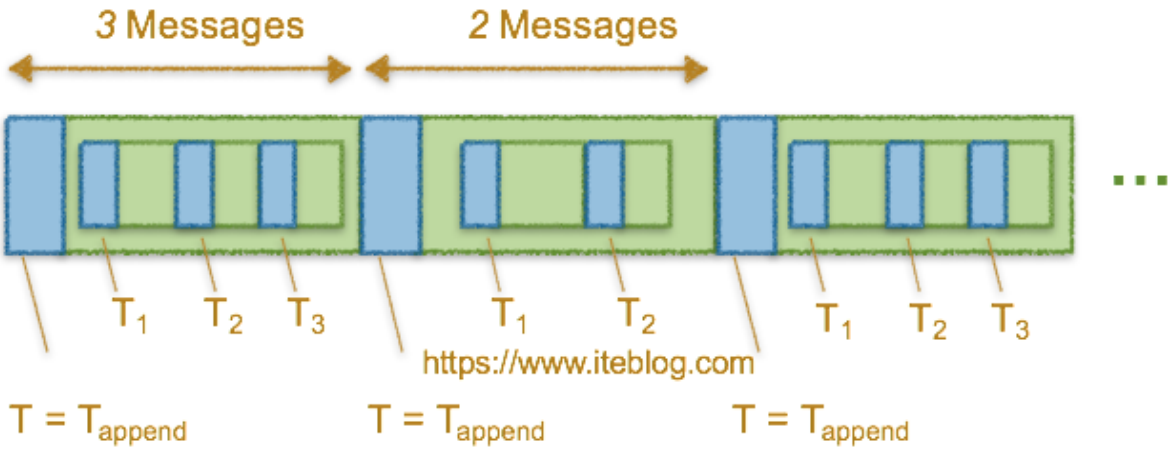
如果时间戳的类型是 CreateTime，那么 Wrapper messages 的时间戳处理结果如下：



从上图可以看出，整个Wrapper messages的时间戳为 Max(T_{inner})，也就是这条 Wrapper messages 中所有 Inner messages 的时间戳最大值。

LogAppendTime

如果时间戳的类型是 LogAppendTime，那么 Wrapper messages 的时间戳处理结果如下：



从上图可以看出，整个Wrapper messages的时间戳为当前 Broker 处理这条消息的时间T_{append}。

下面我们来分别讨论 Producer、 Broker 以及 Client 对压缩消息时间戳的处理。

Producer端对于压缩消息时间戳处理

不管时间戳的类型是什么，Producer端会将压缩消息内部的所有消息时间戳类型设置为CreateTime。但是 Inner messages 的时间戳类型仅会在 Broker 端用于校验（对于压缩的消息，Broker 会校验Inner messages 的时间戳类型是否为CreateTime，如果不是这条消息将会被 Broker 拒绝。），其他情况不会使用到。

Broker端对于压缩消息时间戳处理

Broker收到压缩消息的时候，会做如下处理：

- 如果 message.timestamp.type = LogAppendTime，Broker会用自己的本地时间覆盖Wrapper messages的时间戳，并且把 Wrapper messages 的 timestamp type 位置为1，Broker会忽略 Inner messages 的时间戳。在使用 LogAppendTime 时，之所以不修改每个内部消息的时间戳，是为了避免重新压缩带来的性能损失。
- 如果 message.timestamp.type = CreateTime，这时候判断会多一些，主要如下：
    - 如果消息的时间和当前 Broker 的时间差在 max.message.time.difference.ms 之内，那么 Broker 将会接收这个消息并且把它追加到 log Segment 中。对于压缩后的消息，Broker 将会把压缩后消息的时间戳更新为 Inner messages 的最大的Max(T_{inner})。
    - 如果时间差超过了 max.message.time.difference.ms, Broker 将会以 TimestampExceededThresholdException 的形式拒绝整批消息。

Consumer端对于压缩消息时间戳处理

Consumer 收到压缩消息时，会做如下处理：

- 如果 Wrapper messages 的时间戳类型属性位是0(CreateTime)，那么将会使用内部消息的时间戳
- 如果 Wrapper messages 的时间戳类型属性位是1，那么Wrapper messages的时间戳将会被用作内部消息的时间戳