Kafka 学习笔记

Kafka ISR/HW/LEO/LSO 概念理解

目录

ISR

- 理解
  Kafka中特别重要的概念，指代的是AR中那些与Leader保持同步的副本集合。 在AR中的副本可能不在ISR中，但Leader副本天然就包含在ISR中。
- 原理
  ISR (in sync replica) , 与leader副本保持同步状态的副本集合。
  - Leader对IRS中的节点进行track，当follower挂掉、卡住、落后时会被移出ISR 。
  - 依据replica.lag.time.max.ms参数判断 stuck and lagging replicas。
  - 当所有ISR中副本都将某条消息写入log后，可以确定该消息已被提交，并且只有被已提交的消息才会被消费者消费到。

HW

- 概念
  HW（High watermark）：高水位值，这是控制消费者可读取消息范围的重要字段。一个普通消费者只能“看到”Leader副本上介于Log Start Offset和HW（不含）之间的所有消息。水位以上的消息是对消费者不可见的。
- HW更新
  - follower HW
    follower HW 更新遵从最开始说的那个规律，在日志成功写入，LEO更新之后，就会尝试更新自身HW的值的，这个尝试发生在收到FETCH响应时会比较本地HW值和leader中的HW值，选择小的作为自身的HW值，所以说follower的HW值。但是follower 的HW值，说实话并没有什么卵用，说到用处的话应该是为称为leader做准备吧。相对来说leader 的HW值才是业务中所关心的，它决定了consumer端可消费的进度。

- producer 产生消息并且LEO成功更新
  HW的值可能会尝试更新（这需要根据ISR的同步策略来确定），然后还有leader在处理FETCH的请求时也会尝试更新。另外还有就是follower时、某个副本被提出ISR时都会尝试更新对应的HW值。这四种情形里面，最常见的就是接受FETCH请求时，通过比较自己的LEO值与缓存的其他的follower的LEO值，选择其中最小的LEO值来作为HW值，所以说HW值实际上就是ISR中最小的副本的LEO值啦
- leader HW
  其中leader 是通过follower 的offset来确定follower上次的消息是否写入的，所以就导致，remote LEO是比leader LEO小1的（更新remote LEO更新的其实是上次操作的结果），这就导致了如果最后一个follower同步完成时，HW实际上是未被更新的，得等到第二次FETCH请求才能完成HW的的更新（也就是说 第一轮FETCH完成了消息的同步（但leader 对于folllower是否成功保存毫不知情），第二轮完成了消息同步的同时完成上一轮的HW值的更新）

LEO

- 概念
  日志末端位移值或末端偏移量，表示日志下一条待插入消息的位移值。举个例子，如果日志有10条消息，位移值从0开始，那么，第10条消息的位移值就是9。此时，LEO = 10。

LSO

- 概念
  这是Kafka事务的概念。如果你没有使用到事务，那么这个值不存在（其实也不是不存在，只是设置成一个无意义的值）。该值控制了事务型消费者能够看到的消息范围。它经常与Log Start Offset，即日志起始位移值相混淆，因为有些人将后者缩写成LSO，这是不对的。在Kafka中，LSO就是指代Log Stable Offset。

Kafka的哪些场景中使用了零拷贝

- 概念
  在Kafka中，体现Zero Copy使用场景的地方有两处：基于mmap的索引和日志文件读写所用的TransportLayer。
  - 索引都是基于MappedByteBuffer的，也就是让用户态和内核态共享内核态的数据缓冲区，此时，数据不需要复制到用户态空间。不过，mmap虽然避免了不必要的拷贝，但不一定就能保证很高的性能。在不同的操作系统下，mmap的创建和销毁成本可能是不一样的。很高的创建和销毁开销会抵消Zero Copy带来的性能优势。由于这种不确定性，在Kafka中，只有索引应用了mmap，最核心的日志并未使用mmap机制。
  - TransportLayer是Kafka传输层的接口。它的某个实现类使用了FileChannel的transferTo方法。该方法底层使用sendfile实现了Zero Copy。对Kafka而言，如果I/O通道使用普通的PLAINTEXT，那么，Kafka就可以利用Zero Copy特性，直接将页缓存中的数据发送到网卡的

Buffer中，避免中间的多次拷贝。相反，如果I/O通道启用了SSL，那么，Kafka便无法利用Zero Copy特性了。

Kafka为什么不支持读写分离

- CAP理论
  我们只能保证C（可用性）和A（一致性）取其一，如果支持读写分离，那其实对于一致性的要求可能就会有一定折扣，因为通常的场景下，副本之间都是通过同步来实现副本数据一致的，那同步过程中肯定会有时间的消耗，如果支持了读写分离，就意味着可能的数据不一致，或数据滞后。
- Leader/Follower模型
  并没有规定Follower副本不可以对外提供读服务。很多框架都是允许这么做的，只是 Kafka最初为了避免不一致性的问题，而采用了让Leader统一提供服务的方式。
- Kafka 2.4
  Kafka提供了有限度的读写分离，也就是说，Follower副本能够对外提供读服务。

调优Kafka

- 调优方向
  吞吐量、延时、持久性和可用性，每种目标之前都是由冲突点，这也就要求了，我们在对业务接入使用时，要进行业务场景的了解，以对业务进行相对的集群隔离，因为每一个方向的优化思路都是不同的，甚至是相反的。
- 确定了目标
- 明确优化的维度
- 有些调优属于通用的优化思路，
  比如对操作系统、JVM等的优化；有些则是有针对性的，比如要优化Kafka的TPS。我们需要从3个方向去考虑：
- Producer端
  增加 `batch.size` 和 `linger.ms` ，启用压缩，关闭重试
- Broker端
  增加 `num.replica.fetchers` 提升 `-Follower` 同步 `TPS` ，避免 `Broker Full GC` 等。
- Consumer
  增加 `fetch.min.bytes`