(八)Kafka消息的可靠性保证

Natasha_ LV.3
2020年11月16日 15:34 · 阅读 1130

关注

1. 多副本数据同步策略

为了提高消息的可靠性，Kafka每个Topic的partition都有N个副本。这N个副本中，其中一个replica是Leader，其他都是Follower。在 Kafka 中追随者副本是不对外提供服务的，所有的请求都必须由领导者副本来处理。

而Follower 副本唯一的工作就是不断地从 Leader 副本拉取消息，然后写入到自己的提交日志中。如果这个同步过程的速度持续慢于 Leader 副本的消息写入速度，那么在 `replica.lag.time.max.ms` 时间后，此 Follower 副本就会被认为是与 Leader 副本不同步的，因此不能再放入 ISR 中。此时，Kafka 会自动收缩 ISR 集合，将该副本踢出ISR。

Leader与Follower中，都会维护各自的HW，对于新消息的写入，Consumer并不能立即被消费，需要等待ISR中的Followers从Leader中完成复制。其中Consumer只能消费HW之前的数据。

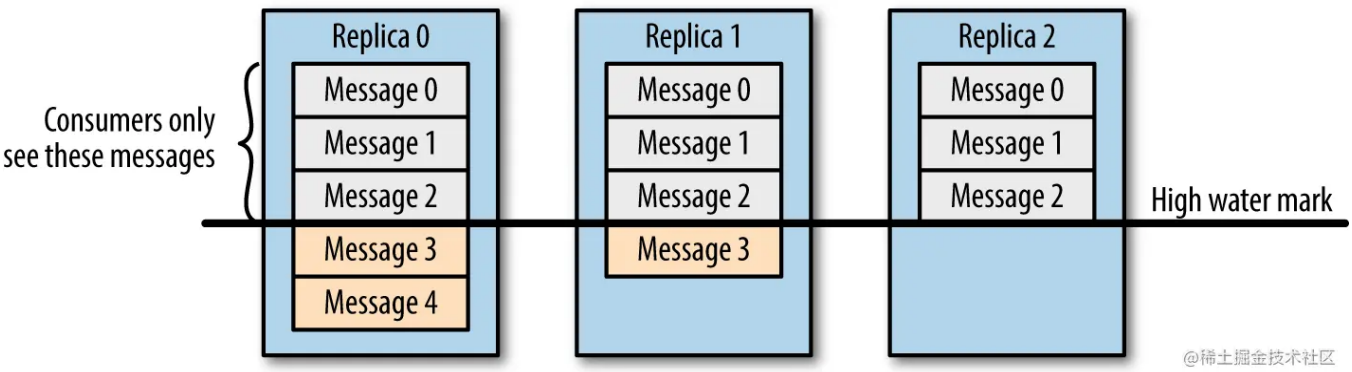
> 在Kafka中维护了 一个AR列表，AR又分为ISR和OSR ，是一个包括所有的分区的副本。
>
> 只有ISR内的副本都同步了leader中的数据，该数据才能被提交，才能被消费者访问。OSR内的副本是否同步了leader的数据，都不影响数据的提交，OSR内的follower 尽力的去同步leader，可能数据版本或落后。
>
> 最开始所有的副本都在ISR中，随着在kafka工作的过程中，如果某个副本同步速度慢于 `replica.lag.time.max.ms` 指定的阈值，则被踢出ISR，存入OSR，如果后续速度恢复可以回到ISR中。
>
> LEO - LogEndOffset： 分区的最新的数据的offset。
>
> HW - HighWatermark： 只有写入的数据被同步到所有的ISR中的副本后，数据才认为已提交，HW更新到该位置，HW之前的数据才可以被消费者访问，保证没有同步完成的数

> 在leader宕机后，只能从ISR列表中选取新的leader，无论ISR中哪个副本被选为新的leader都知道HW之前的数据，可以保证在切换了leader后，消费者可以继续看到之前已经提交的数据。

2. 数据一致性

对于“Consumer只能消费HW之前的数据”，这样做的原因是还没有被足够多副本复制的消息被认为是“不安全”的，如果 Leader 发生崩溃，另一个副本成为新 Leader，那么这些消息很可能丢失了。

如果我们允许消费者读取这些消息，可能就会破坏一致性。试想，一个消费者从当前 Leader（副本0） 读取并处理了 Message4，这个时候 Leader 挂掉了，选举了副本1为新的 Leader，这时候另一个消费者再去从新的 Leader 读取消息，发现这个消息其实并不存在，这就导致了数据不一致性问题。



当然，引入了 High Water Mark 机制，会导致 Broker 间的消息复制因为某些原因变慢，那么消息到达消费者的时间也会随之变长（因为我们会先等待消息复制完毕）。延迟时间可以通过参数 `replica.lag.time.max.ms` 参数配置，它指定了副本在复制消息时可被允许的最大延迟时间。

3. 副本故障处理

3.1 Follower故障

Follower故障后会被临时踢出ISR，当Follower恢复后，Follower会读取本地磁盘记录的上次的HW，并将log文件高于HW的部分截取掉，从HW开始向Leader同步。等该Follower的LEO大于等于该partition的HW，即Follower追上Leader之后，就会被重新加入ISR。

3.2 Leader故障

Leader发生故障，会从ISR中选举出一个新的Leader，其余的Follower会先将各自的log文件高于各自HW的部分截取掉，之后从新的Leader同步数据。

4. Leader选举

kafka会在Zookeeper中为每个partition动态的维护着ISR,当Leader挂掉后，会从ISR中顺序选择一个Follower作为主。 如果碰巧ISR中Follower全部挂掉,那么有两种选择：

- 等待ISR中任意Follower恢复，选定其为Leader。
- 选择第一个恢复的Follower作为Leader，这个Follower不一定在ISR中。

具体Kafka的副本机制，可以参考上一篇： (六)Kafka的副本机制

5. Kafka 在 Producer 的 ACK 消息确认机制

- acks = 0：意味着如果生产者能够通过网络把消息发送出去，那么就认为消息已成功写入 Kafka 。

  在这种情况下还是有可能发生错误，比如发送的对象无能被序列化或者网卡发生故障，但如果是分区离线或整个集群长时间不可用，那就不会收到任何错误。

> 在 acks=0 模式下的运行速度是非常快的（这就是为什么很多基准测试都是基于这个模式），你可以得到惊人的吞吐量和带宽利用率，不过如果选择了这种模式， 一定会丢失一些消息。

- acks = 1：意味若 Leader 在收到消息并把它写入到分区时会返回确认或错误响应。

  不过在这个模式下仍然有可能丢失数据，比如消息已经成功写入 Leader，但在消息被复制到 follower 副本之前 Leader发生崩溃。但发送消息的客户端却认为消息已成功写入，可是消费者看不到丢失的消息，所以此时的系统仍然是一致的。但从生产者角度来看，它丢失了一个消息。

  > 在这个模式下，如果发生正常的 Leader 选举，生产者会在选举时收到一个 `LeaderNotAvailableException` 的可恢复异常，如果生产者能恰当地处理这个错误，它会重试发送悄息，最终消息会安全到达新的 Leader 那里。
  >
  > 遇到不可恢复异常会抛出，这时可以捕获异常记录到数据库或缓存，进行单独处理。

- acks = all / -1：意味着 Leader 在返回确认或错误响应之前，会等待所有同步副本都收到悄息。

  如果和 `min.insync.replicas` 参数结合起来，就可以决定在返回确认前ISR至少中有多少个副本同步消息，生产者会一直重试直到消息被成功提交。不过这也是最慢的做法，因为生产者在继续发送其他消息之前需要等待所有副本都收到当前的消息。

另外，Producer 发送消息还可以选择同步（ `producer.type=sync` ） 或者异步（ `producer.type=async` ）模式。如果设置成异步，虽然会极大的提高消息发送的性能，但是这样会增加丢失数据的风险。如果需要确保消息的可靠性，必须将 `producer.type` 设置为 sync。

6. Kafka服务端数据丢失的解决方法

有一个比较常见的一个场景，就是 Kafka 某个 broker 宕机，然后重新选举 partition 的 leader。大家想想，要是此时其他的 follower 刚好还有些数据没有同步，结果此时 leader 挂了，然后选举某个 follower 成 leader 之后，不就少了一些数据?

所以此时一般是要求起码设置如下：

- topic 级别：设置 `replication.factor > 1` ，既要求每个 partition 必须有至少 2 个副本。

- broker 级别：设置 `min.insync.replicas > 1` ，这个是要求一个 leader 至少感知到有至少一个 follower 还跟自己保持联系，没掉队，这样才能确保 leader 挂了还有一个 follower 。关闭不完全的 Leader 选举，即 `unclean.leader.election.enable=false` 。

- producer 级别：设置 `acks=all` ：这个是要求每条数据，必须是写入所有 replica 之后，才能认为是写成功了。设置 `retries=MAX` （无限次重试）：这个是要求一旦写入失败，就无限重试，卡在这里了。设置 发生模式为同步 `producer.type=sync` 。

对于一致性要求高的业务按照上述要求配置之后，至少在 Kafka broker 端就可以保证在 leader 所在 broker 发生故障，进行 leader 切换时，数据不会丢失。

7. 生产者的重试参数

生产者需要处理的错误包括两部分：一部分是生产者可以自动处理的错误，还有一部分是需要开发者手动处理的错误。错误响应码可以分为两种：

- 可重试错误：重试之后可以解决的。如果broker返回的是 `LEADER_NOT_AVAILABLE` 错误，生产者可以尝试重新发送消息，也许这个时候一个新的首领被选举出来了，那么这次发送就成功了。
- 不可重试错误：无法通过重试解决的。如果broker返回的是 `INVALID_CONFIG` 错误，即使通过重试也无法改变配置选项，所以这样的重试是没有意义的。

一般情况下，如果目标是不丢失任何消息，那么最好让生产者在遇到可重试错误时能够保持重试。因为像首领选举或网络连接这类问题都可以在几秒之内得到解决，如果生产者保持重试，开发者就不需要额外去处理这些问题了。

但是，如果重试的是发送一个已经写入但是返回ack失败的消息，那会带来一些风险。例如，生产者因为网络问题没有收到broker的确认，但实际上消息已经写入成功，生产者会认为网络出现了临时故障，就重试发送该消息（因为它不知道消息已经写入）。

在这种情况下，broker会收到两个相同的消息。这时我们需要恰当地处理来保证每个消息至少被保存一次。实际中很多应用程序可以做到消息的幂等：在消息里加入唯一标识符，消费者在读取消息时检测重复消息并对它们进行处理。也就是说，即使出现了重复消息，也不会对处理结果的正确性造成负面影响。

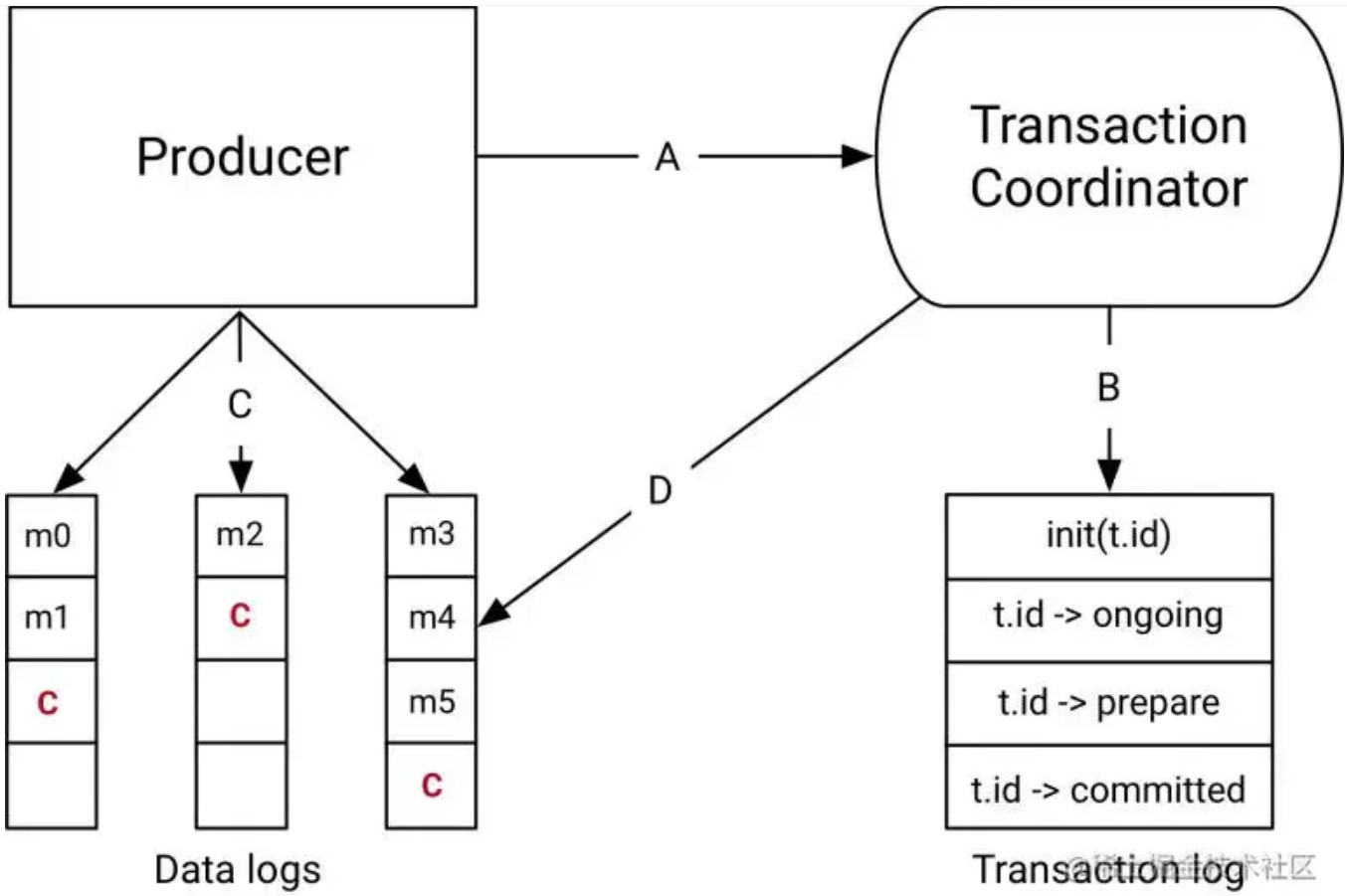
8. 生产者发送重复消息的幂等性解决方法

kafka 0.11.0.0版本引入了 `idempotent producer` 机制，在这个机制中同一消息可能被producer发送多次，但是在broker端只会写入一次，他为每一条消息编号去重，而且对kafka开销影响不大。

启动kafka的幂等性，无需修改代码，默认为关闭，需要修改配置文件：`enable.idempotence=true` 同时要求 `ack=all` 且 `retries > 1` ，这样幂等producer只能保证单分区上无重复消息。

> 幂等原理：
>
> 每个producer有一个producer id，服务端会通过这个id关联记录每个producer的状态，每个producer的每条消息会带上一个递增的sequence，服务端会记录每个producer对应的当前最大sequence，producerId + sequence。
>
> 如果新的消息带上的sequence不大于当前的最大sequence就拒绝这条消息，如果消息落盘会同时更新最大sequence，这个时候重发的消息会被服务端拒掉从而避免消息重复。

而多分区的情况，我们需要保证原子性的写入多个分区，即写入到多个分区的消息要么全部成功，要么全部回滚。这时候就需要使用事务，在producer端设置 `transcational.id` 为一个指定字符串，保证原子性地写入到多个分区。



总结

- 幂等(producer)：保证发送单个分区的消息只会发送一次，不会出现重复消息。
- 事务(transaction)：保证原子性地写入到多个分区，即写入到多个分区的消息要么全部成功，要么全部回滚流处理EOS。流处理本质上可看成是“读取-处理-写入”的管道，EOS保证整个过程的操作是原子性。(注意，这只适用于Kafka Streams)

9. 消费者数据丢失和重复消费的解决方案

唯一可能导致消费者弄丢数据的情况：你消费到了这个消息，然后消费者那边自动提交了 offset，让 Kafka 以为你已经消费好了这个消息，但其实你才刚准备处理这个消息，你还没处理，你自己就挂了，此时这条消息就丢失。

那么只要关闭自动提交 offset，在处理完之后自己手动提交 offset，就可以保证数据不会丢。但是此时还是可能会有重复消费，比如你刚处理完，还没提交 offset，结果自己挂了，此时肯定会重复消费一次。

一般的解决方案是让下游做幂等。建存储消息消费记录表，通过主键ID查询记录表，判断消息状态是否已消费，若没消费过，则处理消息，处理完后，更新消息记录的状态为已消费

分类： 后端 标签： Kafka

评论

输入评论（Enter换行，⌘ + Enter发送）

相关推荐

奔跑的毛球 2月前 后端

RabbitMQ保证消息可靠性

464 7 评论

一条coding 1月前 Spring Boot Kafka Java

SpringBoot集成Kafka——如此简单

3962 20 6

JavaGuide 2年前 Java Kafka

面试官问我如何保证Kafka不丢失消息?我哭了！

3.1w 201 32

二十六画生的博客 1年前 Kafka 消息队列 后端

Kafka 消息队列如何保证顺序性?

2697 9 评论

万俊峰Kevin 1年前 Go

一招让Kafka达到最佳吞吐量

620 5 评论

和耳朵 1年前 Java 后端

RabbitMQ高级之如何保证消息可靠性?

青南　2年前

每一位爬虫工程师都应该学习Kafka

790　2　评论

用户112986583106　3年前　Kafka

关于Kafka日志留存策略的讨论

1357　点赞　评论

用户7936126247356　1年前　Java

Kafka 居然还会丢消息?

177　点赞　评论

朱小厮　3年前　Kafka

为了追求极致的性能，Kafka掌控这11项要领

1437　16　2

hsfxuebao　2年前　Kafka

Kafka保证消息不丢失不重复

1256　点赞　评论

树哥聊编程　25天前　后端　Java

服务器宕机了，Kafka 消息会丢失吗?

3154　41　22

用户7936126247356　1年前　Java

Kafka 居然还会丢消息?

213　点赞　评论

枕邊書　3年前　RabbitMQ　消息队列

解决RabbitMQ消息丢失问题和保证消息可靠性（一）

1.5w　52　9

叫我书豪就行了　2年前　Kafka

Kafka的高性能设计要点

程sq 2月前 后端

kafka如何保证消息不丢失

556 5 评论

zxhtom 1年前 RabbitMQ

rabbitmq如何保证消息可靠性不丢失

2298 19 1

AI码师 1年前 分布式

如何保证消息可靠性?

156 1 评论

xindoo 11月前 Java Kafka 后端

7张图了解kafka基本概念

5072 39 15

iFangcy_ 2年前 Spring

RabbitMQ 如何保证消息可靠性

111 点赞 评论