聊聊 Kafka：Kafka 如何保证可靠性

原创 老周聊架构 已于 2022-06-03 17:32:42 修改 544 收藏 1 版权

分类专栏： 聊聊系列 Kafka 文章标签： kafka

一、前言

在如今的 分布式 环境时代，任何一款中间件产品，大多都有一套机制去保证高可用的，Kafka 作为一个商业级消息中间件，消息可靠性的重要性可想而知，那 Kafka 如何保证可靠性的呢？本文从 Producer 往 Broker 发送消息、Topic 分区副本以及 Leader 选举几个角度介绍 Kafka 是如何保证可靠性的。

二、Producer 往 Broker 发送消息

如果我们要往 Kafka 对应的主题发送消息，我们需要通过 Producer 完成。前面我们讲过 Kafka 主题对应了多个分区，每个分区下面又对应了多个副本；为了让用户设置数据可靠性， Kafka 在 Producer 里面提供了消息确认机制。把选项提供给用户自己去选择，我们可以通过配置来决定消息发送到对应分区的几个副本才算消息发送成功。可以在定义 Producer 时通过 acks 参数指定（在 0.8.2.X 版本之前是通过 `request.required.acks` 参数设置的，详见 KAFKA-3043）。

这个参数支持以下三种值：

- `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`： 意味着 Leader 在收到消息并把它写入到本地磁盘时会返回确认或错误响应，不管其它的 Follower 副本有没有同步过这条消息。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 异常，如果生产者能恰当地处理这个错误，它会重试发送消息，最终消息会安全到达新的 Leader 那里。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。
- `acks = all` （这个和 `request.required.acks = -1` 含义一样）： 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消息。如果和 `min.insync.replicas` 参数结合起来，就可以决定在返回确认前至少有多少个副本能够收到消息，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。

根据实际的应用场景，我们设置不同的 acks，以此保证数据的可靠性。

另外，Producer 发送消息还可以选择同步（默认，通过 producer.type=sync 配置） 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。

三、Topic 分区副本

在 Kafka 0.8.0 之前，Kafka 是没有副本的概念的，那时候人们只会用 Kafka 存储一些不重要的数据，因为没有副本，数据很可能会丢失。但是随着业务的发展，支持副本的功能越来越强烈，所以为了保证数据的可靠性，Kafka 从 0.8.0 版本开始引入了分区副本（详情请参见 KAFKA-50）。也就是说每个分区可以人为的配置几个副本（比如创建主题的时候指定 replication-factor，也可以在 Broker 级别进行配置 default.replication.factor），一般会设置为 3。

Kafka 可以保证单个分区里的事件是有序的，分区可以在线（可用），也可以离线（不可用）。在众多的分区副本里面有一个副本是 Leader，其余的副本是 Follower，所有的读写操作都是经过 Leader 进行的，同时 Follower 会定期地去 Leader 上的复制数据。当 Leader 挂了的时候，其中一个 Follower 会重新成为新的 Leader。通过分区副本，引入了数据冗余，同时也提供了 Kafka 的数据可靠性。

Kafka 的分区多副本架构是 Kafka 可靠性保证的核心，把消息写入多个副本可以使 Kafka 在发生崩溃时仍能保证消息的持久性。

四、Leader 选举

在介绍 Leader 选举之前，让我们先来了解一下 ISR（in-sync replicas）列表。每个分区的 Leader 会维护一个 ISR 列表，ISR 列表里面包括 Leader 副本和 Follower 副本的 Broker 编号，只有跟得上 Leader 的 Follower 副本才能加入到 ISR 里面，这个是通过 `replica.lag.time.max.ms` 参数配置的，这个参数的含义是 Follower 副本能够落后 Leader 副本的最长时间间隔，默认值是 10 秒。这就是说，只要一个 Follower 副本落后 Leader 副本的时间不连续超过 10 秒，那么 Kafka 就认为该 Follower 副本与 Leader 是同步的，即使此时 Follower 副本中保存的消息明显少于 Leader 副本中的消息。

我们在前面说过，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。

值得注意的是，倘若该副本后面慢慢地追上了 Leader 的进度，那么它是能够重新被加回 ISR 的。这也表明，ISR 是一个动态调整的集合，而非静态不变的。

所以当 Leader 挂掉了，而且 unclean.leader.election.enable=false 的情况下，Kafka 会从 ISR 列表中选择第一个 Follower 作为新的 Leader，因为这个分区拥有最新的已经 committed 的消息。通过

这个可以保证已经 committed 的消息的数据可靠性。

五、思考

Q1： 对于数据的读写操作都在 Leader 副本中，Follower 副本只从 Leader 副本复制数据，不对外提供数据读写操作，只做数据冗余保证可靠性。假如 Follower 副本还没同步完，此时 Leader 副本挂掉了，怎么保证数据的可靠性?

A1： 这就要看上面说的 Producer 通过 acks 参数，根据自己的业务场景，设置不同的参数，以此保证数据的可靠性。

Q2： acks = 1，意味着 Leader 在收到消息并把它写入到本地磁盘时会返回确认或错误响应，不管其它的 Follower 副本有没有同步过这条消息。假如 Follower 副本还没同步完，此时 Leader 副本挂掉了，怎么保证数据的可靠性?

A2： 这就要看上面说的 Producer 通过 acks 参数，根据自己的业务场景，设置不同的参数，以此保证数据的可靠性。

Q3： acks = all，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消息。如果 Follower 副本同步完成后，Broker 给 Producer 返回 ack 之前，Leader 副本挂掉了，怎么保证数据的可靠性?

A3： 数据发送到 Leader 副本后，部分 ISR 同步完成，也就是部分在 ISR 列表中的 Follower 副本同步完成后，此时 Leader 副本挂掉了，每个 Follower 副本都有可能成为新的 Leader 副本，Producer 端会返回异常，可以设置让 Producer 端重新发送数据来保证数据的可靠性，不过可能会造成数据的重复，如果要保证数据的一致性的话，业务下游需要做幂等操作，防止数据重复消费。

Q4： acks = all，就可以代表数据一定不会丢失吗?

A4：

- Partition 只有一个副本，也就是只有一个 Leader 副本，没有任何一个 Follower 副本的时候，接收完消息后宕机，还是会存在数据丢失。
- acks = all，必须跟 ISR 列表里至少有两个以上的副本配合使用，min.insync.replicas 这个参数是设置 ISR 中最小副本数是多少，默认值是 1，改为 >= 2，如果 ISR 列表中的副本数小于 min.insync.replicas 配置的数量时，Producer 客户端会返回异常。

Q5： acks = all 时，ISR 列表中有三个副本，一个 Leader 副本，两个 Follower 副本。min.insync.replicas 配置的是 2，比如 Leader 副本和 Follower 1 副本收到消息，此时 Leader 副本挂掉了，就返回异常给 Producer 了，但 Follower 2 副本选举成 Leader 副本了，Follower 2 副本落后 Leader 副本和 Follower 1 副本一些消息，虽然都是在同一个 ISR 列表中，但 Follower 2 副本没

有落后超过 replica.lag.time.max.ms 配置的值（默认是 10s）就还算是同步副本，Follower 2 副本当选 Leader 副本，前面的消息岂不是丢失了?

A5： 社区在 0.11 版本正式引入了 Leader Epoch 概念，来规避因高水位更新错配导致的各种不一致问题。

- Epoch。一个单调增加的版本号。每当副本领导权发生变更时，都会增加该版本号。小版本号的 Leader 被认为是过期 Leader，不能再行使 Leader 权力。
- 起始位移（Start Offset）。Leader 副本在该 Epoch 值上写入的首条消息的位移。

举个例子来说明一下 Leader Epoch。假设现在有两个 Leader Epoch<0, 0> 和 <1, 120>，那么，第一个 Leader Epoch 表示版本号是 0，这个版本的 Leader 从位移 0 开始保存消息，一共保存了 120 条消息。之后，Leader 发生了变更，版本号增加到 1，新版本的起始位移是 120。

这样就可以保证 Follower 2 副本选举成 Leader 副本时，沿用之前的旧的 Leader 副本的最后的消息位移。