22-Kafka和RocketMQ的消息复制实现的差异点在哪？

你好，我是李玥。

之前我在《05 | 如何确保消息不会丢失？》那节课中讲过，消息队列在收发两端，主要是依靠业务代码，配合请求确认的机制，来保证消息不会丢失的。而在服务端，一般采用持久化和复制的方式来保证不丢消息。

把消息复制到多个节点上，不仅可以解决丢消息的问题，还可以保证消息服务的高可用。即使某一个节点宕机了，还可以继续使用其他节点来收发消息。所以大部分生产系统，都会把消息队列配置成集群模式，并开启消息复制，来保证系统的高可用和数据可靠性。

这节课我们来讲一下，消息复制需要解决的一些问题，以及RocketMQ和Kafka都是如何应对这些问题来实现复制的。

消息复制面临什么问题？

我们希望消息队列最好能兼具高性能、高可用并且还能提供数据一致性的保证。虽然很多消息队列产品宣称三个特性全都支持，但你需要知道，这都是有前置条件的。

首先来说性能。任何的复制实现方式，数据的写入性能一定是不如单节点的。这个很好理解，因为无论采用哪种复制实现方式，都需要数据被写入到多个节点之后再返回，性能一定是不如只写入一个节点的。

需要写入的节点数量越多，可用性和数据可靠性就越好，但是写入性能就越低，这是一个天然的矛盾。不过，复制对消费的性能影响不大，不管采用哪种复制方式，消费消息的时候，都只是选择多副本中一个节点去读数据而已，这和单节点消费并没有差别。

再来说一致性，消息队列对数据一致性的要求，既包括了“不丢消息”这个要求，也包括“严格顺序”的要求。如果要确保数据一致性，必须采用“主-从”的复制方式，这个结论是有严格的数学论证的，大家只要记住就可以了。

在“主-从”模式下，数据先写入到主节点上，从节点只从主节点上复制数据，如果出现主从数据不一致的情况，必须以主节点上的数据为准。这里面需要注意一下，这里面的主节点它并不是不可变的，在很多的复制实现中，当主节点出现问题的时候，其他节点可以通过选举的方式，变成主节点。只要保证，在任何一个时刻，集群的主节点数不能超过1个，就可以确保数据一致性。

最后说一下高可用。既然必须要采用主从的复制方式，高可用需要解决的就是，当某个主节点宕机的时候，尽快再选出一个主节点来接替宕机的主节点。

比较快速的实现方式是，使用一个第三方的管理服务来管理这些节点，发现某个主节点宕机的时候，由管理服务来指定一个新的主节点。但引入管理服务会带来一系列问题，比如管理服务本身的高可用、数据一致性如何保证？

有的消息队列选择自选举的方式，由还存活的这些节点通过投票，来选出一个新的主节点，这种投票的实现方式，它的优点是没有外部依赖，可以实现自我管理。缺点就是投票的实现都比较复杂，并且选举的过程是比较慢的，几秒至几十秒都有可能，在选出新的主节点前，服务一直是不可用的。

大部分复制的实现，都不会选择把消息写入全部副本再返回确认，因为这样虽然可以保证数据一致性，但

是，一旦这些副本中有任何一个副本宕机，写入就会卡死了。如果只把消息写入到一部分副本就认为写入成功并返回确认，就可以解决卡死的问题，并且性能也会比写全部副本好很多。

到底写入多少个副本算写入成功呢？这又是一个非常难抉择的问题。

假设我们的集群采用“一主二从三副本”的模式，如果只要消息写入到两个副本就算是写入成功了，那这三个节点最多允许宕机一个节点，否则就没法提供服务了。如果说我们把要求写入的副本数量降到1，只要消息写入到主节点就算成功了，那三个节点中，可以允许宕机两个节点，系统依然可以提供服务，这个可用性就更好一些。但是，有可能出现一种情况：主节点有一部分消息还没来得复制到任何一个从节点上，主节点就宕机了，这时候就会丢消息，数据一致性又没有办法保证了。

以上我讲的这些内容，还没有涉及到任何复制或者选举的方法和算法，都是最朴素，最基本的原理。你可以看出，这里面是有很多天然的矛盾，所以，**目前并没有一种完美的实现方案能够兼顾高性能、高可用和一致性。**

不同的消息队列选择了不同的复制实现方式，这些实现方式都有各自的优缺点，在高性能、高可用和一致性方面提供的能力也是各有高低。接下来我们一起来看一下RocketMQ和Kafka分别是如何来实现复制的。

RocketMQ如何实现复制？

RocketMQ在2018年底迎来了一次重大的更新，引入Deldger，增加了一种全新的复制方式。我们先来说一下传统的复制方式。

在RocketMQ中，复制的基本单位是Broker，也就是服务端的进程。复制采用的也是主从方式，通常情况下配置成一主一从，也可以支持一主多从。

RocketMQ提供了两种复制方式，一种是异步复制，消息先发送到主节点上，就返回“写入成功”，然后消息再异步复制到从节点上。另外一种方式是同步双写，消息同步双写到主从节点上，主从都写成功，才返回“写入成功”。这两种方式本质上的区别是，写入多少个副本再返回“写入成功”的问题，异步复制需要的副本数是1，同步双写需要的副本数是2。

我刚刚讲过，如果在返回“写入成功”前，需要写入的副本数不够多，那就会丢消息。对RocketMQ来说，如果采用异步复制的方式会不会丢消息呢？答案是，并不会丢消息。

我来跟你说一下为什么不会丢消息。

在RocketMQ中，Broker的主从关系是通过配置固定的，不支持动态切换。如果主节点宕机，生产者就不能再生产消息了，消费者可以自动切换到从节点继续进行消费。这时候，即使有一些消息没有来得及复制到从节点上，这些消息依然躺在主节点的磁盘上，除非是主节点的磁盘坏了，否则等主节点重新恢复服务的时候，这些消息依然可以继续复制到从节点上，也可以继续消费，不会丢消息，消息的顺序也是没有问题的。

从设计上来讲，**RocketMQ的这种主从复制方式，牺牲了可用性，换取了比较好的性能和数据一致性。**

那RocketMQ又是如何解决可用性的问题的呢？一对儿主从节点可用性不行，多来几对儿主从节点不就解决了？RocketMQ支持把一个主题分布到多对主从节点上去，每对主从节点中承担主题中的一部分队列，如果某个主节点宕机了，会自动切换到其他主节点上继续发消息，这样既解决了可用性的问题，还可以通过水平扩容来提升Topic总体的性能。

这种复制方式在大多数场景下都可以很好的工作，但也面临一些问题。

比如，在需要保证消息严格顺序的场景下，由于在主题层面无法保证严格顺序，所以必须指定队列来发送消息，对于任何一个队列，它一定是落在一组特定的主从节点上，如果这个主节点宕机，其他的主节点是无法替代这个主节点的，否则就无法保证严格顺序。在这种复制模式下，严格顺序和高可用只能选择一个。

RocketMQ引入Dledger，使用新的复制方式，可以很好地解决这个问题。我们来看一下Dledger是怎么来复制的。

Dledger在写入消息的时候，要求至少消息复制到半数以上的节点之后，才给客户端返回写入成功，并且它是支持通过选举来动态切换主节点的。

同样拿3个节点举例说明一下。当主节点宕机的时候，2个从节点会通过投票选出一个新的主节点来继续提供服务，相比主从的复制模式，解决了可用性的问题。由于消息要至少复制到2个节点上才会返回写入成功，即使主节点宕机了，也至少有一个节点上的消息是和主节点一样的。Dledger在选举时，总会把数据和主节点一样的从节点选为新的主节点，这样就保证了数据的一致性，既不会丢消息，还可以保证严格顺序。

当然，Dledger的复制方式也不是完美的，依然存在一些不足：比如，选举过程中不能提供服务。最少需要3个节点才能保证数据一致性，3节点时，只能保证1个节点宕机时可用，如果2个节点同时宕机，即使还有1个节点存活也无法提供服务，资源的利用率比较低。另外，由于至少要复制到半数以上的节点才返回写入成功，性能上也不如主从异步复制的方式快。

讲完了RocketMQ，我们再来看看Kafka是怎么来实现复制的。

Kafka是如何实现复制的?

Kafka中，复制的基本单位是分区。每个分区的几个副本之间，构成一个小的复制集群，Broker只是这些分区副本的容器，所以Kafka的Broker是不分主从的。

分区的多个副本中也是采用一主多从的方式。Kafka在写入消息的时候，采用的也是异步复制的方式。消息在写入到主节点之后，并不会马上返回写入成功，而是等待足够多的节点都复制成功后再返回。在Kafka中这个“足够多”是多少呢？Kafka的设计哲学是，让用户自己来决定。

Kafka为这个“足够多”创造了一个专有名词：ISR（In Sync Replicas)，翻译过来就是“保持数据同步的副本”。ISR的数量是可配的，但需要注意的是，这个ISR中是包含主节点的。

Kafka使用ZooKeeper来监控每个分区的多个节点，如果发现某个分区的主节点宕机了，Kafka会利用ZooKeeper来选出一个新的主节点，这样解决了可用性的问题。ZooKeeper是一个分布式协调服务，后面，我会专门用一节课来介绍ZooKeeper。选举的时候，会从所有ISR节点中来选新的主节点，这样可以保证数据一致性。

默认情况下，如果所有的ISR节点都宕机了，分区就无法提供服务了。你也可以选择配置成让分区继续提供服务，这样只要有一个节点还活着，就可以提供服务，代价是无法保证数据一致性，会丢消息。

Kafka的这种高度可配置的复制方式，优点是非常灵活，你可以通过配置这些复制参数，在可用性、性能和一致性这几方面做灵活的取舍，缺点就是学习成本比较高。

总结

这节课我们主要来讲了一下，消息复制需要面临的问题以及RocketMQ和Kafka都是如何应对这些问题来实现复制的。

RocketMQ提供新、老两种复制方式：传统的主从模式和新的基于Dledger的复制方式。传统的主从模式性能更好，但灵活性和可用性稍差，而基于Dledger的复制方式，在Broker故障的时候可以自动选举出新节点，可用性更好，性能稍差，并且资源利用率更低一些。Kafka提供了基于ISR的更加灵活可配置的复制方式，用户可以自行配置，在可用性、性能和一致性这几方面根据系统的情况来做取舍。但是，这种灵活的配置方式学习成本较高。

并没有一种完美的复制方案，可以同时能够兼顾高性能、高可用和一致性。你需要根据你实际的业务需求，先做出取舍，然后再去配置消息队列的复制方式。

思考题

假设我们有一个5节点的RocketMQ集群，采用Dledger5副本的复制方式，集群中只有一个主题，50个队列均匀地分布到5个Broker上。

如果需要你来配置一套Kafka集群，要求达到和这个RocketMQ集群一样的性能（不考虑Kafka和RocketMQ本身的性能差异）、可用性和数据一致性，该如何配置？欢迎在留言区与我分享讨论。

感谢阅读，如果你觉得这篇文章对你有帮助，也欢迎把它分享给你的朋友。

精选留言：

- lmtoo 2019-09-14 11:49:30
  1.kafka集群配置broker数量为5
  2.创建主题的时候，指定分区数量为50、分区副本数为5
  3.每个分区的ISR数量为3