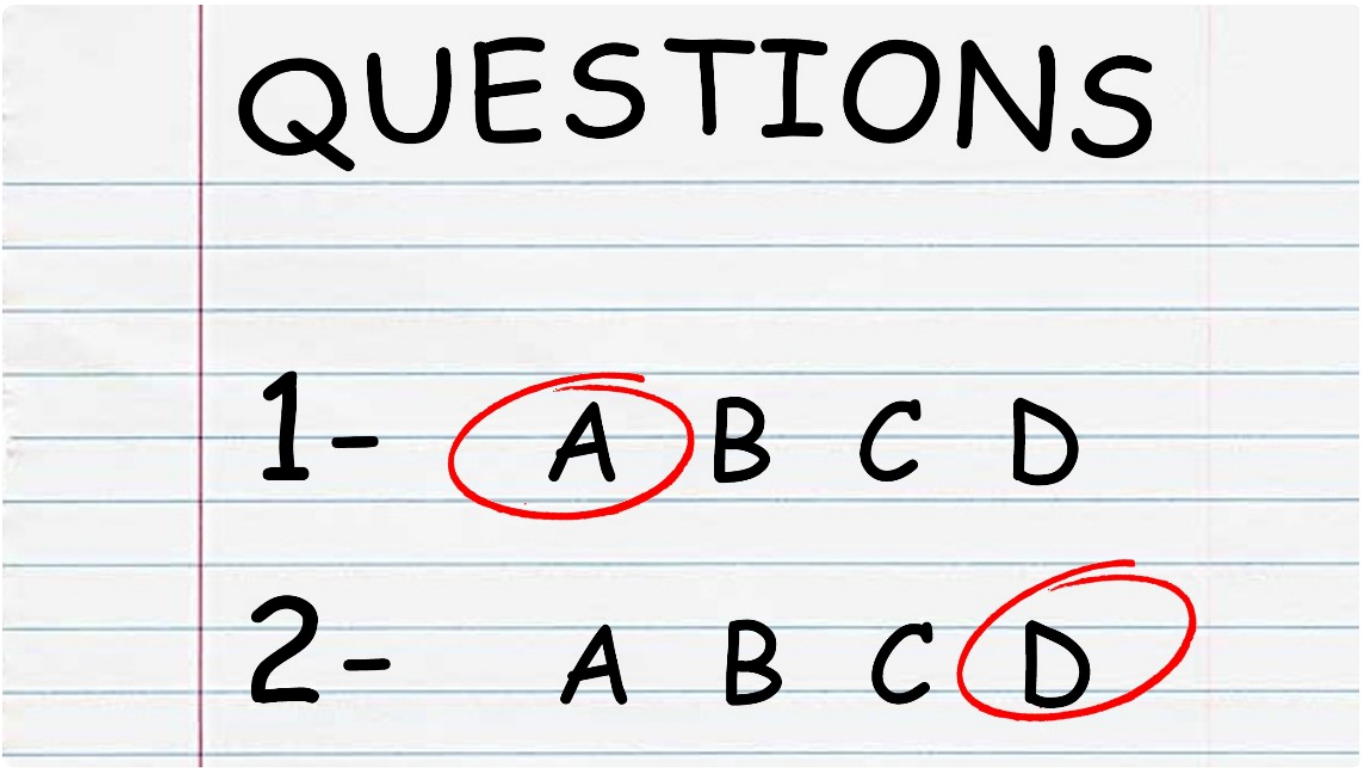
14 | 答疑篇：分布式事务与分布式锁相关问题

2019-10-23 聂鹏程



讲述：聂鹏程
时长 12:55 大小 11.84M

你好，我是聂鹏程。

到目前为止，“分布式技术原理与算法解析”专栏已经更新 13 篇文章了，主要与你介绍了“分布式起源”“分布式协调与同步”和“分布式资源管理与负载调度”。

在这里，我首先要感谢你们在评论区留下的一条条精彩留言。这让我感觉到，你们对分布式技术的浓厚兴趣，以及对我和对这个专栏的支持。这不但活跃了整个专栏的氛围、丰富了专栏的内容，也让我备受鼓舞，干劲儿十足地去交付更高质量的文章。

比如，@xfan、@Jackey 等同学积极思考问题，并对文中有疑惑的地方结合其他资料给出了自己的思考和建议；再比如，@zhaozp 、@静水流深等同学，每次更新后都在坚持打卡

学习；再比如，@约书亚等同学针对分布式事务提出了非常好的问题，@每天晒白牙等同学对文中内容进行了非常好的总结。

这些同学有很多，我就不再一一点名了。感谢你们的同时，我也相信，积极参与并留言也会帮助你更深入地理解每一个知识点。所以我希望，你接下来可以多多留言给我，让我们一起学习，共同进步。

留言涉及的问题有很多，但我经过进一步地分析和总结后，发现针对分布式事务和分布式锁的问题比较多，同学们的疑惑也比较多。

确实，这两大问题属于分布式技术的关键问题。因此，今天的这篇答疑文章，我就围绕这两大问题来进行一次集中的分析和回答吧。

我们先来看一下分布式事务的相关问题。

分布式事务的相关问题

在第 6 篇文章“分布式事务：All or Nothing”中，我介绍了两阶段提交协议和三阶段提交协议。有很多同学提出了疑问：两阶段提交协议（2PC）和三阶段提交协议（3PC）的区别，到底是什么？

在回答这个问题前，我建议你先回到第 6 篇文章，去回忆一下它们的流程。然后，我们看看**2PC 和 3PC 的第一步到底是不是“类似”的？**

2PC 的第一步投票（voting）阶段中，参与者收到事务执行询问请求时，就执行事务但不提交；而 3PC 却写着在 PreCommit 阶段执行事务不提交。2PC 和 3PC 的第一步，是非常不类似吧？

其实，我说它们类似，是指它们均是通过协调者，来询问参与者是否可以正常执行事务操作，参与者也都会给协调者回复。

> 在 2PC 中，如果所有参与者都返回结果后，会进入第二阶段，也就是提交阶段，也可以说是执行阶段，根据第一阶段的投票结果，进行提交或取消。

在 3PC 中，进入真正的提交阶段前，还会有一个预提交阶段，这个预提交阶段不会做真正的提交，而是会将相关信息记录到事务日志中，当所有参与者都返回 Yes 消息后，才会真正进入提交阶段。

这样说明后，相信你对这个问题的疑惑应该解决了吧。

现在，我们继续延展一下这个问题吧。

追问 1：3PC 在预提交阶段，才开始执行事务操作，那协调者发送 CanCommit 给参与者的时候，参与者根据什么返回 Yes 或者 No 消息呢?

3PC 在投票阶段（CanCommit 阶段），协调者发送 CanCommit 询问后，参与者会根据自身情况，比如自身空闲资源是否足以支撑事务、是否会存在故障等，预估自己是否可以执行事务，但不会执行事务，参与者根据预估结果给协调者反回 Yes 或者 No 消息。

追问 2：3PC 出现的目的是，解决 2PC 的同步阻塞和数据不一致性问题。那么，我们不可以在 2PC 中直接去解决这些问题吗？3PC 多了预提交和超时机制，就真的解决这些问题了吗?

我们先来看看同步阻塞的问题。

在 2PC 中，参与者必须等待协调者发送的事务操作指令，才会执行事务，比如提交事务或回滚等操作，如果协调者故障，也就是说参与者无法收到协调者的指令了，那么参与者只能一直等待下去。这就好比在一个班级里面，班主任是协调者，学生是参与者，班主任告诉学生今天下午 6 点组织一个比赛，但班主任今天生病了，根本到不了学校，并且也无法发送信息告诉学生，那么学生们就只能一直等待。

3PC 在协调者和参与者中都引入了超时机制（2PC 只是在协调者引入了超时），也就是说当参与者在一定时间内，没有接收到协调者的通知时，会执行默认的操作，从而减少了整个集群的阻塞时间。这就好比班主任生病了，学生默认等待半个小时，如果班主任还没有任何通知，那么默认比赛取消，学生可以自由安排，做自己的事情去了。

但其实，阻塞在实际业务中是不可能完全避免的。在上面的例子中，学生等待超时的半个小时中，其实还是阻塞的，只是阻塞的时间缩短了。所以，相对于 2PC 来说，3PC 只是在一

定程度上减少（或者说减弱）了阻塞问题。

接下来，我们再看看数据不一致的问题吧。

通过上面的分析可以看到，同步阻塞的根本原因是协调者发生故障，想象一下，比如现在有 10 个参与者，协调者在发送事务操作信息的时候，假设在发送给了 5 个参与者之后发生了故障。在这种情况下，未收到信息的 5 个参与者会发生阻塞，收到信息的 5 个参与者会执行事务，以至于这 10 个参与者的数据信息不一致。

3PC 中引入了预提交阶段，相对于 2PC 来讲是增加了一个预判断，如果在预判断阶段协调者出现故障，那就不会执行事务。这样，可以在一定程度上减少故障导致的数据不一致问题，尽可能保证在最后提交阶段之前，各参与节点的状态是一致的。

所以说，3PC 是研究者们针对 2PC 中存在的问题做的一个改进，虽然没能完全解决这些问题，但也起到了一定的效果。

在实际使用中，通常采用多数投票策略来代替第一阶段的全票策略，比如 Raft 算法等。关于 Raft 算法的具体原理，你可以再回顾下第 4 篇文章“分布式选举：国不可一日无君”中的相关内容。

追问 3：3PC 也是只有一个协调者，为什么就不会有单点故障问题了？

首先，我先明确下这里所说的单点故障问题。

因为系统中只有一个协调者，那么协调者所在服务器出现故障时，系统肯定是无法正常运行的。所以说，2PC 和 3PC 都会有单点故障问题。

但是，3PC 因为在协调者和参与者中都引入了超时机制，可以减弱单点故障对整个系统造成的影响。为什么这么说呢？

因为引入的超时机制，参与者可以在长时间没有得到协调者响应的情况下，自动将超时的事务进行提交，不会像 2PC 那样被阻塞住。

好了，以上就是关于分布式事务中的 2PC 和 3PC 的相关问题了，相信你对这两个提交协议有了更深刻的认识。接下来，我们再看一下分布式锁的相关问题吧。

分布式锁的相关问题

在第 7 篇文章“分布式锁：关键重地，非请勿入”后的留言中，我看到很多同学都问到了分布式互斥和分布式锁的关系是什么。

我们先来回顾下，分布式互斥和分布式锁分别是什么吧。

在分布式系统中，某些资源（即临界资源）同一时刻只有一个程序能够访问，这种排他性的资源访问方式，就叫作**分布式互斥。**这里，你可以再回顾下第 3 篇文章中的相关内容。

分布式锁指的是，在分布式环境下，系统部署在多个机器中，实现多进程分布式互斥的一种锁。这里，你可以再回顾下第 7 篇文章中的相关内容。

分布式锁的目的是，保证多个进程访问临界资源时，同一时刻只有一个进程可以访问，以保证数据的正确性。因此，我们可以说分布式锁是实现分布式互斥的一种手段或方法。

除了分布式互斥和分布式锁的关系外，很多同学都针对基于 ZooKeeper 和基于 Redis 实现分布式锁，提出了不少好问题。我们具体看看这些问题吧。

首先，我们来看一下基于 ZooKeeper 实现分布式锁的问题。有同学问，ZooKeeper 分布式锁，可能存在多个节点对应的客户端在同一时间完成事务的情况吗？

这里，我需要先澄清一下，ZooKeeper 不是分布式锁，而是一个分布式的、提供分布式应用协调服务的组件。基于 ZooKeeper 的分布式锁是基于 ZooKeeper 的数据结构中的临时顺序节点来实现的。

请注意，这里提到了 ZooKeeper 是一个分布式应用协调服务的组件。比如，在一个集中式集群中，以 Mesos 为例，Mesos 包括 master 节点和 slave 节点，slave 节点启动后是主动去和 master 节点建立连接的，但建立连接的条件是，需要知道 master 节点的 IP 地址和状态等。而 master 节点启动后会将自己的 IP 地址和状态等写入 ZooKeeper 中，这样每个 slave 节点启动后都可以去找 ZooKeeper 获取 master 的信息。而每个 slave 节点与 ZooKeeper 进行交互的时候，均需要一个对应的客户端。

这个例子，说明了存在多个节点对应的客户端与 ZooKeeper 进行交互。同时，由于每个节点之间并未进行通信协商，且它们都是独立自主的，启动时间、与 ZooKeeper 交互的时间、事务完成时间都是独立的，因此存在多个节点对应的客户端在同一时间完成事务的这种情况。

接下来，我们看一下基于 Resis 实现分布式锁的问题。**Redis 为什么需要通过队列来维持进程访问共享资源的先后顺序?**

在我看来，这是一个很好的问题。

@开心小毛认为，Redis 的分布式锁根本没有队列，收到 setnx 返回为 0 的进程会不断地重试，直到某一次的重试成为 DEL 命令后第一个到达的 setnx 从而获得锁，至于此进程在等待获得锁的众多进程中是不是第一个发出 setnx 的，Redis 并不关心。

其实，客观地说，这个理解是合情合理的，是我们第一反应所能想到的最直接、最简单的解决方法。可以说，这是一种简单、粗暴的方法，也就是获取不到锁，就不停尝试，直到获取到锁为止。

但你有没有想过，当多个进程频繁去访问 Redis 时，Redis 会不会成为瓶颈，性能会不会受影响。带着这个疑惑，我们来具体看看基于 Redis 实现的分布式锁到底需不需要队列吧。

如果没有队列维护多进程请求，那我们可以想到的解决方式，就是我刚刚和你分析过的，通过多进程反复尝试以获取锁。

但，这种方式有三个问题：

一是，反复尝试会增加通信成本和性能开销；

二是，到底过多久再重新尝试；

三是，如果每次都是众多进程进行竞争的话，有可能会导致有些进程永远获取不到锁。

在实际的业务场景中，尝试时间的设置，是一个比较难的问题，与节点规模、事务类型均有关系。

比如，节点规模大的情况下，如果设置的时间周期较短，多个节点频繁访问 Redis，会给 Redis 带来性能冲击，甚至导致 Redis 崩溃；对于节点规模小、事务执行时间短的情况，若设置的重试时间周期过长，会导致节点执行事务的整体时间变长。

基于队列来维持进程访问共享资源先后顺序的方法中，当一个进程释放锁之后，队列里第一个进程可以访问共享资源。也就说，这样一来就解决了上面提到的三个问题。

总结

我针对前面 13 篇文章留言涉及的问题，进行了归纳总结，从中摘取了分布式事务和分布式锁这两个知识点，串成了今天这篇答疑文章。

今天没来得及和你扩展的问题，后续我会再找机会进行解答。最后，我要和你说的是，和我一起打卡分布式核心技术，一起遇见更优秀的自己吧。

篇幅所限，留言区见。

我是聂鹏程，感谢你的收听，欢迎你在评论区给我留言分享你的观点，也欢迎你把这篇文章分享给更多的朋友一起阅读。我们下期再会！

上一篇　13 | 分布式调度架构之共享状态调度：物质文明、精神文明多手协商抓

下一篇　15 | 分布式计算模式之MR：一门同流合污的艺术

精选留言 (7)

写留言

xfan
2019-10-26

要动手实际将现有的系统改造为分布式系统，才能明白很多细节

leslie
2019-10-23

其实单独去提redis我觉得可能有点片面：不同的中间件干不同的事情；单独去用单一数据库个人认为有点窄了，这回造成；分布式就是传统CS或BS的简单升级。我们在谈论它的优点时忘了去忽略了它只是系统中的一部分而已。

　　老师的文章跟到现在：由于在极客时间有其它学习压力非常大的课程，基本上有些东西仅仅只能暂时保持学习了。不过关于分布式调度其实个人觉得是这块的核心：调度的核...

展开

Jackey
2019-10-23

感谢老师答疑，打算回去再刷一遍分布式事务和分布式锁😃

随心而至
2019-10-23

我觉得关于redis实现分布式锁的那段，可以类比着Java里面原子类和AQS来理解，一个是不断尝试CAS直到成功，一个是多线程访问共享锁，拿不到就放到链表中排队。

随心而至
2019-10-23

关于redis 实现分布式锁，我觉得可以类比Java里面原子类和AQS，一个不断尝试cas直到成功，一个多线程尝试获取共享锁，拿不到就排队。

展开

1

Hurt

2019-10-23

打卡 打卡

展开

忆水寒

2019-10-23

收获很多

展开