mp.weixin.qq.com

# 分布式系统一致性发展史（二）丨两阶段和三阶段提交

> 导语：
>
> 上一部分我们提到了consensus问题，今天这

> 篇文章我们介绍一下1975到1981年，科学家提出的解决consensus问题的方法——2pc和3pc，即两阶段提交和三阶段提交。

**两阶段提交(two phase commit)**

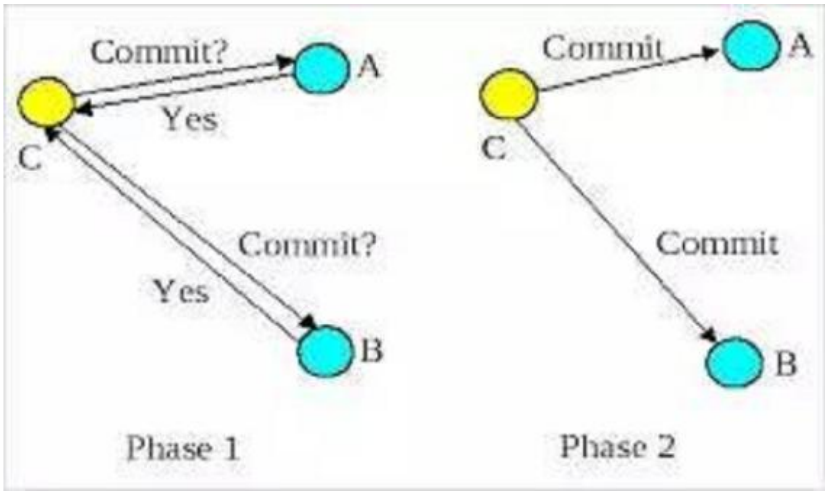
两阶段提交(two phase commit) 或者又简称为2PC是解决consensus问题的一个方法. 在Paxos算

法出现之前, Jim Gary在在Leslie Lamport那篇开山之作的同年1978年提出了two phase commit的概念, 图灵奖获得者Jim Gary是分布式事务的专家, 他所设计的2PC的主要用途是实现分布式事务, 让两个数据库或者队列参与同一个分布式事务.

这个过程中有Coordinator

负责协调各个资源参与者(cohort)去提交或者回滚自己的事务. 首先coordinator会通知所有cohort告诉他们要提交的内容, 如果cohort写入成功(这时还没有提交), 那么cohort返回同意的应答给coordinator. 然后coordinator收集到全部的应答之后如果有任何一个应答是拒绝的(某个cohort写入失败), 那么

coordinator就通知所有cohort回滚, 否则通知所有cohort提交.




图片来源: Jboss

2PC最早的实际应用可能是八十年代的Tuxedo, 后

来演化为XA/Open规范,
之后大多数商业数据库都
开始支持XA/Open规范.
但是2PC的局限性很多人
不太了解, 认为2PC是解决
分布式事务的银弹。

其实2PC有一个很大的**局限性**: 2PC是一个阻塞协议. 假设coordinator给cohort A和B发出了事务内容, A和B都成功写入并返

回同意的应答, 这时候coordinator和A都挂了, 然后B此时是无法得知现在事务已经被决定提交了还是要决定回滚了, B什么也不能干, 只能继续傻等, 直到coordinator恢复, 这时候就算重新启动一个新的coordinator也无法得知刚才的事务到底是什么状态了. 除非A也被恢复. 尽管事务阻塞了, 但是至此数据

还是安全的，只是事务在一个中间状态暂停了，B这个时候可以阻塞本地事务，并锁住这条数据禁止访问.

尽管这样数据是安全的，但是由于实际应用中事务阻塞是不可接受的，如果cohort一直锁定着资源，这样可能会导致整个系统不可用，所以大多数实现都会超时回滚。但是这种情况

下超时回滚这个事务就变成了heuristic transaction, 这时候A和B的数据就不一致了, 后面需要人工介入去改数据或者通过补偿去修复. 当你使用JTA的时候遇到个HeuristicMixedExceptionn, 那么很可能是这种情况发生了.

虽然2PC并不能像很多人

想象的那样保证事务的一致性, 不考虑超时回滚的情况下它是安全的, 2PC可以保证safety但是不保证liveness. 为了解决这个阻塞问题, 后来又出现了3PC. 2PC的阻塞主要原因是当coordinator和cohorts同时crash的时候, 之前cohorts之间没有沟通过表决的结果, 它们只和coordinator表决过, 表决结

果没有保留下来, 就算立即重新启动一个新的coordinator也无法判断刚才事务的状态, 所以他们会进入群龙无首, 进退两难的情况, 如果有一轮协商的过程, 那么即便coordinator挂了, 也可以再启动一个coordinator去询问cohorts上一轮协商的结果并把事务继续下去, 那就能解决阻塞问题了.

## 3PC

三年后出现的3PC [Nonblocking Commit Protocols, Dale Skeen, 1981] 就是在2PC两个阶段之间插入一个阶段增加了一个相互协商的过程, 并引入了超时来防止阻塞。

这个中间阶段让coordinator发现全体写入资源并收到ACK之后, 先

发一个prepare commit消息到全体cohorts, 当cohort全体都同意并返回ACK给coordinator之后, coordinator才发commit消息出去让cohorts提交.

如果prepare commit发出给A之后, coordinator和A都挂了, 如果立刻重新启动一个新的coordinator, 那么它发现B没有收到过

prepare commit, 这个coordinator就可以发消息给所有cohorts去取消提交. B会回滚, 当A恢复回来之后可以去问coordinator或者任何一个cohort都会知道事务已经回滚. 这样整个事务就回滚了, 因为表决结果通过prepare commit消息可以保留在所有cohorts节点上, 这个情况是2PC无法解决的.

对于引入的中间阶段, 本身也是安全的. 比如A和B都写入资源之后, 如果coordinator没能发出prepare commit就挂了, 那么A/B会超时而回滚事务, 这是安全的. 如果coordinator 发出了prepare commit给A, 还没能给B发出, coordinator和A都挂了, 那么B也会超时

回滚. 如果coordinator发出了prepare commit给A和B然后coordinator挂了, 这时候如果不启动任何coordinator那么A/B都会超时提交, 如果能再启动一个coordinator那么这个coordinator会发现所有节点都收到了prepare commit消息, 这个coordinator也会让所有节点提交.

图片来源: wikipedia

**3PC局限性：**实际上, 3PC没有什么实际应用, 高延迟是个很大的局限, 而且它还有一个更严重的问题, 那就是在网络分区的情况下也会出现事务不一致问题.

解释3PC局限性原因前先

## 介绍Two Generals Paradox问题.

Two Generals Paradox在1975年被提出[Some Constraints and Trade-offs in the Design of Network Communications], 但是广为人知还是靠分布式事务的专家Jim Gray在1978年的一篇文章[Notes on

Database Operating Systems]中再次提及这个问题(估计是因为Jim Gray名气太大). 问题描述的是有两个将军A和B分别处于敌军的东侧和西侧, 他们决定互相派信使, 好商量一个时间来同时发起攻击, 如果这个时间没有商量好, 一方先攻击了, 那么就会战败. 但是问题来了, 假设将军里没有叛徒(没有故障节点发

出faulty message), 信使如果被中间的敌人抓住了, 会被直接被处死(会有丢包, 但是不会被篡改消息), 那么这两个将军能达成一致么?

好了, 看出来了吧, 这就是一个最简单的consensus问题. 这其实是一个弱化版本的拜占庭将军问题.

如果A送了一个信使m1去

B商量一个进攻时间, B收到之后必须要这个信使回去告诉A自己收到消息了, 这是一个ACK应答消息, 没有ACK显然双方是无法达成确认一致的. 可是如果信使m1回去路上被抓住了, 被杀了. A这个时候等了半天没人回来, 他就会进入两难的境地, 到底是信使m1是去的路上被杀了, 还是回来的路上被杀了呢?

前者是B没有收到消息, A就不能在指定时间发起进攻, 后者是B已经收到了消息, A就必须要进攻. 实际上, 就算A收到了信使m1带回的B的ACK消息, 虽然A放心了, 但是B也不能放心. 因为B送走m1之后, B并不知道他的ACK有没有回到A那里, 如果回到A那里了, B认为自己可以发动攻击, 但是万一信使m1回

去路上被杀了, A没有收到ACK, B岂不是会自己贸然发动进攻了? 怎么办?

有人说了, 让B交代信使m1, 你回去如果见到A了, 让他再送个信使m2过来告诉我一声, 我才放心. 信使m1回去了, 见到了A, A听了之后想了想, 说有道理, 然后又派了m2去B那里告诉B他收到B的ACK了, 读

者看出来没, 这其实是第二个ACK, 这个ACK同样存在不确定因素, 和第一个m1附带的消息所面临的问题是一样的, 按照这个思路两个将军将会进入一个无限循环, 再去发ACK3, ACK4, 但是却无法解决问题. 根源就在于信使其实是一个异步网络, 而且会发生网络分区, 会有延迟和丢包.

这个问题告诉我们在一个异步不可靠的网络内想用简单的消息应答方式解决consensus问题是不可能的.

现在再解释**3PC在网络分区下的不一致原因**就容易了。假设coordinator第一轮发出了事务请求给所有cohorts, 结果所有cohort都锁定资源并写入成功, 而

且都返回了同意应答. 第二轮的时候coordinator给所有cohorts都发出了prepare commit, 并收到了所有的ACK, 到了第三轮, 如果碰巧发生了网络分区, coordinator被隔离开, 无法和任何cohorts通讯, 超时之后, coordinator还没法把commit发出去, 它会认为cohort写入失败或者挂了, coordinator只能发出

rollback请求给所有cohorts, 与此同时, 网络分区的另外一边cohorts那边发现coordinator联系不上了, 不给我们发commit了, 也超时了, 他们会选一个新的coordinator, 这个新的coordinator询问了所有节点发现都已经写入了并且表决同意过了, 那么这个新coordinator会发出commit.

碰巧这时候网络分区恢复了, 老的coordinator发出的rollback和新的coordinator发出的commit将会交错混杂在一起通过网络发给所有的cohorts. 结果将是一片混乱和不确定. 3PC虽然是非阻塞的, 但是他的超高延迟和缺乏网络分区的忍耐性让它的实际应用大打折扣.

## 2PC和3PC的功过总结

它们的正确性都是假设分布式系统中网络是稳定的, 延迟稳定, 带宽无限 [1994, Peter Deutsch] 而当网络分区的时候, 节点的故障判断非常困难. 2PC和3PC假设节点的故障判断非常容易, 就是超时, 并且假设节点故障之后不会自动恢复. 这种假设叫做crash-stop, 意思就是说我

看不到你就认为你挂了，有点像王明阳的”你未看此花时，此花与汝心同归于寂”，在分布式系统中这种故障模型的假设是比较窄的. 你必须要假设你看不到他的时候，他也可能活着，说不定过了几秒钟他可能又给你发消息和你交互了. 在大多数情况下，我们至少要把故障模型放大到crash-recovery。

**点击查看相关精彩文章：**

[分布式系统发展史第一篇]

[分布式系统发展史第二篇第一部分]

**今天是分布式系统发展史第二篇的第二部分内容，明天我们将继续发布第三部分内容《拜占庭将军问题和FLP的启示》，敬请**

**期待。**

**如果您想投稿给我们，或者想转发和采用我们的稿件，请回复“合作”，小编会在2小时内回复您的投稿和合作需求。**

---

**本文作者：**Daniel，吴强， 现任点融网首席社交平台架构师，前盛大架构

师, 专注分布式系统和移动应用, 有十三年的开发经验, 目前在点融做最爱的两件事情: 写代码和重构代码。

---

**随着新一轮融资，点融网开始了大规模的扩张，需要各种优秀人才的加入，如果您觉得自己够优秀，欢迎加入我们！**