# 分布式一致性

分布式一致性算法的目的并不是达到了副本数据的强一致性，因为CAP理论告诉我们，这是不可能的，而是在可用性和一致性的均衡处理上，用这些算法可以达到BASE状态，确保我们的分布式系统正确运行，那些数据副本的同步问题并不是一致性算法所要解决的，因为副本的同步不管是Mysql主从同步，还是其他master-slave集群的同步都是异步的，所以不可能达到强一致性，所以这些一致性算法是用来保证分布式系统的可用--一致的算法。例如，我们可以将paxos算法看做是分布式共识算法，它的目的就是怎么样对某个值（决议）在集群中达成一致，所以这可以用于集群中的leader选举，写数据过程。而2pc，3pc就是解决分布式事务（参与事务的事务管理器，参与者位于不同的节点）的原子性，一致性问题，而且这些算法并不是只能单一的应用于某个问题上，有很多问题都可以转化为用这些算法解决。

## 2PC

二阶段提交协议主要分为来个阶段：准备阶段和提交阶段。

### 例子

牧师：”你愿意娶这个女人吗?爱她、忠诚于她，无论她贫困、患病或者残疾，直至死亡。Doyou(你愿意吗)?”

新郎：”Ido(我愿意)!”

牧师：”你愿意嫁给这个男人吗?爱他、忠诚于他，无论他贫困、患病或者残疾，直至死亡。Doyou(你愿意吗)?”

新娘：”Ido(我愿意)!”

牧师：现在请你们面向对方，握住对方的双手，作为妻子和丈夫向对方宣告誓言。

新郎：我——某某某，全心全意娶你做我的妻子，无论是顺境或逆境，富裕或贫穷，健康或疾病，快乐或忧愁，我都将毫无保留地爱你，我将努力去理解你，完完全全信任你。我们将成为一个整体，互为彼此的一部分，我们将一起面对人生的一切，去分享我们的梦想，作为平等的忠实伴侣，度过今后的一生。

新娘：我全心全意嫁给你作为你的妻子，无论是顺境或逆境，富裕或贫穷，健康或疾病，快乐或忧愁，我都将毫无保留的爱你，我将努力去理解你，完完全全信任你，我们将成为一个整体，互为彼此的一部分，我们将一起面对人生的一切，去分享我们的梦想，作为平等的忠实伴侣，度过今后的一生。

首先协调者（牧师）会询问两个参与者（二位新人）是否能执行事务提交操作（愿意结婚）。如果两个参与者能够执行事务的提交，先执行事务操作，然后返回YES，如果没有成功执行事务操作，就返回NO。

当协调者接收到所有的参与者的反馈之后，开始进入事务提交阶段。如果所有参与者都返回YES，那就发送COMMIT请求，如果有一个人返回NO，那就返送roolback请求。

值得注意的是，二阶段提交协议的第一阶段准备阶段不仅仅是回答YES or NO，还是要执行事务操作的，只是执行完事务操作，并没有进行commit还是roolback。

### 存在的问题

**同步阻塞问题**。执行过程中，所有参与节点都是事务阻塞型的。当参与者占有公共资源时，其他第三方节点访问公共资源不得不处于阻塞状态，**各个参与者在等待协调者发出提交或中断请求时，会一直阻塞，而协调者的发出时间要依赖于所有参与者的响应时间，如果协调者宕机了（单点），那么他就一直阻塞在这，而且无法达成一致（3PC引入了超时提交解决）。**

**单点故障**。由于协调者的重要性，一旦协调者发生故障。参与者会一直阻塞下去。尤其在第二阶段**，协调者发生故障，那么所有的参与者还都处于锁定事务资源的状态中，而无法继续完成事务操作**。

**数据不一致**

第二阶段协调者和参与者挂了，挂了的这个参与者在挂之前已经执行了操作。但是由于他挂了，没有人知道他执行了什么操作。

这种情况下，新的协调者被选出来之后，如果他想负起协调者的责任的话他就只能按照之前那种情况来执行commit或者roolback操作。这样新的协调者和所有没挂掉的参与者就保持了数据的一致性，我们假定他们执行了commit。但是，这个时候，那个挂掉的参与者恢复了怎么办，因为他之前已经执行完了之前的事务，如果他执行的是commit那还好，和其他的机器保持一致了，万一他执行的是roolback操作那？这不就导致数据的不一致性了么？虽然这个时候可以再通过手段让他和协调者通信，再想办法把数据搞成一致的，但是，这段时间内他的数据状态已经是不一致的了。

## 3PC

三阶段提交有CanCommit、PreCommit、DoCommit三个阶段。在第一阶段，只是询问所有参与者是否可可以执行事务操作，并不在本阶段执行事务操作。当协调者收到所有的参与者都返回YES时，在第二阶段才执行事务操作，然后在第三阶段在执行commit或者rollback。

### 例子

班长要组织全班同学聚餐，由于大家毕业多年，所以要逐个打电话敲定时间，时间初定10.1日。然后开始逐个打电话。

班长：小A，我们想定在10.1号聚会，你有时间嘛？有时间你就说YES，没有你就说NO，然后我还会再去问其他人，具体时间地点我会再通知你，这段时间你可先去干你自己的事儿，不用一直等着我。（**协调者询问事务是否可以执行，这一步不会锁定资源**）

小A：好的，我有时间。（**参与者反馈**）

班长：小B，我们想定在10.1号聚会……不用一直等我。

班长收集完大家的时间情况了，一看大家都有时间，那么就再次通知大家。（**协调者接收到所有YES指令**）

班长：小A，我们确定了10.1号聚餐，你要把这一天的时间空出来，这一天你不能再安排其他的事儿了。然后我会逐个通知其他同学，通知完之后我会再来和你确认一下，还有啊，如果我没有特意给你打电话，你就10.1号那天来聚餐就行了。对了，你确定能来是吧？（**协调者发送事务执行指令，这一步锁住资源。如果由于网络原因参与者在后面没有收到协调者的命令，他也会执行commit**）

小A顺手在自己的日历上把10.1号这一天圈上了，然后跟班长说，我可以去。（**参与者执行事务操作，反馈状态**）

班长：小B，我们觉得了10.1号聚餐……你就10.1号那天来聚餐就行了。

班长通知完一圈之后。所有同学都跟他说：”我已经把10.1号这天空出来了”。于是，他在10.1号这一天又挨个打了一遍电话告诉他们：嘿，现在你们可以出门拉。。。。（**协调者收到所有参与者的ACK响应，通知所有参与者执行事务的commit**）

小A，小B：我已经出门拉。（**执行commit操作，反馈状态**）

### ****3PC解决的问题：****

相对于2PC，**3PC主要解决的单点故障问题，并减少阻塞**，因为一旦参与者无法及时收到来自协调者的信息之后，他会默认执行commit（所以产生新的协调者之后，可以确定事务的状态，这就解决了单点）。而不会一直持有事务资源并处于阻塞状态。

**3PC无法解决：数据不一致以及太过保守问题。**

由于网络原因，协调者发送的abort响应没有及时被参与者接收到，那么参与者在等待超时之后执行了commit操作。这样就和其他接到abort命令并执行回滚的参与者之间存在数据不一致的情况；除此之外，还有2pc的问题也没有解决。