## 前言

Paxos算法是用来解决分布式系统中，如何就某个值达成一致的算法。

一致性问题

为了实现集群的高可用性，用户的数据往往要多重备份，多个副本虽然避免了单点故障，但同时也引入了新的挑战。

假设有一组服务器保存了用户的余额，初始是100块，现在用户提交了两个订单，一个订单是消费10元，一个订单是充值50元。由于网络错误和延迟等原因，导致一部分服务器只收到了第一个订单（余额更新为90元），一部分服务器只收到了第二个订单（余额更新为150元），还有一部分服务器两个订单都接收到了（余额更新为140元），这三者无法就最终余额达成一致。这就是一致性问题。

一致性算法并不保证所有提出的值都是正确的（这可能是安全管理员的职责）。我们假设所有提交的值都是正确的，算法需要对到底该选哪个做出决策，并使决策的结果被所有参与者获悉。

一致性算法并不保证所有节点中的数据是完全一致的，但它能保证即使一小部分节点出现故障，仍然能对外提供一致的服务（客户端无感知）

在正式开始介绍Paxos所面临的难题前，为了表述方便，先提一下Paxos算法中的三个角色，后面会比较频繁的用到：

Proposer：提案发起者。

Acceptor：决策者，可以批准提案。

Learner：最终决策的学习者。

## Paxos要实现的目标是：

**T1：一次选举，最终一定要选定一个提案（即不能出现所有提案都被拒绝的情况）**

**T2：一次选举，最终只能选定一个提案（即不能出现两个提案有不同的值，却都被选定的情况）**

**T3：当提案被选定后，进程最终能获取到被选定的提案。**

## Paxos算法的推导

要实现paxos算法的目标，最简单的解决方式就是只允许一个acceptor存在，在这种方式下，所有proposer只能发送提案给唯一的那个acceptor，而该acceptor只需要选择它接收到第一个提案作为最终选定的提案即可。

上面说的方式虽然实现起来简单，但是会存在acceptor的单点问题，因此我们不得不寻找其他更好的解决方式，首先我们得使用多个acceptor来避免acceptor的单点问题。实现上来说，在有多个acceptor的情况下，proposer就会向一个acceptor集合发送提案了，我们可以要求每个acceptor最多只能批准（accept）一个提案，并约定当有足够多的（过半）acceptor批准了同一个提案时就意味着该提案被选定了。

根据上面的讨论，约定paxos算法要满足的如下的基本需求：

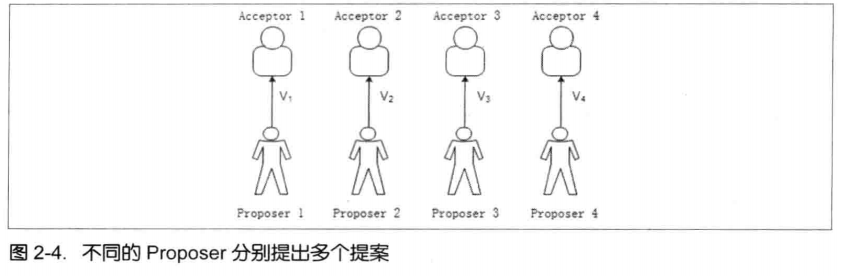
|  |
| --- |
| **B0：采用多个acceptor，每个acceptor最多只能批准一个提案，当过半acceptor批准了同一个提案时就意味着该提案被选定了。** |

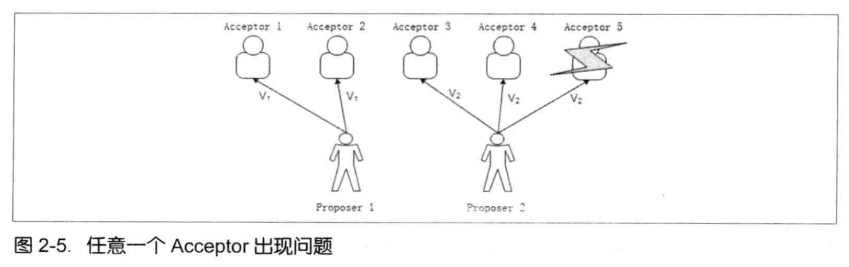
接着分析一种特殊情况，对于一次选举中只有一个提案被一个proposer提出的情况，如果acceptor能够拒绝其收到的第一个提案，就可能导致最终没有过半acceptor批准这个唯一的提案，最终也就没有提案会被选定，就无法实现目标T1。所以paxos算法必须满足的第一个需求P1是：

|  |
| --- |
| **P1：一个Acceptor必须批准它收到的第一个提案。** |

在满足P1的情况下，接着分析。

如图2-4，一次选举中可能有多个提案被同时提出，最终很可能没有一个提案被过半acceptor批准。另外，如图2-5，就算在一次选举中只有两个提案被同时提出，最终也可能出现没有一个提案被过半acceptor批准的情况。





为了解决上面两个问题，需要对基本需求B0做如下修改：

|  |
| --- |
| **B1：采用多个acceptor，对于每个acceptor，不再最多只能批准一个提案，而是改为必须能够批准不止一个提案（即一个acceptor后批准的提案可以覆盖掉其之前批准的提案）”。此时以“[全局编号，Value]”来代表一个提案，其中全局编号用于唯一标识每一个被acceptor批准的提案，在这种情况下，当一个具有某Value值的提案被过半acceptor批准后，就意味着该Value被选定了，同时也意味着该提案被选定了。**  最终选定结果大体是：一个值为Vx的Value被选定了，同时多个提案被选定了，这些提案的编号可能各有不同，但是它们的Value值一定都为Vx。 |

接下来，在B1、P1的基础上，做需求P2：

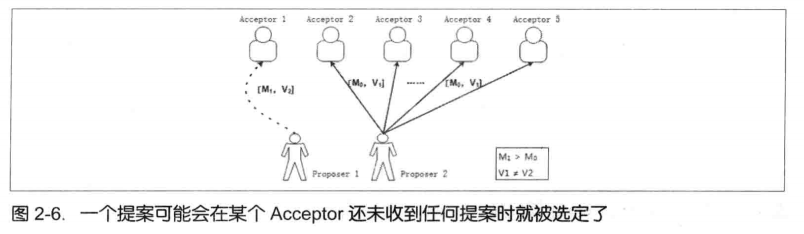
|  |
| --- |
| **P2：如果提案[M0, V0]被选定了，那么所有编号比M0更高且被选定的提案，其Value必须也是V0。** |

因为提案的编号是全序的，P2就能保证T2。也就是说，如果P1和P2都能够被满足，那么Paxos的目标就能够达成了。

如何才能满足P2呢？一个提案必须要经过被acceptor批准才能被选定，因此要满足P2，我们只要满足下面的条件P2a：

|  |
| --- |
| **P2a：如果提案[M0, V0]被选定了，那么所有编号比M0更高且被acceptor批准的提案，其Value必须也是V0。** |

但是P2a存在问题：因为通信是异步的，一个提案可能会在某个acceptor还未收到任何提案时就被选定了。如图2-6所示，在acceptor1没有收到任何提案的情况下，其他4个acceptor已经批准了来自proposer2的提案[M0,V1]，而此时，proposer1产生了一个具有其他Value值的、编号更高的提案[M1,V2]，并发送给acceptor1。根据P1，就需要acceptor1批准该提案，但是这与P2a矛盾。

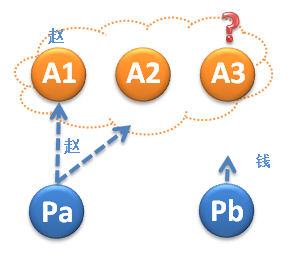


因此如果要同时满足P1和P2a，需要将P2a强化为P2b：

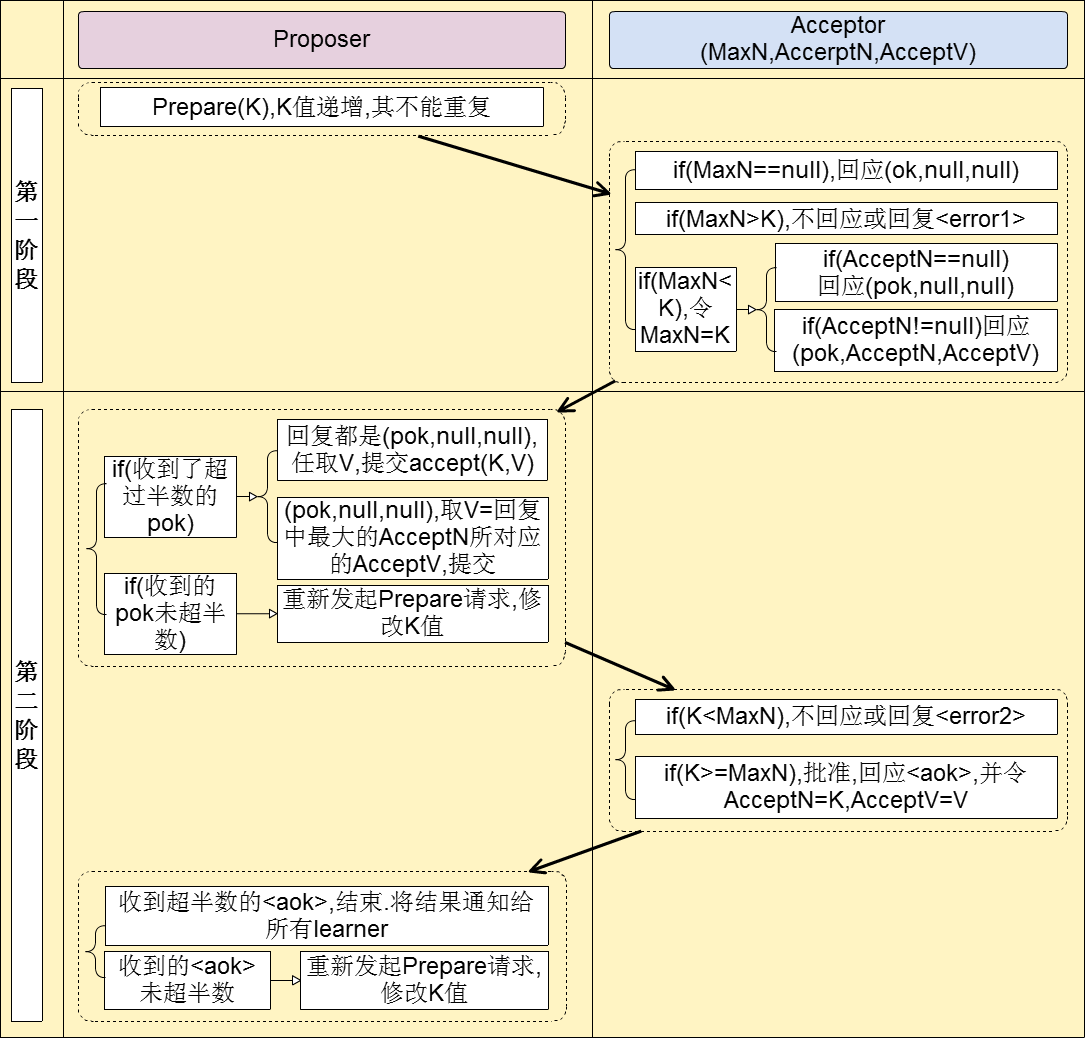
|  |
| --- |
| **P2b：如果提案[M0, V0]被选定了，那么之后任何proposer产生的编号更高的提案，其Value必须也是V0。** |

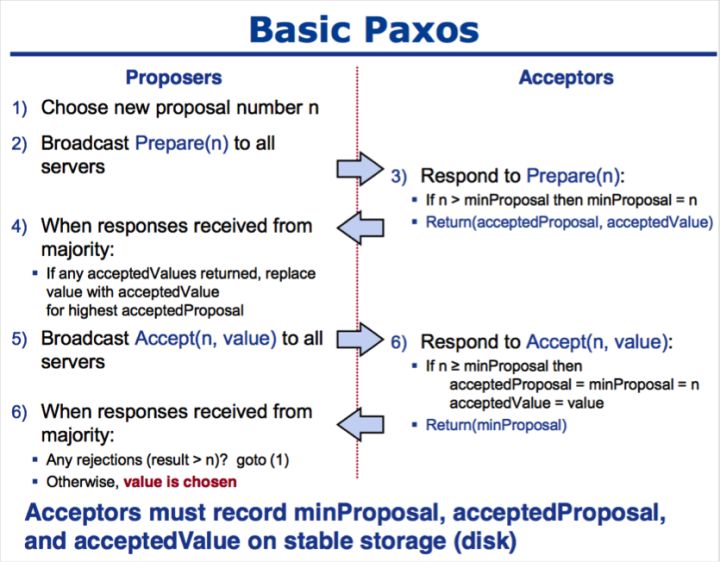
因为一个提案必须在被proposer提出后才能被acceptor批准，所以P2b包含了P2a，进而包含了P2。

但P2b很难被满足，考虑下图这种情况，A1接受了议案“赵”，A2即将接受议案“赵”，此时Pb提出了一个议案“钱”，这种情况下我们又会遇到跟P2a完全相同的麻烦。



## Paxos运行流程





1. 获取一个Proposal ID n，为了保证Proposal ID唯一，可采用时间戳+Server ID生成；
2. Proposer向所有Acceptors广播Prepare(n)请求；
3. Acceptor比较n和minProposal，如果n>minProposal，minProposal=n，并且将 acceptedProposal 和 acceptedValue 返回；
4. Proposer接收到过半数回复后，如果发现有acceptedValue返回，将所有回复中acceptedProposal最大的acceptedValue作为本次提案的value，否则可以任意决定本次提案的value；
5. 到这里可以进入第二阶段，广播Accept (n,value) 到所有节点；
6. Acceptor比较n和minProposal，如果n>=minProposal，则acceptedProposal=minProposal=n，acceptedValue=value，本地持久化后，返回；否则，返回minProposal。
7. 提议者接收到过半数请求后，如果发现有返回值result >n，表示有更新的提议，跳转到1；否则value达成一致。