## 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。

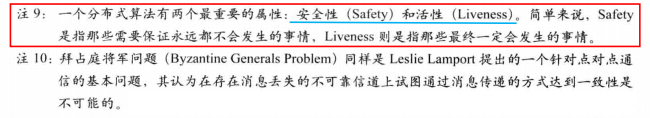
Learner：最终决策的学习者。

===============================================

Paxos是一种提高分布式系统容错性的一致性算法．其核心是一个synod算法（会议算法、共识算法）。

接下来基于论文《paxos made simple》，从一个一致性算法所必须满足的条件展开，来向读者讲解Paxos作为一种一致性算法的合理性。

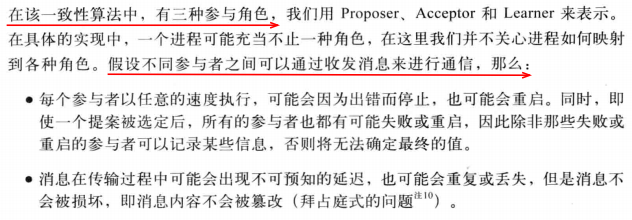
## 01、对一致性问题的描述



假设有一组可以提出提案的进程集合，那么对于一个一致性算法来说，**安全性需求**如下：

|  |
| --- |
| 1、只有被提出的提案才能被选定。 2、只能有一个值被选定。  3、当一个提案被选定后，进程才能获取被选定的提案信息（A process never learns that a value has been chosen unless it actually has been.） |

我们不去精确地定义其活性（Livencss）需求，从整体上来说，Paxos算法的目标就是要保证最终有一个提案会被选定，当提案披选定后，进程最终也能获取到被选定的提案。



## 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，最终只能选定一个值（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。**

## 02、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就只能发送提案给该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一种更好的解决方式。首先我们得使用**多个acceptor**来避免acceptor的单点问题，在这种情况下，proposer会向一个acceptor集合发送提案，我们可以约定每个acceptor最多只能批准（accept）一个提案，并且当所有acceptor集合中过半acceptor都批准了同一个提案时就意味着该提案被选定了，这样就能保证只有一个提案被选定了。

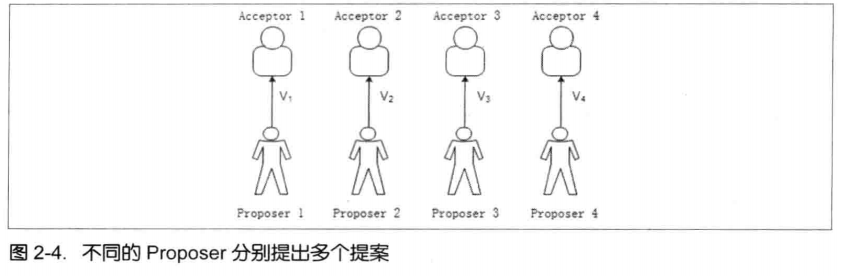
根据上面的讨论，paxos算法要满足的如下的基本需求：

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

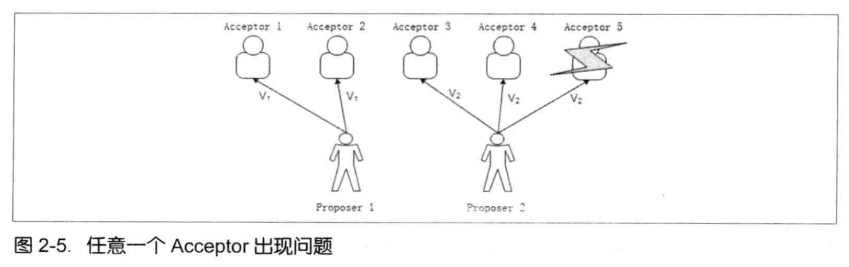
接着继续进行细化，先分析一种特殊情况，对于（一次选举中）只有一个提案被（一个proposer）提出的情况，要求最终能够选定一个值（目标T1）。如果acceptor能够拒绝其收到的第一个提案，就可能导致最终没有过半acceptor批准这个唯一的提案，也就没有提案会被选定。所以paxos算法必须满足的第一个需求P1是：

|  |
| --- |
| **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，接着分析其它可能出现的问题。如图2-4，一次选举中可能有多个提案被不同proposer同时提出，虽然每个acceptor都批准了它收到的第一个提案，但是最终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，只要任意一个acceptor出现问题，最终也可能出现无法选定提案的情况。~~



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“[全局唯一编号，Value]”来代表一个提案，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。** |
| 注意：全局唯一编号生成并不是paxos算法需要关注的地方，就算法本身而言，其假设当前已经具备这样的外部组件能够生成一个全局唯一的编号。 |

B1预示着我们允许多个提案被选定，但是同时必须要保证所有被选定的提案都具有相同的Value值，也就是paxos算法必须要满足需求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值必须也是V0。** |

因为提案的编号是全序的，P2就能保证目标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。**

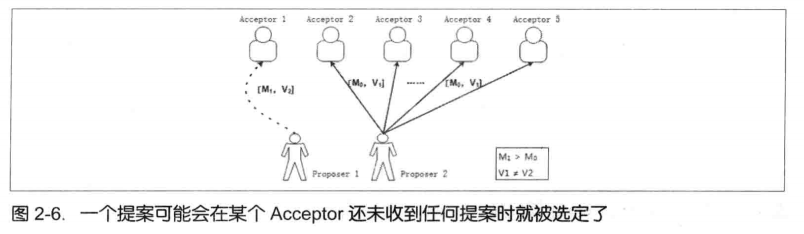
-----------------------------------------------------------------------------------------------------------------------------------------------

**接下来我们讨论，在满足B1和P1的情况下，如何才能满足P2需求**。

首先，一个提案必须要经过被acceptor批准才能被选定，因此我们可以通过满足如下的P2a来满足P2：

|  |
| --- |
| **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值必须也是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，就需要将P2a强化为P2b：

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

-----------------------------------------------------------------------------------------------------------------------------------------------

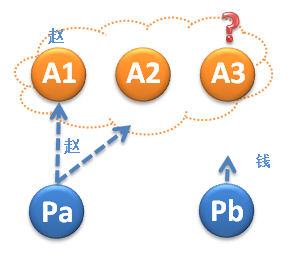
**接下来要讨论，在已经选定了第一个提案[M0,V0]的情况下，proposer要以怎样的方式提出提案才能满足P2b需求**，下面的P2c则规定了每个proposer如何产生一个提案的方式。

|  |
| --- |
| **P2c：对于每个产生的提案[Mn,Vn]，必须存在一个由过半acceptor组成的集合S，满足下面两个条件之一：**  **①要么S中没有acceptor批准过编号小于Mn的任何提案。**  **②要么S中所有Accoptor批准的所有编号小于Mn的提案中，编号最大的那个提案的Value值为Vn。** |

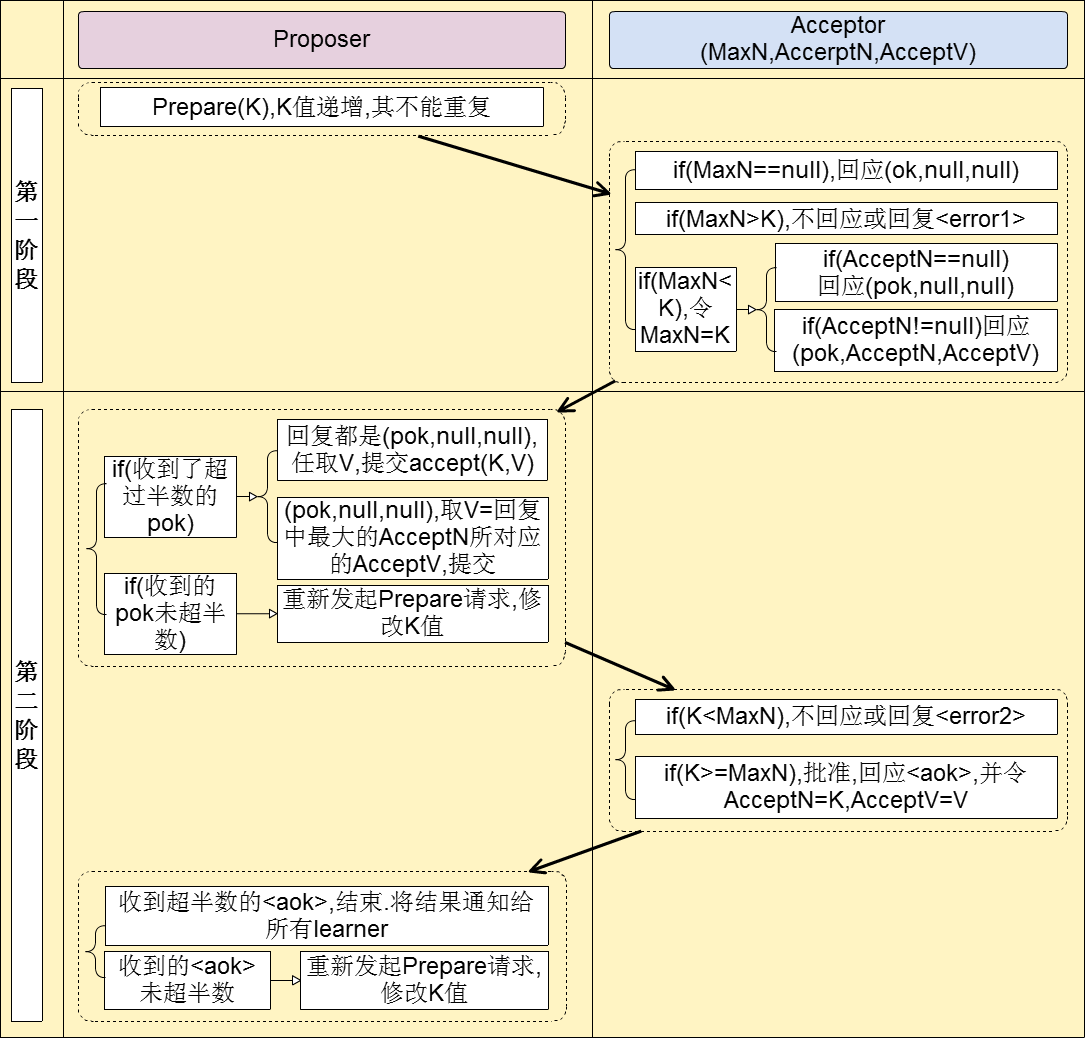
接下来用**第二数学归纳法**来证明：在已经选定了第一个提案[M0,V0]的情况下，如果每个proposer以P2c的方式来产生提案，就能满足P2b需求。

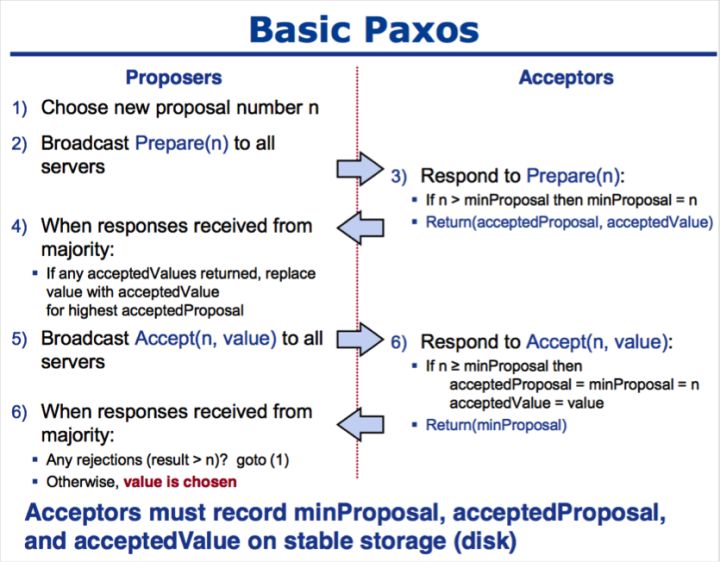
|  |
| --- |
| **首先要证明，当Mn=M0+1时，Vn=V0成立。**  首先，由于[M0,V0]是第一个被选定的提案，所以在编号为(M0+1)的提案被提出来并批准之前，在集群所有acceptor批准的提案当中，最大的提案编号必为M0。且此时由于[M0,V0]被选定了，所以肯定存在一个由过半acceptor组成的集合C，其中每个acceptor都批准了[M0, V0]。  然后当我们提出了编号为(M0+1)的提案时，就一定存在一个由过半acceptor组成的集合S满足P2c。由于S跟C肯定有交集，也就是说S中肯定存在已经批准了[M0,V0]提案（注：编号小于Mn）的acceptor，于是根据P2c，Vn只能是S中所有Accoptor批准的所有编号小于Mn的提案中，编号最大的那个提案的Value值。而此时由于集群当中已被批准的编号最大的提案的编号为M0，所以S中已被批准的编号最大的提案的编号就是M0，所以Vn就是编号为M0的提案的值V0，命题成立。 |
| **接下来要证明，假设当Mn<=M0+k时，Vn=V0成立，能推导出当Mn=M0+（k+1）时Vn=V0也成立。**  根据假设，编号在(M0+1)到(M0+k)区间内的所有提案的Value值为V0，需要证明的是编号为(M0+(k+1))的提案的Value值也为V0 。由于[M0,V0]被选定了，所以肯定存在一个由过半acceptor组成的集合C，其中每个acceptor所批准的提案都满足编号在(M0+1)到(M0+k)之间，值为V0。  当我们提出编号为(M0+(k+1))的提案时，就一定存在一个由过半acceptor组成的集合S满足P2c。由于S跟C肯定有交集，也就是说S中肯定存在已经批准了编号在(M0+1)到(M0+k)之间的提案（注：编号小于Mn）的acceptor，于是根据P2c，Vn只能是S中所有Accoptor批准的所有编号小于Mn的提案中，编号最大的那个提案的Value值。如果这个最大编号落在(M0+1)到(M0+k)区间内，那么Value值肯定是V0；如果不落在(M0+1)到(M0+k)区间内，由于S和C一定有交集，所以它的编号不可能比M0再小了，肯定就是M0,那么Value值就是编号为M0的提案的值V0，命题成立。 |

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。