#### basic paxos学习笔记

学了paxos算法后，想实现一个paxos工程，但需要对paxos算法进行安全性和容错性的证明，以及一些工程细节的脑补。

（算法过程大量摘抄或意译《从paxos到zookeeper分布式一致性原理与实践》这本书(简称《从paxos到zookeeper》）或者paxos论文（paxos made basic）以及b站视频《分布一致性算法Raft作者讲解Paxos》）

安全性指的是：1）只有被提出的提案才能被选定；2）只有一个值被选定；3）如果某个进程觉得某个提案被选定了，那么这个提案必须真的被选定了。）

容错性指的是串行容错：（即就算网络出现错误，主机死机，安全性还可以保证，只是可用性差了）将算法按节点的交互来做划分，节点的交互是划分的准则，就类似于编译原理里控制图按jump来划分。每个节点在拓扑上可以是有向有环，比如节点2与节点6交互。那么容错性证明需要做两类证明：节点之间的交互容错（即看着拓扑图，对每一个边，问如果这个边出现网络延迟或网络断开，算法还正确吗？）；和节点内的逻辑代码容错（即看着拓扑图节点的内容（逻辑代码），逐行问，如果运行到第n行到第n+1行之间，电脑死机了，算法还正确吗？）

并发容错是串行容错的复合版，指节点交互，或节点内代码间都出错，算法还正确吗？容易知道，算法是串行容错的可推出算法是并发容错的。

先从算法的理解开始，写一下basic paxos需要满足的性质与算法实现之间的紧密联系。

在《从paxos到zookeeper》（或者paxos论文）里，一致性协议想要满足的性质：

（注意：一个acceptor可以批准多个提案。这里《从paxos到zookeeper》这本书在开始算法讲解时画蛇添足说了句“另外我们再规定，每一个acceptor最多只能只能批准一个提案，那么就能保证只有一个提案被选定了。”，书里这句话是错的。）

P1：一个acceptor必须批准它收到的第一个提案。{强制要acceptor们批准第一个提案，这是个简单粗暴的，强的要求，简单和强都在于直接一开始就批准提案了（一开始就get started工作了起来）。}

P2：如果编号为M0,Value值为V0的提案[M0,V0]被选定了，那么所有比编号M0更高的,且被选定的提案，其Value值必须也是V0。{这样就在acceptor形成多数派时做限制，保证了比M0编号更高的提案和M0提案的共识。}

令人高兴的是，paxos可以做到加强版的P2,即P2A。

P2A：如，果编号为M0,Value值为V0的提案[M0,V0]被选定了，那么所有比编号M0更高的，且被acceptor批准的提案，其Value值必须也是V0。{这样就在acceptor形成多数派前的acceptor端做限制，保证了比M0编号更高的提案和M0提案的共识。}

这里P2A和P1在一些场景中有矛盾，比如可能已经有提案被选定了后，网络延迟导致acceptor收到的第一个提案竟然是编号比被选定了的提案更高可Value值不一样的提案，该acceptor按P1应该批准该提案，可按P2A不该批准该提案，矛盾了。

看来应该在proposer端做限制，保证比M0编号更高的提案和M0提案的共识。

令人高兴的是，paxos可以做到在proposer端做限制,即加强版的P2A——P2B，P2B与P1不矛盾。

P2B:如果一个提案[M0,V0]被选定了，那么之后任何proposal产生的编号更高的提案，其Value值都为V0。{这样就在proposer端做限制，保证了比M0编号更高的提案和M0提案的共识。}

可以看到，保证了P2B就保证的安全性。

其实，paxos可以做到比P2B更强的P2C,P2C可以推出P2B。怎么想到的P2C,paxos作者的想法和读写quorum(w+r>n)很相似。

P2C:对于任意的Mn和Vn，如果提案[Mn,Vn]被提出，那么肯定存在由半数以上的acceptor组成的集合S，满足一下两个条件的任意一个：

I）S中不存在任何批准过编号小于Mn的提案的acceptor。

II）选取S中所有acceptor批准的编号小于Mn的提案，其中编号最大的那个提案，其Value值是Vn。

（简单说下P2C怎么按直觉推出P2B:假设P2C成立，如果某提案[M0,V0]被选定了，而紧接着M0过后的第一个提案是[M1,V1]，[M0,V0]被选定说明有多数派S0批准了[M0,V0],[M1,V1]被提出来成提案，根据P2C,说明存在多数派S1，S1和S0交集非空（都是多数派嘛），那么存在某acceptor属于S1和S0,那么这个acceptor批准过编号小于M1的提案（因为这个acceptor属于S0，它批准过[M0,V0]呀），于是条件I不成立，那么只能条件II成立。于是S1中所有acceptor批准的编号小于Mn的提案，其中编号最大的那个提案，其Value值是V1，这里由于[M0,V0]是紧接着[M1,V1]的,那么“S1中所有acceptor批准的编号小于M1的提案，其中编号最大的那个提案”，其实就是M0嘛，所以M0的Value值是V1，即M1的Value值是V0。同理，如果紧接着M1过后的第一个提案是[M2,V2]，那么“S1中所有acceptor批准的编号小于M1的提案，其中编号最大的那个提案”，其实就是M1嘛，所以M1的Value值是V2，即M2的Value值是V1等于V0。然后可以一直下去。证完。（也可将这直觉转写成数学归纳法））

如果paxos可以保证P2C,那么paxos算法可以达成共识了。

Paxos算法是怎样的呢？怎么证明它能保证P2C呢?证明很简单，因为它基本是由P2C量身定做的。

先说paxos算法（的某一个版本，当然这个版本的paxos某些步骤还可以删减，并且还不会损害安全性。可能可以研究下，最简（最懒）paxos是怎样的，可想而知，最简paxos的可用性很差。相反的，这个版本的paxos还可以做一些优化，提高它的可用性），再说它与P2C的紧密联系。

Paxos算法是一个两阶段算法：

首先，每一个proposer有一个提案编号生成器，并且假设提案的编号是对每个proposer递增且全局唯一（有很多方法可做到这个），proposer需要本地保存一个值：最后生成的提案编号；然后，acceptor需要本地保存3个值：自己在prepare阶段回应的最大提案编号maxRespondedProposal；自己在accept阶段批准的最大提案编号maxAcceptedProposal，自己在accept阶段批准的最大提案编号maxAcceptedProposal相对应的Value——maxAcceptedValue。

Prepare阶段：

1）proposer本地的提案编号生成器生成M,然后向多数派的某acceptor集合发送prepare请求（这个在prepare阶段的请求称为prepare请求，prepare请求里只包含信息M）。

2）acceptor收到prepare请求后，acceptor比较该prepare请求对应的M与本地保存的maxRespondedProposal做比较，如果 M>maxRespondedProposal,则本地修改maxRespondedProposal成M,并且回应给该proposer[maxAcceptedProposal,maxAcceptedValue]。

（两者为空的话回应[null,null]），如果M<=maxRespondedProposal,acceptor不修改maxRespondedProposal，并且不做任何回应。

（即if M>maxRespondedProposal

            set maxRespondedProposal=M

            return [maxAcceptedProposal,maxAcceptedValue]

        else

            do nothing

）

（这里的如果M>maxRespondedProposal才回应，其实是paxos论文或《从paxos到zookeeper》提到的优化，这里直接整合进算法了，如果不管是否大于maxRespondedProposal都回应[maxAcceptedProposal,maxAcceptedValue]也可行，但只有>才返回是优化了，可以帮助3）做决策）

3)如果某proposer接受到的回应集合形成了一个多数派，那么看这些回应的[M,V]，取其中最大的提案编号M1所对应的Value值V1,那么这个V1就是接下来accept阶段proposer的提案的Value值。当然有退化情况，是全部接受的回应都是[null,null],那么这个Value值随proposer取。

Accept阶段：（在3）取得的回应形成多数派后，才会开启Acceptor阶段）

4)Proposer向多数派的某acceptor集合发送accept请求（这个在accept阶段的请求称为accept请求，accept请求里包含信息[M,V1]）

5)Acceptor收到accept请求后，看accept请求里包含的M是否大于等于本地的maxRespondedProposal，如果大于等于，则修改maxAcceptedProposal成M,修改maxRespondedProposal成M,修改maxAcceptedValue成V1，并且回应给该proposer maxRespondedProposal的值。如果小于，acceptor不做任何修改，但还是回应给该proposer maxRespondedProposal的值。

（即if M>maxRespondedProposal

            set maxAcceptedProposal=M

            set maxRespondedProposal=M

            set maxAcceptedValue=V1

        else

            do nothing

        return maxRespondedProposal

）

（这个“如果M>=maxRespondedProposal”里的有=号，注意，和2）相比，多了个等号，当然2）也可以大于等于，等于的话做了一个无用的等值修改罢了（之前已经批准过了），这里5）也可以只有大于没有等于，但有了等于相当于多做了有效决策，帮助共识生成。另外，和2）不同的是，5）总是返回maxRespondedProposal 才是paxos论文或《从paxos到zookeeper》提到的优化，这里直接整合进算法了，而如果M<=maxRespondedProposal时，不返回maxRespondedProposal 也可行，不过两种情况都返回是优化了，可以帮助6）做有效决策）

6)如果某proposer接受到的回应集合形成了一个多数派，看接收到的各acceptor的maxRespondedProposal，如果有任意一个maxRespondedProposal大于提案编号M,则回到1）重新开始生成新的提案编号，重新开始二阶段。如果任意一个maxRespondedProposal都小于提案编号M,那么提案的Value值V1是被选定了的共识。

现在看basic paxos需要满足的性质与算法实现之间的紧密联系。

Basic paxos需要满足P1和P2C。

再写下：

P1:一个acceptor必须批准它收到的第一个提案。

P2C:对于任意的Mn和Vn，如果提案[Mn,Vn]被提出，那么肯定存在由半数以上的acceptor组成的集合S，满足一下两个条件的任意一个：

I)S中不存在任何批准过编号小于Mn的提案的acceptor。

II)选取S中所有acceptor批准的编号小于Mn的提案，其中编号最大的那个提案，其Value值是Vn。

P2C有点tricky，它表面上说的是，提案[Mn,Vn]被提出的这个时间点t0，存在由半数以上的acceptor组成的集合S balabala。但实际上，由于网络延迟或断开的原因，P2C应该保证在t0后的时间内，也存在存在由半数以上的acceptor组成的集合S balabala。

所以其实paxos算法做到了保证比P2C更强的P2C’

P2C’：对于任意的Mn和Vn，提案[Mn,Vn]被提出的当时t0，以及t0之后的时间里，肯定存在由半数以上的acceptor组成的集合S，满足一下两个条件的任意一个：

I)S中不存在任何批准过编号小于Mn的提案的acceptor。

II)选取S中所有acceptor批准的编号小于Mn的提案，其中编号最大的那个提案，其Value值是Vn。

P2C’需要算法去除这两种情况：a）某个编号小于Mn的t0时刻编号最大的提案在t0后的时间点t1被s的某个acceptor重新批准了另一个Value值，或者b）编号小于Mn但编号更大的提案在t1被S的某个acceptor批准了另一个Value值。

首先a）不可能发生，算法层面，一个编号的提案只能在一个二阶段中存活，后被新的不同编号的提案替代，同一个编号的提案只能被发起一次，所以不可能有两个同一编号不同Value值的提案被提出，当然更不可能被批准。

然后b）是由算法的2）和5）保证的，即书和论文里说的“acceptor向proposer承诺，保证不再批准任何编号小于(prepare阶段发来的)M的提案”。

然后由算法的3），对应着P2C’的两个条件。

看看3）:如果某proposer接受到的回应集合形成了一个多数派，那么看这些回应的[M,V]，取其中最大的提案编号M1所对应的Value值V1,那么这个V1就是接下来accept阶段proposer的提案的Value值。当然有退化情况，是全部接受的回应都是[null,null],那么这个Value值随proposer取。

这里的“如果某proposer接受到的回应集合形成了一个多数派，那么看这些回应的[M,V]，取其中最大的提案编号M1所对应的Value值V1,那么这个V1就是接下来accept阶段proposer的提案的Value值。”对应着P2C’的条件II)，而“当然有退化情况，是全部接受的回应都是[null,null],那么这个Value值随proposer取”对应着P2C’的条件I)。

当然还要保证P1,其实算法的5）就是保证了P1.

于是算法可以保证了P1和P2C’，P2C’=>P2C=>P2B,而P2B保证了安全性，所以算法的安全性是可以保证的。

剩下容错性的证明。容错性的证明用到开头说的框架，即“那么容错性证明需要做两类证明：节点之间的交互容错（即看着拓扑图，对每一个边，问如果这个边出现网络延迟或网络断开，算法还正确吗？）；和节点内的逻辑代码容错（即看着拓扑图节点的内容（逻辑代码），逐行问，如果运行到第n行到第n+1行之间，电脑死机了，算法还正确吗？）”

首先是节点之间的交互容错。

假设算法的1）到2）之间proposer和acceptor网络不通，这没有关系，因为acceptor是可以自由忽略prepare请求的。（只是忽略后，3）的多数派更难形成了，即可用性差了。）

假设算法的2）到3）之间acceptor和proposer网络不通，同样没有关系，如果还能形成多数派，那算你牛。

而算法3）到4）在同一个prooser节点上。

假设算法的4）到5）之间acceptor和proposer网络不通，也没关系，因为因为acceptor是可以自由忽略accept请求的。

假设算法的5）到6）之间acceptor和proposer网络不通，同理也没关系。

难的是“节点内的逻辑代码容错（即看着拓扑图节点的内容（逻辑代码），逐行问，如果运行到第n行到第n+1行之间，电脑死机了，算法还正确吗？）”

这部分和工程的实现细节有关。

待续。。。