#### 之前basic paxos 文章勘误 以及容错性补充（简要）证明

之前basic paxos 文章勘误 以及容错性补充（简要）证明。可能写的比较简单，有人不理解的话，可以和我讨论。。。

算法的5）原来是：

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 maxRespondedProposal=M

            set maxAcceptedProposal=M

            set maxAcceptedValue=V1

       else

            do nothing

       return maxRespondedProposal

）

首先，这个 return maxRespondedProposal 可以是只有if条件成立才return，也可以是不过if条件成立是否成立都return，安全性都可以保证。

但我之前的文章里说，“不管if条件是否成立，最后都返回maxRespondedProposal 是优化”，这句话是错的。正确的优化应该是：只有if条件成立才返回。

（即if M>maxRespondedProposal

            set maxRespondedProposal=M

            set maxAcceptedProposal=M

            set maxAcceptedValue=V1

            return maxRespondedProposal

    else

            do nothing

）

并且注意：这个逻辑代码先后顺序是有讲究的，应该先set maxRespondedProposal=M,再设maxAcceptedProposal 和maxAcceptedValue，这样结合算法的2）可以一直保证，acceptor本地持久化的maxRespondedProposal>=maxAcceptedProposal（即使在这两行代码间，持久化maxRespondedProposal后 持久化maxAcceptedProposal前，节点断电，也可以保持这个>=不变性，而这个5）的逻辑代码的容错性证明，需要依赖这个不变性）。

还有 应该将maxAcceptedProposal和maxAcceptedValue组合起来，一起持久化，而不是分开持久化，分开持久化的话，可能两个持久化代码之间主机断电，这样只有maxAcceptedProposal持久化成功，重启主机后，会造成数据的不一致性，算法就错误了。

之前文章的容错性证明讨论了节点之间的交互容错，于是只剩下——“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？）”

（注：正确指的是保证安全性，活性的话先不考虑）

首先算法2）和5）保证了acceptor本地持久化的maxRespondedProposal总是>=maxAcceptedProposal，将它称作R-A不变性。（为什么不变？供讨论。。。）。

算法的1）的逻辑代码是：

只是生成提案编号M；

发出prepare(M)请求；

可以看到，逻辑代码的开始，中间，结尾有任何故障，不会损害安全性。（大不了不发出prepare请求呗，或者发出老的M比较小的prepare请求然后不能被拒绝，不能成功走进accept阶段呗）

算法2）的逻辑代码是：

if M>maxRespondedProposal

    set maxRespondedProposal=M

    return [maxAcceptedProposal,maxAcceptedValue]

else

    do nothing

可以看到2）不会损坏R-A不变性，并且如果一开始该acceptor节点死机或怎样，那就相当于忽略prepare请求呗，而这个忽略是不影响安全性的（为什么？供讨论）；If后，set前死机，也相当于忽略prepare请求；set 后，return前死机，相当于发送了有效信息，但proposer没收到，但这不会损害安全性，因为proposer如果没拿到我的有效信息，还形成了多数派进入accept阶段，算你牛。Else那什么没做，不会损害安全性。

算法3）和4）在同一个节点,代码逻辑是：

If 接受到的回应集合形成了一个多数派

        If 全部回应都是[null,null]

    向某多数派的acceptor集合发送[M，V](V随便取）

    Else 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，向某多数派的acceptor集合发送[M，V](V随便取）

由于这个逻辑代码没有持久化什么，里面有什么事故，都可以归结为最开始的事故和最后面的事故，其实就归结为节点之间的交互容错了嘛，而节点间的交互容错，是不影响安全性的。

算法的5）的逻辑代码是：

if M>maxRespondedProposal

    set maxRespondedProposal=M

    set maxAcceptedProposal\_maxAcceptedValue=M\_V1

    return maxRespondedProposal

else

    do nothing

注意，一开始没有收到请求，安全性没问题。

其实都不需要逐行分析，只需证明set maxRespondedProposal=M 后 ，set maxAcceptedProposal\_maxAcceptedValue=M\_V1 前主机死机，重启后不损害安全性即可。

注意到这和一开始没有收到请求的区别是，这里持久化了更大的maxRespondedProposal，但是本地存了更大的maxRespondedProposal，算法里受到影响的地方只有2),而maxRespondedProposal，就只是条件更苛刻了呗，不然不理你，这无损与安全性。

算法的6）里，中间代码间的死机，最多是让这个proposer不知道某个它发出的[M,V]已经被选定而已，可被选定的事实，已经在acceptor里储存了。不知道不影响安全性。

综上，算法的容错性证明完毕。（很简陋。。。）​​​​