mp.weixin.qq.com

拜占庭将军问题和FLP的启示

> 导语：
>
> 本篇继续介绍分布

> 式系统一致性发展史中的重要知识——拜占庭将军问题和FLP原理。

Byzantine Generals Problem

昨天那篇文章提到的Two General Paradox实际上是一

个弱化的Byzantine Generals Problem. 1982年, Leslie Lamport和另外两位科学家年发表了一篇论文阐述 [The Byzantine Generals Problem, ACM Transactions on

Programming Languages and Systems,Volume 4 Issue 3, July 1982 Pages 382-401] 描述了更一般化的情况, 这种情况下不仅是网络会有故障, 节点本身也会有不按

照逻辑执行的问题.
比如一个叛徒将军
乱发消息或者不按
照程序逻辑执行. 完
整的拜占庭将军问
题更加复杂, 必须加
以特定场景的假设
才能解决, 比如同步
网络. 这个问题比较

复杂, 本文只是简要介绍一下.

在同步网络中, 有3f+1个节点, 如果故障节点不超过f个, 那么这个问题是可以解决的. 我们这里不做严格证明, 但是可以简单解释一下原

因。

我们考虑最基础的一种情况，假设有三个将军，只有一个叛徒。如果A是叛徒，那么A可能会给B发出进攻(1)，然后给C发出撤退(0)的命令。当B和C互相同步信息

的时候他们会发现两个不一致的信息. 但是B和C谁也无法判断谁是叛徒, 比如从B的角度来看, 他无法判断A是叛徒或者C是叛徒. 所以三个将军里有一个叛徒是无法解决的. 如

果消息可以防止伪造，那么在同步网络中叛徒达到1/3也是可以解决的. 下图右边从C的角度来看，因为消息是真实无法伪造的，那么很明显A是叛徒. 由此可以推导到3f+1的情

况.

但是同步网络现实生活中太少，如果要考虑在异步网络之中，拜占庭将军问题是非常难解决的，实际上根据FLP定理，

异步网络中是没有完全同时保证safety和liveness的一致性算法的. 但是在实际工程中我们如果放松liveness的要求, 是有实际可用的算法的, 这个算法进入无限循环的概率非

常非常低. 1999年Miguel Castro和Barbara Liskov提出了PBFT算法(Practical Byzantine Fault Tolerance), 这个算法可以在异步网络中不保证liveness的情况下解

决拜占庭将军问题.
虽然不保证Liveness
但是这个算法进入
无限循环的概率非
常低, 在工程中是完
全可用的. 实际上他
们实现了一个PBFT
的分布式NFS文件
服务器, 最坏的时候

性能只下降了24%!
为此BarbaraLiskov
获得了2008年代图
灵奖. 有兴趣的同学
可以自己去看
Practical Byzantine
Fault Tolerance and
Proactive Recovery.
(ACM Transactions

on Computer Systems, Vol. 20, No. 4, November 2002, Pages 398–461).

顺便提一句, 面向对象中的Liskov替换原则也是她提出的.

图片来源:维基百科,
Barbara Liskov
2010

FLP Impossibility

分布式事务作为Consensus类型问题, 在异步网络中非常难实现. 很多科学家们做了很多伟大的尝试, 包括2PC, 3PC, 等等, 但是1985年的时候, 一个重要的论文告诉了

我们答案，这就是著名的**FLP Impossibility:**

No completely asynchronous consensus protocol can tolerate even a single unannounced process death. [

Impossibility of Distributed Consensus with One Faulty Process,Journal of the Association for Computing Machinery, Vol. 32, No. 2, April 1985]

在异步网络环境中只要有一个故障节点, 任何Consensus算法都无法保证正确结束. (这里unannounced process death是指一个进程停止工作了但是其它节点不

知道, 其它节点认为是消息延迟或者这个进程特别慢. FLP假设没有拜占庭的故障节点, 那种情况过于困难, 故障在这里的定义是指进程停止尝试读取消息, 相当于crash-stop).

FLP中设计的模型是一个比现实情况要更可靠的模型，当然了，如果连更可靠的模型下一致性问题失效那么现实中更宽松的环境当然也是失效的. FLP还假设异步网络是可靠

的, 尽管有延迟但是所有的消息都会投递一次且仅一次, 每个进程只会写入一次状态, 然后就进入了decision state, 这是一个很强的保证, 几乎没有任何网络能达到这样的可靠

性. FLP并不要求所有非故障节点都达成一致, 只要有一个进程进入decision state就算达成一致了, 而且一致结果只能是属于{0, 1}, 这也是非常”容易”的agreement约束. 加

上前面还提到最多只有一个进程发生故障，相对于现实情况这已经是一个极端可靠的环境了，但是在这样可靠的环境中仍然无法有一个一致性算法存在！更不用说真实世界

中的网络分区问题
和拜占庭式问题了.
FLP的证明告诉我们
一致性算法的
liveness是无法保证
的, 如果你要safety
那么就会可能进入
无限循环, 每次状态
变化都会可能保持

当前的状态是可分支的(bivalent). FLP的严格证明很有趣,但限于篇幅这里就不做介绍了. 有兴趣的同学可以去看原论文, 如果你觉得里面的数学语言比较晦涩, 可以看我写的

一篇白话文的证明过程描述(链接：http://danielw.cn/FLP-proof/).

图片来源:MIT, Nancy Lynch, 两次Dijkstra奖获得者.

所有consensus问题最终在异步网络上只要有一个故障节点就都无法达成完全一致并结束. 这个理论的证明非常重

要，它终止了多年的争论，现在你可以省省力气了，不要再浪费精力去试图设计一个能在异步网络上能够容忍各种故障并保持一致的系统了。比如分布式事务是永远无法实现

单体应用级别的一致性的.

无论是Paxos还是Raft算法, 理论上都可能会进入无法表决通过的死循环(但是这个概率其实是非常非常低的), 但是他们都是满足safety

的, 只是放松了liveness的要求, Barbara Liskov的PBFT也是这样. 在实际应用中, 上下游系统之间的一致性除了通过2PC或者Paxos Commit来实现, 我们也可以通过

其他方式来实现最终的一致, 来避免2PC的缺点. 现实中你不得不接受短时间的不一致在分布式系统中是一种常态的事实. CAP理论的提出者Eric Brewer曾经这样说

过:

So the general answer is you allow things to be inconsistent and then you find ways to compensate for mistakes, versus trying to prevent

mistakes altogether. In fact, the financial system is actually not based on consistency, it's based on auditing and compensation. They didn't know any thing about the

CAP theorem, that was just the decision they made in figuring out what they wanted, and that's actually, I think, the right decision.

举个例子,实际上你

从A银行往B银行转账实际上是没有两阶段提交或者分布式事务的，金融行业是一个古老的行业，在计算机出现之前银行就已经出现了，金融行业有一点非常值得我们借鉴，那

就是WORM(write once read many).

金融行业把不一致当做常态而非异常.

举个例子, 会计记账的时候, 如果发现前面有一笔预付记录或者错误记录, 会计不会用橡皮抹掉那

条记录再改成正确的，会计只会在最后面加一笔记录来抵消前面的错误，或者按照差额加一笔抵冲的记录。任何记录只能写入一次，然后再也不会改变。我们假设没有人民银行

作为中间节点，把转账简化为A银行直接给B银行转账，在转账过程中如果A已经扣款并通知了B,但是B发现这个账号由于洗钱被锁定了，不能入款，那么B返回拒绝消息给A，这时候A

可以再追加一笔补偿交易，把刚才扣掉的钱补偿回来. 整个过程可能是几秒钟，也可能是几分钟，也有可能是第二天(比如网络故障，重试多次后放弃，对账时发现). 在任何一个步骤

发生故障，用户都会经历一定时间的不一致，从前面的讨论我们知道2PC如果允许超时回滚，2PC也无法消除这个不一致的时间窗口，但是只要有历史记录我们就可以通过自动

补偿或者每日对账去补偿，让数据重新一致. 这种事务可以很好地忍耐各种故障，包括网络分区，只要每个消息都有全局唯一的id或者消息是幂等的即可. 当网络恢复时任何一

个节点都可以根据消息id轻松地去掉重复的消息，当消息丢失时，可以稍后重试或者每天对账.

如果事务包含特别复杂的计算，或者涉及了线下的物流或者多方交易，那么这

样的事务可能需要几天才能完成, 这种长事务(Long Lived Transaction) 在2PC中更是无法处理, 我们总不能去锁定这些数据几天吧?

1987年普林斯顿大学的Garcia-Molina

和Salem发表了一篇论文提出了saga的概念. 一个长事务T中的操作可以拆分为彼此独立的本地事务T1, T2, T3, 那么就可以称之为saga. 其中的每一个Ti都有一个相应的补

偿事务Ci. 如果部分Ti失败了需要Ci来修复回原始状态,Ci不会直接把数据库改回原先的状态, Ci通常是像前面说的会计的做法追加修正内容去抹平Ti带来的变化. 下图中事务参

与者有A,B, C, 每次发起的事务都有一个全局唯一的id, 参与者之间的消息在网络故障时可以重发 (可以通过id去重复消息, 或者保证幂等操作). 假如T3在C中执行时失败了, 如

果T3已经提交了或者这个系统不支持回滚, 那么必须使用C3来补偿T3带来的变化. 然后发消息给B, B会执行C2去补偿T2, 然后B可以选择继续通知A去回滚, 或者稍后等外部条

件发生变化再执行T2, 然后让C去重试. 这样整个系统变成了一个状态机,参与者之间虽然有可能不一致,一个订单或者一笔交易一定会处于状态机的某一个状态, 整个事务的

过程仍然是整体可以追溯的.

Saga这种方法并不适合所有的情况, 它也不是银弹, 但是它是比2PC/3PC更适合来解决分布式事

务. 2PC/3PC的思路是想在源头上阻止分布式事务不一致的产生, 但这是不可能实现的. 不一致是常态, 不是异常, 异步网络中能容忍节点故障的total correct的consensus

算法不存在。

点击查看相关精彩文章：

分布式系统发展史第一篇

分布式系统发展史第二篇第一部分

分布式系统发展史第二篇第二部分

今天是分布式系统发展史第二篇的第三部分内容，下周一将发布本篇最后一部分内容《Paxos算法和Uniform

Consensus》，敬请期待。

如果您想投稿给我们，或者想转发和采用我们的稿件，请回复“合作”，小编会在2小时内回复您的投稿和合作需求。

本文作者：Daniel，吴强， 现任点融网首席社交平台架构师，前盛大架构师，专注分布式系统和移动应用，有十三年的开发经验，目前在点融做最爱的两件

事情: 写代码和重构代码。

随着新一轮融资，点融网开始了大规模的扩张，需要各种优秀人才的加入，如果您觉得自己够优秀，欢迎加

入我们！