**从Paxos到拜占庭容错，兼谈区块链的共识协议（转）**

[**区块链**](http://blockchaindev.org/tags/区块链/) [**分布式算法**](http://blockchaindev.org/tags/分布式算法/)

这是一篇转载文章，原文地址：<http://chuansong.me/n/348459051259>，原文作者data-geek已授权转载。

本文建立在在很多论文的基础上，概括得非常好，比我的《[浅谈区块链共识机制与分布式一致性算法](http://blockchaindev.org/article/consensus_introduction.html)》更深度一些。其中区块链相关的观点和我的非常相似，不知是巧合还是有借鉴。

    高可用架构在分布式系统设计中是最核心的挑战之一，拜占庭容错则是解决高效容错问题的通用方案。拜占庭系统来源于拜占庭将军问题，在古代，一些拜占庭的将军率领他们的部队要攻占敌人的一个城池, 每个将军只能控制他们自己的部队并且通过信使传递消息给其他的将军(这条消息只有参与的两个将军知道，其他的将军可以打听，但是不能验证消息的正确性)。如果这些将军中有些是来自敌方的奸细，那么如何使忠诚的将军仍然可以达成行动协议而不受奸细的挑拨，就是拜占庭将军问题。分布式系统的每一个节点可以类比成将军，服务器之间的消息传递可以类比成信使，服务器可能会发生错误而产生错误的信息传达给其他服务器。因此拜占庭容错系统是指：在一个拥有n台服务器的系统中，整个系统对于每一个请求需满足以下两个条件：

1. 所有非拜占庭服务器使用相同的输入信息，产生一致的结果；
2. 如果输入的信息正确，那么所有非拜占庭服务器必须接受这个信息，并计算相应的结果。

其中，发生故障的服务器称为拜占庭服务器。这两个条件被称为共识Consensus，它是分布式系统容错处理的最基础问题。在拜占庭系统的实际运行过程中，一般假设整个系统的拜占庭服务器不超过f台，并且每个请求还需要满足同时两个指标：

safety：任何已经完成的请求都不会被更改,它可以被之后请求看到;

liveness：可以接受并且执行非拜占庭客户端的请求，不会被任何因素影响而导致非拜占庭客户端的请求不能执行。

在实际网络中，拜占庭系统需要指数级的算法才能解决，随着大规模分布式系统尤其是云计算的兴起，通过放松liveness来提高性能的简化拜占庭协议也不断涌现并在具体系统中得到运用，例如PBFT ，Paxos，Raft等等。从目前研究现状来看，拜占庭系统的设计主要分为状态机拜占庭协议和Quorum拜占庭协议。两种协议在功能上的主要区别在于：前者需要给所有的请求安排序号，所有请求必须按顺序执行，主要用于对于系统状态敏感的分布式计算系统中；后者不需要为请求安排序号，多个请求可以同时执行，主要被应用于分布式存储系统中。拜占庭容错技术的研究方向主要是降低系统开销,缩小与目前实际应用中的系统之间的差距。

状态机拜占庭系统的特点是整个系统共同维护一个状态，所有服务器采取一致的行动，包含三种协议：一致性协议agreement，检查点协议checkpoint，视图更换协议view change。

一致性协议的目标是使来自客户端的请求在每个服务器上都按照一个确定的顺序执行。一般有一个节点作为主节点，负责将客户端的请求排序，其余是从节点，按照主节点提供的顺序执行请求。所有的服务器在相同的配置信息下工作，这个配置信息称为view，每更换一次主节点，view随之发生变化。一致性协议包含至少三个阶段，发送请求request，序号分配pre-prepare，和返回结果reply。根据协议设计不同，还可能会包含交互prepare，序号确认commit等阶段。

拜占庭系统每执行一个请求，服务器需要记录日志。如果日志得不到及时的清理，就会导致系统资源被大量的日志所占用，影响系统性能及可用性。另一方面，由于拜占庭服务器的存在，一致性协议并不能保证每一台服务器都执行了相同的请求，所以,不同服务器状态可能不一致。周期性的检查点协议可以定期地处理日志，及时纠正服务器状态。

一旦主节点自身发生错误，就可能导致从节点接收到具有相同序号的不同请求，或者同一个请求被分配多个序号等问题，这将直接导致请求不能被正确执行。视图更换协议的作用就是在主节点不能继续履行职责时，将其用一个从节点替换掉，并且保证已经被非拜占庭服务器执行的请求不会被篡改。

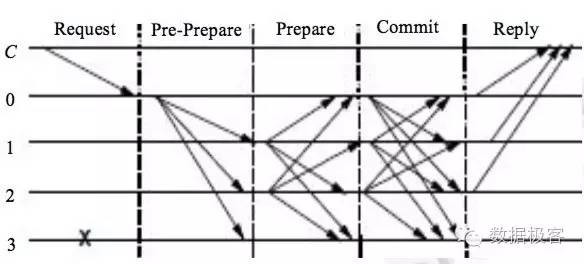
Castro和Liskov[4]早先提出了Practical Byzantine Fault Tolerance(PBFT)，首次将拜占庭协议的复杂度从指数级降低到多项式级别，使拜占庭协议在分布式系统中应用成为可能。这个算法可以在异步网络中不保证liveness的情况下解决拜占庭容错问题。 虽然不保证Liveness，但是这个算法进入无限循环的概率非常低，在工程中是完全可用的。为此Barbara Liskov获得了2008年代图灵奖。PBFT采取两个假设提供状态复制：

1. 系统中只有一小部分数目可能是拜占庭服务器

2. 系统依赖占据主导的选举来达成共识，确保结果正确

如果系统中有3f+1个节点，那么PBFT可以容忍拜占庭服务器不超过f个。简单解释一下原因：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，然后给C发出撤退的命令。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。为了确保正确，PBFT两两交互确定交集后才能确定一个非叛徒：因此由-(N-f)＋(N-f)-N>=f+1推导出N>=3f+1。

PBFT的一致性协议如下图所示，每一个客户端的请求需要5个阶段才能完成。PBFT通过采用两次两两交互的方式在服务器达成一致之后再执行客户端的请求，由于客户端不能从服务器端获得任何服务器运行状态的信息，因此PBFT 协议中主节点是否发生错误只能由服务器监测。如果服务器在一段时间内都不能完成客户端的请求，则会触发视图更换协议。

  
PBFT采取的设计思路是将所有的工作都放在服务器端进行，例如达成一致性，监测拜占庭主节点等。因此，它的一致性协议设计较复杂，其中有两个阶段需要服务器之间的两两交互，数据处理量大，计算复杂。

PBFT在很多场景都有应用，例如Liskov曾经把它用于一种NFS的实现，最近一个热门的应用是在IBM主导的区块链超级账本项目中，除了PBFT之外，超级账本项目还引入了基于PBFT的自用共识协议Sieve[5]，它的目的是希望在PBFT基础之上能够对节点的输出也做好共识，这是因为超级账本项目的一个重要功能是提供区块链之上的智能合约——就是在区块链上执行的一段代码，因此它会带来区块链账本上最终状态的不确定，为此Sieve会在PBFT实现的基础之上引入代码执行结果签名进行验证。

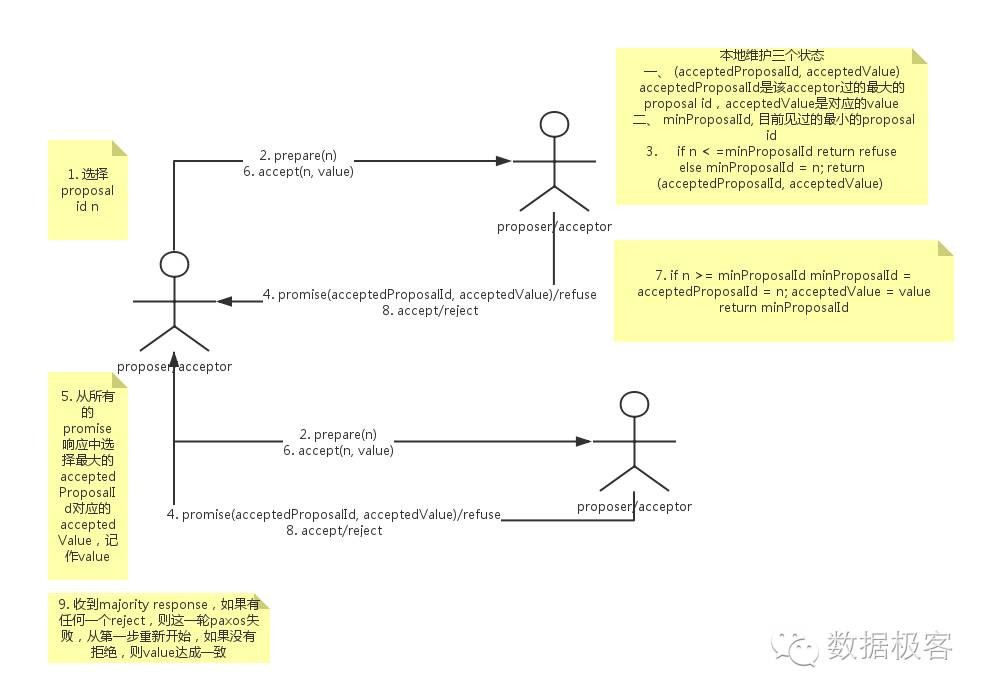
状态机拜占庭系统同一时刻只能执行一个请求，因此服务器之间执行请求顺序要求完全一致。而在许多分布式存储系统中，应用通常不要求严格的执行顺序，但对响应时间很敏感，因此人们提出Quorum拜占庭系统。Quorum系统是指共享数据存储在一组服务器上，通过访问服务器的一个大小恒定的子集(quorum)来提供读/写操作。这类协议都含有一个特性：规定访问的子集大小后，任何一个这样的子集都包含最新的数据，并且一定可以读出来。Quorum 拜占庭系统则是在此基础上保证在服务器出现拜占庭错误时，系统的一致性语义仍然成立。这一类系统写操作过程是：首先，客户端c从某一个大小恒定的服务器子集Q中读取时间戳A，客户端c从可用时间戳t中选取大于任何一个A 中的时间戳并且大于所有c使用过的时间戳；然后，客户端c发送(v,t)给所有服务器(v为需更新数据，t为时间戳)，直到收到某一个大小恒定的服务器子集的反馈信息。这一类系统读操作过程是：客户端c获得某一个恒定大小的服务器子集Q中所有服务器返回的最新数据信息；然后，c按照具体协议要求从中选取需要的数值。

把拜占庭容错系统的场景放松，不允许系统中出现叛徒（也就是说没有黑客攻击，不会出现伪造消息等情况），这样对于数据中心分布式系统构建是普遍情况。Paxos是第一个适用于这种系统的共识算法，但不适用于拜占庭容错系统（Byzanetine Paxos除外）。Paxos运行在称为副本的一组进程集合，这些进程需要在即使发生错误的情况下也能在某一个值上达成一致，如果半数以上的副本在足够长的时间内没有发生crash，那么所有运行中的副本将可以在某个提出的value上达成一致：在状态机复制中，这意味着需要在“(多副本)日志中的下一条记录是哪条”上达成一致。

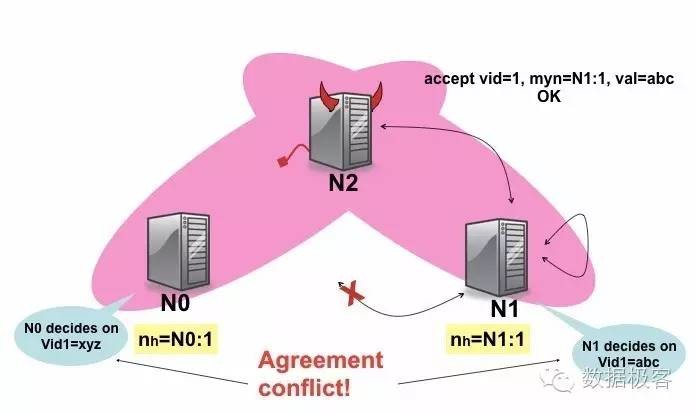
借用吴镝的图说明下Paxos过程：Paxos分为Prepare和Accept两个阶段。协议中有两个主要的角色，Proposer和Acceptor。value被大多数接受之前，每个Acceptor可以accept多个不同的值。但是，一旦一个value被majority accept(即value达成一致)，那么这个value就不会变了。因为Prepare阶段会将该value给找出来，随后Accept阶段会使用这个value，后续的所有的提案都会选择这个value。 需要注意的是，每个阶段都是收到majority的响应后即开始处理。并且由于机器会宕机，Acceptor需要对acceptedProposalID，acceptedValue和minProposal进行持久化。 从流程中可以看出Prepare有两个作用:

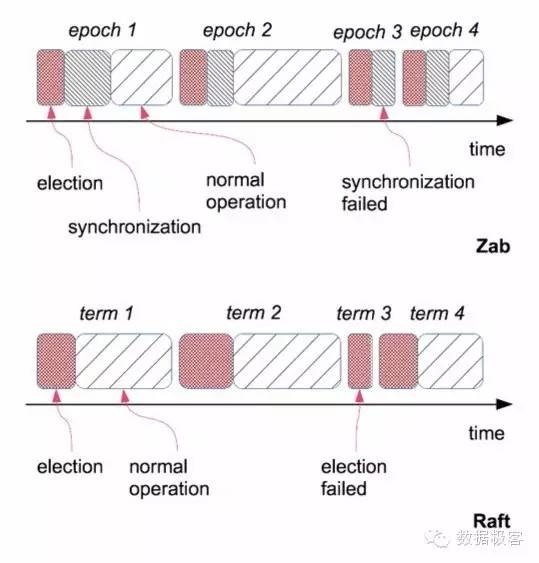
1. 大的proposal id会block未完成的小的proposal id达成一致的过程，所以为了减少无效的Prepare请求，每次都选择比自己以往见过的proposal id更大的id。 2. 一旦某个value达成一致，那么后续的Prepare都会找到这个value作为accept阶段的值可以看出，一次Paxos达成一致至少需要两次网络交互。

（作者：吴镝 链接：https://zhuanlan.zhihu.com/p/20872811 ）



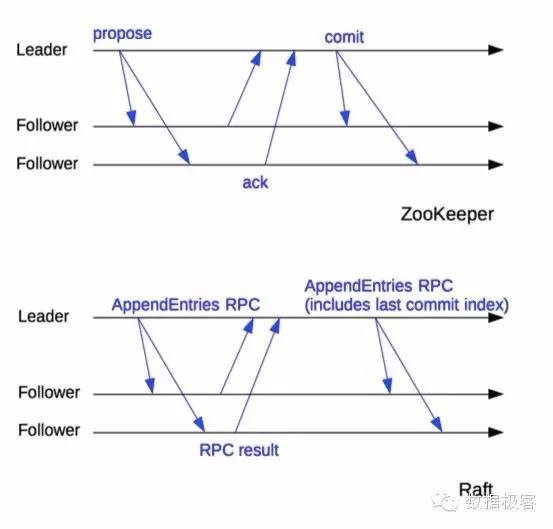
Paxos并不适合于拜占庭容错系统，例如下图中，N2服务器伪造数据，就会让Paxos系统无法达成共识。



  
Zab(ZooKeeper采用的协议)是一个Paxos变种协议，它名字中的AB是原子多播的意思，因为Zab能够让系统里每个节点按照同样顺序执行同样操作。原子多播和一致性选举被认为是等价问题[3]。Raft是近几年出现的一致性选举协议，在完成Paxos等价的正确性和性能的基础上更加容易理解。从上图可以看到，Zab和Raft都有一个专门步骤用于选举出一个Primary Leader，两者均把要解决的一致性问题分解为一系列独立的子问题：Leader选举，日志复制，以及安全和其他活动。Leader的变化，在Zab和Raft中分别由epoch和term表示，每次Leader选举都会增加epoch或者term的值，因此所有的节点只接受epoch或者term值高的Leader节点，在Leader选举之后，由Leader提议所有副本按照统一顺序进行操作。

在所有Paxos类协议中，每个值(就是提案)实质上就是一条日志记录，记录由两部分标记，在Paxos中叫做slot和ballot，在Zab中叫epoch和counter，在Raft中叫term和index。当Leader针对当前日志广播提案时，跟随者依据法定多数(Quorum)选举，并且在Leader提交后在本地应用该提案。所有的Paxos类协议都是保障全局顺序的。

上面讲述了这些Paxos类协议的相似之处，那么它们在设计上也有一些差别。首先是Leader选举，Zab和Raft都把执行分作阶段来进行(epoch或term)，而Paxos没有。每个epoch都从一次新的选举开始，接下来是广播，然后以Leader失败作为结束。在Zab和Raft中，任何时刻至多只能有一个Leader，而Paxos则没有如此要求，因为Paxos并没有独立的Leader选举阶段，因此Paxos系统会存在多个Leader，然而它仍然可以保证正确，这是由ballot的数字以及Quorum决定的。Zab协议中有三个阶段，每个节点在任何时刻都处于这三个阶段中的某一个。发现阶段就是Leader选举；随后是同步阶段，跟随者按照新Leader自上一次epoch以来所有的日志进行同步；最后是广播阶段，也就是上图中的正常工作模式(normal operation)，Leader持续提交客户决议直至它失效。在Raft中，没有明显的同步阶段，Leader在正常工作模式下跟每个跟随者保持同步，这通过比较每条日志的index和term来做到。因此，Raft的状态更为简单一些。

  
其次的差别在于副本之间的通信机制。Zab采用消息模型，每次更新都需要至少三个消息：提议，确认，和提交，如上图所示，而Raft则依赖RPC。

Paxos系统在构建分布式系统有一些典型应用场景：首先最容易想到的是Server复制，利用状态机复制可以同步多个Server之间的状态。其次是元数据复制，由于Paxos系统通常不能操作大量数据，因此一些元数据复制会基于Paxos来进行，比如Apache BookKeeper系统和Kafka。第三种是同步服务，例如各种分布式锁。第四种叫拦阻器编排，例如在分布式图处理框架中，基于BSP的模型需要大量拦阻器来进行同步，这通常也由Paxos系统进行，例如Apache Giraph和Hama，以及Google Pregel等。

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(尽管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, PBFT也是这样。

通常的基础架构中，掌握Paxos协议就已经能够应对所有的高可用设计挑战，然而随着比特币和区块链的热门，Paxos应用的环境就逐渐不再适合，因为随着去中心化网络的引入，分布式系统不再全是可以相信的，因此对于共识算法的需求更加多样，共识算法可以说是区块链为数不多的核心技术之一。目前，区块链应用分为三类：

1. 私有链：这是指在企业内部部署的区块链应用，所有节点都是可以信任的；
2. 联盟链：半封闭生态的交易网络，存在不对等信任的节点；
3. 公有链：开放生态的交易网络，为联盟链和私有链等提供全球交易网络。

由于私有链是封闭生态的存储系统，因此采用Paxos类系统可以达到最优的性能；联盟链有半公开半开放特性，因此拜占庭容错是适合选择之一，例如IBM超级账本项目；对于公有链来说，这种共识算法的要求已经超出了普通分布式系统构建的范畴，再加上交易的特性，因此需要引入更多的安全考虑。

例如比特币采用的工作量证明PoW，就是第一种适用于公有链的共识算法。公有区块链中，参与到系统中的每个节点都是中心，都存放一份完整的交易账本系统。然而，每个节点却不能同时记账，因为节点所处的环境不同，接收到的信息自然不同，如果同时记账的话，必然会导致账本的不一致，造成混乱。既然节点不能同时记账，那就不得不选择哪个节点拥有记账的权力。但是，如果指定某些特殊节点拥有记账的权力，势必又会与去中心化的初衷相违背。比特币区块链通过竞争记账的方式解决了去中心化的记账系统的一致性问题。所谓竞争记账，就是以每个节点的计算能力即“算力”来竞争记账权的一种机制。在比特币系统中，大约每十分钟进行一轮算力竞赛（算力大小会决定赢得一轮竞争的概率，算力高的节点赢得算力竞争的概率更大），竞赛的胜利者，就获得一次记账的权力，这样，一定时间内，只有竞争的胜利者才能记账并向其他节点同步新增账本信息，这就是工作量证明。除了工作量证明之外，还有一些其它的共识算法，主要是解决PoW浪费计算资源，以及吞吐量低下的缺点。例如PoS股权证明机制，基本概念是产生区块的难度应该与在网络里所占的股权(所有权占比)成比例；去中心化股权证明机制DPoS，每个股东可以将其投票权授予一名代表，获票数最多的前100位代表按既定时间表轮流产生区块。

因此，设计区块链，需要在吞吐量，安全等方面做出适合自己的trade off，来选择恰当的共识算法。从区块链的角度来说，Paxos和PoW可以说是trade off的两个极端[6]，而着力于通用区块链的超级账本选取了PBFT也就不难理解了。

[1] Paxos made live: an engineering perspective, Tushar Chandra, Robert Griesemer and Joshua Redstone, ACM Symposium on Principles of Distributed Computing, 2007

[2] Consensus in the Cloud: Paxos Systems Demystified, Ailidani Ailijiang, Aleksey Charapko† and Murat Demirbas, Technical Report 2016

[3] Unreliable failure detectors for reliable distributed systems, T. Chandra and S. Toueg, Journal of the ACM, vol. 43, no. 2, 1996.

[4] Practical Byzantine fault tolerance and proactive recovery, Castro, Miguel and Liskov, Barbara, ACM Transactions on Computer Systems (TOCS), 2002

[5] Non-determinism in Byzantine Fault-Tolerant Replication, Cachin, Christian and Schubert, Simon and Vukoli'c, Marko, arXiv preprint arXiv:1603.0735, 2016

[6] On scaling decentralized blockchains, Croman, Kyle and Decker, Christian and Eyal, Ittay and Gencer, Adem Efe and Juels, Ari and Kosba, Ahmed and Miller, Andrew and Saxena, Prateek and Shi, Elaine and G"un, Emin, Proc. 3rd Workshop on Bitcoin and Blockchain Research, 2016