漫谈分布式系统、拜占庭将军问题与区块链

2018-01-22

最近区块链的话题很火。有人想用它改变世界，有人想用它招摇撞骗。

但是我们今天只分析技术。从技术的角度看，区块链是一种与分布式系统有关的技术。它与分布式系统的各个概念之间有什么联系？今天本文就借这个机会，跟大家一起讨论一下分布式系统的核心问题和概念。最后，我们将尽量沿着逻辑上前后一贯的思路，讨论一下区块链技术。

分布式系统和一致性问题

一个由区块链技术支撑的系统，比如比特币网络或以太坊，从技术上看，是一个很庞大的分布式系统。因此，我们首先从分布式系统开始说起。

对于技术人员来说，特别是服务器开发人员，几乎每个人都经常和分布式系统打交道。当服务的规模越来越大的时候，它必然发展成一个复杂的分布式系统。很典型的，就是各种分布式数据库，它们通常能将数据以某种方式在多个节点上存储，在高可用的基础上保证数据的一致性。

实际上，一致性问题(consensus problem (https://en.wikipedia.org/wiki/Consensus_(computer_science)))是分布式系统需要解决的一个核心问题。分布式系统一般是由多个地位相等的节点组成，各个节点之间的交互就好比几个人聚在一起讨论问题。让我们设想一个更具体的场景，比如三个人讨论中午去哪里吃饭，第一个人说附近刚开了一个火锅店，听说味道非常不错；但第二个人说，不好，吃火锅花的时间太久了，还是随便喝点粥算了；而第三个人说，那个粥店我昨天刚去过，太难喝了，还不如去吃麦当劳。结果，三个人僵持不下，始终达不成一致。

有人说，这还不好解决，投票呗。于是三个人投了一轮票，结果每个人仍然坚持自己的提议，问题还是没有解决。有人又想了个主意，干脆我们选出一个leader，这个leader说什么，我们就听他的，这样大家就不用争了。于是，大家开始投票选leader。结果很悲剧，每个人都觉得自己应该做这个leader。三个人终于发现，「选leader」这件事仍然和原来的「去哪里吃饭」这个问题在本质上是一样的，同样难以解决。

这时恐怕有些读者们心里在想，这三个人是有毛病吧……就吃个饭这么点小事，用得着争成这样吗？实际上，在分布式系统中的每个节点之间，如果没有某种严格定义的规则和协议，它们之间的交互就真的有可能像上面说的情形一样。整个系统达不成一致，就根本没法工作。

所以，就有聪明人设计出了一致性协议(consensus protocol)，像我们常见的比如Paxos、Raft、Zab之类。与前面几个人商量问题类似，如果翻译成Paxos的术语，相当于每个节点可以提出自己的提议(称为proposal，里面包含提议的具体值)，协议的最终目标就是各个节点根据一定的规则达成相同的proposal。但以谁的提议为准呢？我们容易想到的一个规则可能是这样：哪个节点先提出提议，就以谁的为准，后提出的提议无效。但是，在一个分布式系统中的情况可比几个人聚在一起讨论问题复杂多了，这里边还有网络延迟的问题，导致你很难对发生的所有事件进行全局地排序。举个简单的例子，假设节点A和B分别几乎同时地向节点X和Y发出了自己的proposal，但由于消息在网络中的延迟情况不同，最后结果是：X先收到了A的proposal，后收到了B的proposal；但是Y正好相反，它先收到了B的proposal，后收到了A的proposal。这样在X和Y分别看来，谁先谁后就难以达成一致了。

此外，如果考虑到节点宕机和消息丢失的可能性，情况还会更复杂。节点宕机可以看成是消息丢失的特例，相当于发给这个节点的消息全部丢失了。这在CAP (https://en.wikipedia.org/wiki/CAP_theorem)的理论框架下，相当于发生了网络分割(network partitioning)，也就是对应CAP中的P。为什么节点宕机和消息丢失都能归结到网络分割的情况上去呢？是因为这几种情况实际上无法区分。比如，有若干个节点联系不上了，也就是说，对于其它节点来说，它们发送给这些节点的消息收不到任何回应。真正的原因，可能是网络中间不通了，也可能是那些目的节点宕机了，也可能是消息无限期地被延迟了。总之，就是系统中有些节点联系不上了，它们不能再参与决策，但也不代表它们过一段时间不能重新联系上。

为了表达上更直观，下面我们还是假设某些节点宕机了。那在这个时候，剩下的节点在缺少了某些节点参与决策的情况下，还能不能对于提议达成一致呢？即使是达成了一致，那么在那些宕机的节点重新恢复过来之后（注意这时候它们对于其它节点之间已经达成一致的提议可能一无所知），它们会不会对于已经达成的一致提议重新提出异议，从而造成混乱？所有这些问题，都是分布式一致性协议需要解决的。

我们这里没有足够的篇幅来详细讨论这些协议具体的实现了，感兴趣的读者可以去查阅相关的论文。实际上，**理解问题本身比理解问题的答案要重要的多**。总之，我们需要知道的是，我们已经有了一些现成的分布式一致性算法，它们能解决上面讨论的这些问题，保证在一个去中心化的网络中，各个节点之间最终能够对于提议达成一致。而且，这些协议都具有一定的容错性。一般来说，只要网络中的大部分节点(或一个quorum)仍然存活（即它们相互间可以收发消息），这个一致性的提议就可以达成。

拜占庭将军问题

我们前面讨论的一致性协议，有一个重要的前提条件，就是：各个节点都是可以信任的，它们都严格遵守同样的一套规则。这个条件，在一个公司的内部网络中可以认为是基本能满足的。但如果这个条件不满足会怎么样呢？假设网络中有些节点是恶意的，它们不但不遵守协议，还故意捣乱（比如胡乱发送消息），那么其它正常的节点还能够顺利工作吗？

在分布式系统理论中，这个问题被抽象成了一个著名的问题——拜占庭将军问题(Byzantine Generals Problem)。这个问题由大名鼎鼎的Leslie Lamport (http://www.lamport.org/)提出，也就是Paxos的作者。同时，Lamport还是2013年的图灵奖得主。

这要从一个故事开始说起（当然这个故事是Lamport编出来的）。拜占庭帝国的几支军队攻打到了敌人的城市外面，然后分开驻扎。每一支军队由一位拜占庭将军(Byzantine general)率领。为了制定出一个统一的作战计划，每一位将军需要通过信差(messenger)与其它将军互通消息。但是，在拜占庭将军之间可能出现了叛徒(traitor)。这些叛徒将军的目的是阻挠其他忠诚的将军(loyal generals)达成一致的作战计划。为了这一目的，他们可能做任何事，比如串通起来，故意传出虚假消息，或者不传出任何消息。

我们来看一个简单的例子。假设有5位将军，他们投票来决定是进攻还是撤退。其中两位认为应该进攻，还有两位认为应该撤退，这时候进攻和撤退的票数是2:2打平了。第五位将军恰好是个叛徒，他告诉前两位应该进攻，但告诉后两位应该撤退，结果前两位将军最终决定进攻，而后两位将军却决定撤退。没有达成一致的作战计划。

这个问题显然比我们在前一章讨论的可信任环境下的一致性问题要更难。要解决这个问题，我们是希望能找到一个算法，保证在存在叛徒阻挠的情况下，我们仍然能够达成如下目标：

- **A**. 所有忠诚的将军都得到了相同（一致）的作战计划。比如都决定进攻，或都决定撤退，而不是有些将军认为应该进攻，其他将军却决定撤退。
- **B**. 忠诚的将军不仅得到了相同的作战计划，还应该保证得到的作战计划是合理的(reasonable)。比如，本来进攻是更有利的作战计划，但由于叛徒的阻挠，最终却制定出了一起撤退的计划。这样我们的算法也算失败了。

可以看出，上面的目标A，是比较明确的，至少给定一个算法很容易判定它有没有达到这个目标。但目标B却让人无从下手。一个作战计划是不是「合理」的，本来就不好定义。即使没有叛徒的存在，忠诚的将军们也未必就一定能制定出合理的计划。这涉及到科学研究中一个非常重要的问题，如果一个事情不能用一种形式化的方式清晰的定义出来，对于它的研究也就无从谈起，这个事情本身也无法上升到科学的层面。Lamport在对拜占庭将军问题的研究中，一个突出的贡献就是，把这个看似不太好界定的问题，巧妙地归约到了一个能用数学语言精确描述的问题上去。下面我们就看一下这个过程是怎么做的。

首先我们考虑一下将军们制定作战计划的过程（先假设没有叛徒）。每一位将军根据自己对战局的观察，给出他建议的作战计划——是进攻还是撤退，这称为作战提议。然后，每位将军把自己的作战提议通过信差传达给其他每一位将军。现在每一位将军都知道了其他将军的作战提议，再加上他自己的作战提议，他需要根据所有这些信息得到最终的一个作战计划。为了表达上更清晰，我们给每位将军进行编号，分别是1, 2, …, n，每位将军提出的作战提议记为v(1), v(2), …, v(n)，一共是n个值，这其中有些代表「进攻」，有些代表「撤退」。经过信差传递消息之后，每位将军都看到了相同的作战提议序列v(1), v(2), …, v(n)，当然这其中的一个是当前这位将军自己提出来的。然后只要每位将军采用同样的方法，对所有的v(1), v(2), …, v(n)这些信息进行汇总，就能得到同样的最终作战计划。比如，容易想到的一个方法是投票法，即对v(1), v(2), …, v(n)中不同的作战提议进行投票，最后选择得票最多的作为最终作战计划。

当然，这样得到的最终作战计划也不能保证就是最好的，但这应该是我们能做到的最好的了。我们现在仍然假设将军里没有叛徒。我们发现，前面提到的目标A和目标B的要求可以适当「降低」一些：我们不再关注将军们是否能达成最终一致的作战计划，并且这个计划是不是「合理」；我们只关注每个将军是否收到了完全相同的作战提议v(1), v(2), …, v(n)。只要每位将军收到的这些作战提议是完全相同的，他们再用同样的方法进行汇总，就很容易得到最终一致的作战计划。至于这个最终的作战计划是不是最好的，那就跟很多「人为」的因素有关了，我们不去管它。

现在我们考虑将军中出现了叛徒。遵循前面的思路，我们仍然希望每位将军能够收到完全相同的作战提议v(1), v(2), …, v(n)。现在我们仔细审视一下其中的一个值，v(i)，在前面的描述中，它表示来自第i个将军的作战提议。如果第i个将军是忠诚的，那么这个定义没有什么问题。但是，如果第i个将军是叛徒，那么就有问题了。为什么呢？因为叛徒可以为所欲为，他为了扰乱整个作战计划的制定，完全可能向不同的将军给出不同的作战提议。这样的话，不同的忠诚将军收到的来自第i个将军的v(i)可能是不同的值。这样v(i)这个定义就不对了，它需要改一改。

不管怎么样，即使存在叛徒，我们还是希望每位将军最终是基于完全相同的作战提议来做汇总，这些作战提议仍然记为v(1), v(2), …, v(n)。不过，这里的v(i)不再表示来自第i个将军的作战提议，而是表示经过我们设计的某个一致性算法处理之后，每位将军最终看到的第i个提议。这里需要分两种情况讨论。首先第一种情况，如果第i个将军是忠诚的，那么我们自然希望这个v(i)就是第i个将军发送出来的作战提议。换句话说，我们希望经过一致性算法处理之后，第i个将军如果是忠诚的，那么它的提议能够被如实地传达给其他将军，而不会被叛徒的行为所干扰。这是可能制定出「合理」作战计划的前提。第二种情况，如果第i个将军是叛徒，那么他有可能向不同的将军发送不同的提议。这时候我们不能够只听他的一面之词，而是希望经过一致性算法处理之后，各个将军之间充分交换意见，然后根据其他各个将军转述的信息，综合判断得到一个v(i)。这个v(i)是进攻还是撤退，并不太重要，关键是要保证每位将军得到的v(i)是相同的。只有这样，各位将军经过汇总所有的v(1), v(2), …, v(n)之后才能得到最终的完全一致的作战计划。

根据上面的分析，我们发现，在这两种情况中，我们都只需要关注单个将军（也就是第i个将军）所发出的提议如何传达给其他将军。重点终于来了！至此，我们就能够把原来的问题归约到一个子问题上。这个子问题，才是Leslie Lamport (http://www.lamport.org/)在他的论文中被真正命名为「拜占庭将军问题(Byzantine Generals Problem)」的那个问题。在这个问题中，我们只关注发送命令的单个将军，称他为主将(commanding general)，而其他接受命令的将军称为副官(lieutenant)。下面是「拜占庭将军问题」的精确描述。

一个主将发送命令给n-1个副官，如何才能确保下面两个条件：

- (IC1) 所有忠诚的副官最终都接受相同的命令。
- (IC2) 如果主将是忠诚的，那么所有忠诚的副官都接受主将发出的命令。

这其实正好对应了我们前面已经讨论过的两种情况。如果主将是忠诚的，那么条件IC2保证了命令如实地传递，这时候条件IC1自然也满足了；如果主将是叛徒，那么条件IC2没有意义了，而条件IC1保证了，即使叛徒主将对每个副官发出不同的命令，每个副官仍然能最终获得一致的命令。

这里有两个地方可能让人产生疑惑。

第一，有些人会问了，难道主将还能是叛徒？主将都是叛徒了，还有啥搞头啊？其实是这样的，这个「拜占庭将军问题」只是原问题的一个子问题。当n个将军通过传递消息来决策作战计划的时候，可以分解成n个「拜占庭将军问题」，即分别以每位将军作为主将，以其余n-1位将军作为副官。如果有一个算法能够解决「拜占庭将军问题」，那么同时运行n个算法实例，就能使得每位将军都获得完全相同的作战提议序列，即前面我们提到的v(1), v(2), …, v(n)。最后，每位将军将v(1), v(2), …, v(n)使用相同的方法进行汇总（比如按多数投票），就能得到最终的作战计划。

第二，当主将是叛徒的时候，他可以向不同的副官发送不同的命令，怎么可能每个副官仍然能最终获得一致的命令呢？这正是算法需要解决的。其实这也容易解释（我们前面也提到过这个思路），由于主将可能向不同的副官发送不同的命令，所以副官不能直接采用主将发来的命令，而是也要看看其他副官转述来的主将的命令是什么。然后，一个副官综合了由所有副官转述的命令（再加上主将直接发来的命令）之后，就可能得到比较全面的信息，从而做出一致的判断（在实际中是个不断迭代的过程）。

好了，我们用了这么多篇幅，终于把「拜占庭将军问题」本身描述清楚了。这实际上也是最难的部分。我们上一章提到过，理解问题本身比理解问题的答案更重要。只要问题本身分析清楚了，如何设计一个能解决它的算法就只是细节问题了。我们这里不深入算法的细节了，感兴趣的读者可以去查阅下列论文：

- 《The Byzantine Generals Problem》，下载地址：http://lamport.azurewebsites.net/pubs/byz.pdf (http://lamport.azurewebsites.net/pubs/byz.pdf)

- 《Reaching Agreement in the Presence of Faults》，下载地址：http://lamport.azurewebsites.net/pubs/reaching.pdf (http://lamport.azurewebsites.net/pubs/reaching.pdf)

我们这里只提一下论文给出的算法的结论。

使用不同的消息模型，「拜占庭将军问题」有不同的解法。

- 如果将军之间使用口头消息(oral messages)，也就是说，消息被转述的时候是可能被篡改的，那么要对付m个叛徒，需要至少有3m+1个将军（其中至少2m+1个将军是忠诚的）。
- 如果将军之间使用签名消息(signed messages)，也就是说，消息被发出来之后是无法伪造的，只要被篡改就会被发现，那么对付m个叛徒，只需要至少m+2个将军，也就是说至少2个忠诚的将军（如果只有1个忠诚的将军，显然这个问题没有意义）。这种情况实际相当于对忠诚将军的数目没有限制。

容错性(fault tolerance)

我们前面提到过，以Paxos为代表的分布式一致性协议，是在可信任的环境下运行的。而在「拜占庭将军问题」中，网络中则存在恶意节点。因此我们很容易产生一个想法：Paxos是不是「拜占庭将军问题」在叛徒数为零时的一个特例解？

这样看其实有点问题。在「拜占庭将军问题」中，除了叛徒，剩下的是忠诚的将军。「忠诚」这个词，其实暗含了一个意思：他是能够正常工作的（即你可以随时通过消息跟他进行交互）。为什么这么说呢？我们知道，一个叛徒可以做任何事，包括发送错误消息，也包括不发送任何消息。「不发送任何消息」，相当于不能正常工作，或者说，发生了某种故障。所以，不仅仅是故意的恶意行为，即使是单纯的故障，也应该能归入叛徒的行为。这在其他将军看来没有区别。

按照这种理解，「忠诚」这个词并不是很恰当。叛徒数为零，相当于网络中每个节点都在正常工作。但是Paxos的设计也是能够容错的，就像我们在前面讨论的一样，网络中的少数节点发生故障（比如宕机），Paxos仍然能正常工作。可见，Paxos并不能看成是「拜占庭将军问题」在叛徒数为零时的一个特例解。

那「拜占庭将军问题」和Paxos这类分布式一致性算法的关系应该如何看待呢？我们可以从容错性的强弱程度上来分析。

一般来说，设计一个计算机系统，小到一块芯片，大到一个分布式网络，都需要考虑一定的容错性(fault tolerance (https://en.wikipedia.org/wiki/Fault_tolerance))。但根据错误不同的性质，可以分为两大类：

- 拜占庭错误(Byzantine fault)。这种错误，在不同的观察者看来，会有前后不一致的表现。
- 非拜占庭错误(non-Byzantine fault)。从字面意思看，是指那些不属于前一类错误的其它错误。

这两类错误的含义并没有字面上那么好理解。

先说说拜占庭错误。在「拜占庭将军问题」中，叛徒的恶意行为固然是属于这一类错误的。在不同的将军看来，叛徒可能发送完全不一致的作战提议。而在计算机系统中，出现故障的节点或部件也可能表现出前后不一致的行为，虽然这并非恶意，但也属于这一类错误。比如信道不稳定，导致节点发送给其它节点的消息发生了随机错误，或者说，消息损坏了(corrupted)。再比如，在数据库系统中，commit之后的数据明明已经同步给磁盘了（通过操作系统的fsync），但由于突然断电等原因，最终数据还是没有真正落盘成功，甚至出现数据错乱。

再看一下非拜占庭错误。Lamport在他关于Paxos的一篇论文中也使用了non-Byzantine这个词（见《Paxos Made Simple (http://lamport.azurewebsites.net/pubs/paxos-simple.pdf)》）。但是这个词的命名的确让人有点不好理解。在分布式系统中，如果节点宕机了，或者网络不通了，都会导致某些节点不能工作。其它节点其实没法区分这两种情况，在它看来，只是发现某个节点暂时联系不上了（即接收消息超时了）。至于是因为那个节点本身出问题了，还是网络不通了，或者是消息出现了严重的延迟，是无法区分的。而且，过一会之后，节点可能会重新恢复（或是自己恢复了，或经过了人工干预）。换句话说，对于出现这种错误的节点，我们只是收不到它的消息了，而不会收到来自它的错误消息。相反，只要收到了来自它的消息，那么消息本身是「忠实」的。

可见，拜占庭错误是更强的一类错误。在「拜占庭将军问题」中，叛徒发送前后不一致的作战提议，属于拜占庭错误；而不发送任何消息，属于非拜占庭错误。所以，解决「拜占庭将军问题」的算法，既能处理拜占庭错误，又能处理非拜占庭错误。这听起来稍微有些奇怪，不过这只是命名带来的问题。

总之，「拜占庭将军问题」的解法应该是最强的一类分布式一致性算法，它理论上能够处理任何错误。而Paxos只能处理非拜占庭错误。通常把能够处理拜占庭错误的这种容错性称为「Byzantine fault tolerance (https://en.wikipedia.org/wiki/Byzantine_fault_tolerance)」，简称为BFT。

这样说来，BFT的算法应该可以解决任何错误下的分布式一致性问题，也包括Paxos所解决的问题。那为什么不统一使用BFT的算法来解决所有的分布式一致性问题呢？为什么还需要再费力气设计Paxos之类的一些算法呢？我们前面没有仔细讨论解决「拜占庭将军问题」的算法，所以这里也不做仔细的分析了。但容易想象的是，提供BFT这么强的错误容忍性，肯定需要付出很高的代价。比如需要消息的大量传递。而Paxos不需要提供那么强的容错性，因此可以比较高效地运行。另外，具体到Lamport在论文中给出的解决「拜占庭将军问题」的算法，它还对系统的记时假设(timing assumption)有更强的要求。这也容易理解，既然算法的容错性要求这么高，自然对于运行环境的假设(assumption)也有可能要高一点。由于这个问题是分布式系统中一个挺关键的问题，所以我们在这里单独拿出来讨论一下。在Lamport在论文中，算法对于系统的假设有这么一条：

- The absence of a message can be detected.

这条假设要求，如果某位叛徒将军没有发送任何消息（当然也可能是消息丢失了），那么这件事是可以检测出来的。显然，这只能依赖某种超时机制(time-out)，依赖节点之间的时钟达到一定程度的同步，即时钟的偏移不能超过一个最大值。这实际上是一种同步模型(synchronous model)。而Paxos的系统假设在这一点上就没有这么强，它是基于异步模型(asynchronous model)，对系统时钟没有特定的要求。我在之前的另一篇文章《基于Redis的分布式锁到底安全吗？(/posts/blog-redlock-reasoning.html)》一文中也有提到过这个问题。这有时候会成为一些分布式算法产生争议的根源。

具体来说，根据Paxos的论文所说，Paxos的设计是基于异步(asynchronous)、非拜占庭(non-Byzantine)的系统模型，即：

- 节点可以以任意速度运行，可能宕机、重启。但是，算法执行过程中需要记录的一些变量，在重启后应该能够恢复。
- 消息可以延迟任意长时间，可以重复(duplicated)，可以丢失(lost)，但不能损坏(corrupted)。

上面第一条其实是要求数据在数据库中持久化，并且要保证在落盘过程中没有发生拜占庭错误（我们前面刚提到过）。但实际中由于突然断电、磁盘缓存等现实问题，拜占庭错误是有可能发生的（虽然概率很低），所以这就要求工程上做一些特殊处理。

上面第二条，消息损坏，属于拜占庭错误。所以Paxos要求不能有消息损坏发生。这在使用TCP协议进行消息传输的情况下，可以认为是能够满足要求的。

综上分析，解决「拜占庭将军问题」的算法，提供了最强的容错性，即BFT，而Paxos只能容忍非拜占庭错误。但是，在只有非拜占庭错误出现的前提下，Paxos基于异步模型，是比同步模型更弱的系统假设，因此算法更鲁棒。当然，Paxos也更高效。

区块链

在现实中，真正需要达到BFT容错性的系统很少，除非是一些容错性要求非常高的系统，比如波音飞机上的控制系统，或者SpaceX Dragon太空船这类系统（参见https://www.weusecoins.com/bitcoin-byzantine-generals-problem/ (https://www.weusecoins.com/bitcoin-byzantine-generals-problem/)）。

我们平常能接触到的BFT的一个典型的例子，就是区块链了。一个区块链网络是一个完全开放的网络，其中的矿工节点(miner)是可以自由加入和自由退出的。这些节点当然有可能是恶意的，所以区块链网络在设计的时候必须要考虑这个问题。这实际上就是典型的「拜占庭将军问题」。

接下来为了将区块链与「拜占庭将军问题」之间的联系讨论得更加清楚，我们先来非常粗略地介绍一下区块链技术。

以比特币网络为例，它的核心操作就是进行比特币交易，即某个比特币的拥有者将自己一定数量的比特币转移给其他人。首先，比特币的拥有者要发起交易(transaction)，他需要先用自己的私钥对交易进行签名，然后将交易请求发给矿工节点。矿工将收到的所有交易打包到一个区块(block)当中，并通过一系列复杂度很高的运算找到一个nonce值，保证对于它和区块内其它信息进行hash计算后的结果能够符合预定的要求。这一步对于整个区块链网络至关重要，被称为工作量证明(Proof of Work)。然后矿工把该区块在全网发布，由其它矿工来验证这个区块。这个验证既包括对交易签名进行验证（使用比特币拥有者的公钥），也包括对工作量证明的有效性进行验证。如果验证通过，就把这个区块挂在当前最长的区块链上。

如果两个矿工几乎同时完成了区块的打包和工作量证明，它们可能都会将区块进行发布，这时区块链就会分叉(fork)。但矿工们会不停地产生新区块，并将新区块挂在当前最长的区块链上，所以最终哪个分叉变得更长，哪个分叉就会被多数矿工节点承认。这么看来，区块链其实不是一个链，而是一棵树。我们知道，在树这种数据结构中，从根节点到叶子节点只有唯一的一条路径。因此，当前有效的区块链其实是这棵树中从根节点到叶子节点最长的那条路径。只要一个区块在最长的链上，那么它就是有效的，它里面包含的所有交易就被固化下来了（被多数节点承认）。

我们只是非常粗略地介绍了一下区块链的工作原理。如果想了解细节，建议研究一下以太坊的官方wiki，地址是：

> https://github.com/ethereum/wiki/wiki (https://github.com/ethereum/wiki/wiki)

下面我们开始讨论区块链技术与「拜占庭将军问题」的关联。

前面我们讨论「拜占庭将军问题」的时候，得到过以下结论：

- 如果使用口头消息，那么至少需要多于2/3的将军是忠诚的。
- 如果使用签名消息，那么对忠诚将军的数量是没有要求的。

根据前面的介绍，我们在区块链中使用的消息应该属于签名消息。具体体现在：每一个区块中的交易都进行了签名，保证无法被篡改，也能保证这个消息只能是由最初的发起者发出的。那么，这属于上述第二种情况，难道说区块链网络中忠诚节点的数目没有要求？显然不是这样。比如在比特币网络中，要求恶意节点不能掌握多于50%的算力。为什么两者之间似乎不一致呢？这是因为，「拜占庭将军问题」只是关注一个子问题，它关注的是其中一个将军（称为主将）向其他所有将军（称为副官）发送命令的情况。而最终对所有命令进行汇总则要求所有忠诚的将军达成共识。如果忠诚的将军数目太少，不管最终确定的作战计划是什么，还是会失败，因为叛徒可能不执行这个作战计划。这类似于比特币网络中的情况，其中对于最长链的选择过程，就相当于将军们对所有命令进行汇总的操作（按多数投票）。

在「拜占庭将军问题」中，一个叛徒可能向不同的将军发送不一致的命令。如果算法设计得不好，就可能造成最终无法达成一致。在区块链网络中，类似的行为将会成本很高。这是由于矿工节点发布区块的消息必须经过工作量证明，它如果发布不一致的区块，每个区块都需要工作量证明，这将耗费它大量的算力。另外，这样做也没有动机，它只会产生更多分叉，不会产生最长链。

在「拜占庭将军问题」的框架下，如何看待工作量证明呢？它其实相当于提高了做叛徒的成本，从而极大降低了叛徒超过半数的可能性。这里可以做一个对比，假设历史上存在真实的拜占庭将军问题，那么可以想象，敌军的间谍打入拜占庭将军这个群体中的成本应该是很高的。所以，可以认为将军中的叛徒不至于太多。但对应到计算机网络中，如果没有类似工作量证明的机制，那么成为叛徒矿工的成本就是非常低的。这就很有可能使得叛徒比忠诚的矿工还要更多。

当然，从经济学的角度看，在需要工作量证明的前提下，成为叛徒矿工也是不明智的。因为它既然拥有比较强的算力，还不如按照合理的方式通过挖矿赚取收益更为稳妥。不过这是技术之外的因素了。

除了工作量证明这种机制(Proof of Work)之外，还有一种被称为Proof of Stake的机制。虽然有人质疑这种机制存在缺点(比如nothing at stake (https://github.com/ethereum/wiki/wiki/Problems))，但站在「拜占庭将军问题」的角度，它也是相当于提高了做叛徒的成本。这就好比一个间谍要混入董事会，成本肯定是比较高的，因为他需要首先持有大量股票。

区块链到底是什么？有人说是个无法篡改的超级账本，也有人说是个去中心化的交易系统，还有人说它是构建数字货币的底层工具。但是，从技术的角度来说，它首先是个**解决了拜占庭将军问题的分布式网络**，在完全开放的环境中，实现了数据的一致性和安全性。而其它的属性，都附着于这一技术本质之上。

在今天这篇文章中，我们从分布式系统的一致性问题开始谈起，直到「拜占庭将军问题」，最后又跟区块链产生了关联。这几部分逐步深入，虽是「漫谈」，逻辑上却是前后贯穿的。

最后，区块链，是不是一项伟大的革新？是。特别是比特币系统，它是分布式系统技术与金融系统业务相结合造就的一次成功的创举。

区块链技术到底会不会颠覆未来呢？我们现在只能说：有可能。

（完）

其它精选文章：

- 基于Redis的分布式锁到底安全吗（下） (https://mp.weixin.qq.com/s?__biz=MzA4NTg1MjM0Mg==&mid=2657261521&idx=1&sn=7bbb80c8fe4f9dff7cd6a8883cc8fc0a&chksm=84479e08b330171e89732ec1460258a85afe732
- Redis为什么用跳表而不用平衡树？ (https://mp.weixin.qq.com/s?__biz=MzA4NTg1MjM0Mg==&mid=2657261425&idx=1&sn=d840079ea35875a8c8e02d9b3e44cf95#rd)
- 为什么未来是增强现实的？ (https://mp.weixin.qq.com/s?__biz=MzA4NTg1MjM0Mg==&mid=2657261622&idx=1&sn=a917c7124f087ec4ea20cc1434c6ab06&chksm=844791efb33018f915ff50a3583628ba8469ea
- 三个字节的历险 (https://mp.weixin.qq.com/s?__biz=MzA4NTg1MjM0Mg==&mid=2657261541&idx=1&sn=2f1ea200389d82e7340a5b4103968d7f&chksm=84479e3cb330172a6b2285d4199822143ad0
- 做技术的五比一原则 (https://mp.weixin.qq.com/s?__biz=MzA4NTg1MjM0Mg==&mid=2657261555&idx=1&sn=3662a2635ecf6f67185abfd697b1057c&chksm=84479e2ab330173cebe16826942b034daec79
- 知识的三个层次 (https://mp.weixin.qq.com/s?__biz=MzA4NTg1MjM0Mg==&mid=2657261491&idx=1&sn=cff9bcc4d4cc8c5e642309f7ac1dd5b3&chksm=84479e6ab330177c51bbf8178edc0a6f0a1d56
- 技术的正宗与野路子 (https://mp.weixin.qq.com/s?__biz=MzA4NTg1MjM0Mg==&mid=2657261357&idx=1&sn=ebb11a1623e00ca8e6ad55c9ad6b2547#rd)
- 技术攻关：从零到精通 (https://mp.weixin.qq.com/s?__biz=MzA4NTg1MjM0Mg==&mid=2657261530&idx=1&sn=6e2e80a0895325861541c2b4266ae374&chksm=84479e03b3301715c53f0eebff06f6eca7d4a

原创文章，转载请注明出处，并包含下面的二维码！否则拒绝转载！
本文链接：http://zhangtielei.com/posts/blog-consensus-byzantine-and-blockchain.html (http://zhangtielei.com/posts/blog-consensus-byzantine-and-blockchain.html)

上篇： 为什么未来是增强现实的？ (/posts/blog-talk-about-ar-future.html)　　下篇： 一个错别字100块钱？ (/posts/blog-no-typo-and-error.html)

栏目分类

最新文章