分布式共识的工作原理

原文链接: https://medium.com/s/story/lets-take-a-crack-at-understanding-distributed-consensus-dad23d0dc95
作者: Preethi Kasireddy
翻译&校对: Ray, IAN LIU, Stormpang, 安仔 Clint & 阿剑, Elisa @ EthFans.org

分布式系统经常让人觉得云山雾罩，**主要是因为相关知识点比较零散不成体系。**但别担心，我很清楚这个有点尴尬的情况。我自学分布式计算的时候，就有很多次完全摸不着头脑。现在，经历过很多次尝试和死磕之后，我终于有信心可以跟大家分享一下有关分布式系统的基础知识了。

我还想讨论一下区块链技术给分布式相关的学术界和工业界已然带来的深远影响。**区块链促使工程师和科学家们重新审视和反思那些在分布式计算领域已经根深蒂固的范式。**也许在对这个领域的研究发展推动作用上，没有什么比区块链技术更强有力的了。

分布式系统肯定不能算新技术了。科学家和工程师们过去数十年中一直在研究这个课题。那么区块链跟分布式系统有什么关系呢？简单地说，如果没有分布式系统，也就不可能有区块链带来的技术贡献。

本质上，一条区块链就是一种新型的分布式系统。区块链自起源于比特币以来就一直对分布式计算领域持续产生影响。所以想真正搞明白区块链的运行原理，深入理解分布式系统的原理就至关重要。

但不幸的是，大部分关于分布式计算的文献资料，要么太晦涩艰深，要么散布在不计其数的学术论文中。**更麻烦的是，分布式系统为了满足不同的需求，可能有多达上百种不同的架构。要把这些都归纳成一个统一且易于理解的框架的确很困难。**

因为这个领域涉及知识点太广，我必须很专注在我力所能及可以讲明白的部分。而且我需要一些概括性的描述以便对系统复杂性进行简化。请注意，我不会帮你成为这个领域的专家。但我会帮你快速入门分布式系统及其共识机制。

读完这篇长文之后，你应该会对以下内容有更深刻的认识：

- 什么是分布式系统，
- 分布式系统的特性，
- 分布式系统中的共识意味着什么，
- 基础共识算法（比如 DLS 和 PBFT）以及
- 为什么中本聪共识非常重要。

嗯，不作过多解释了，快上车。

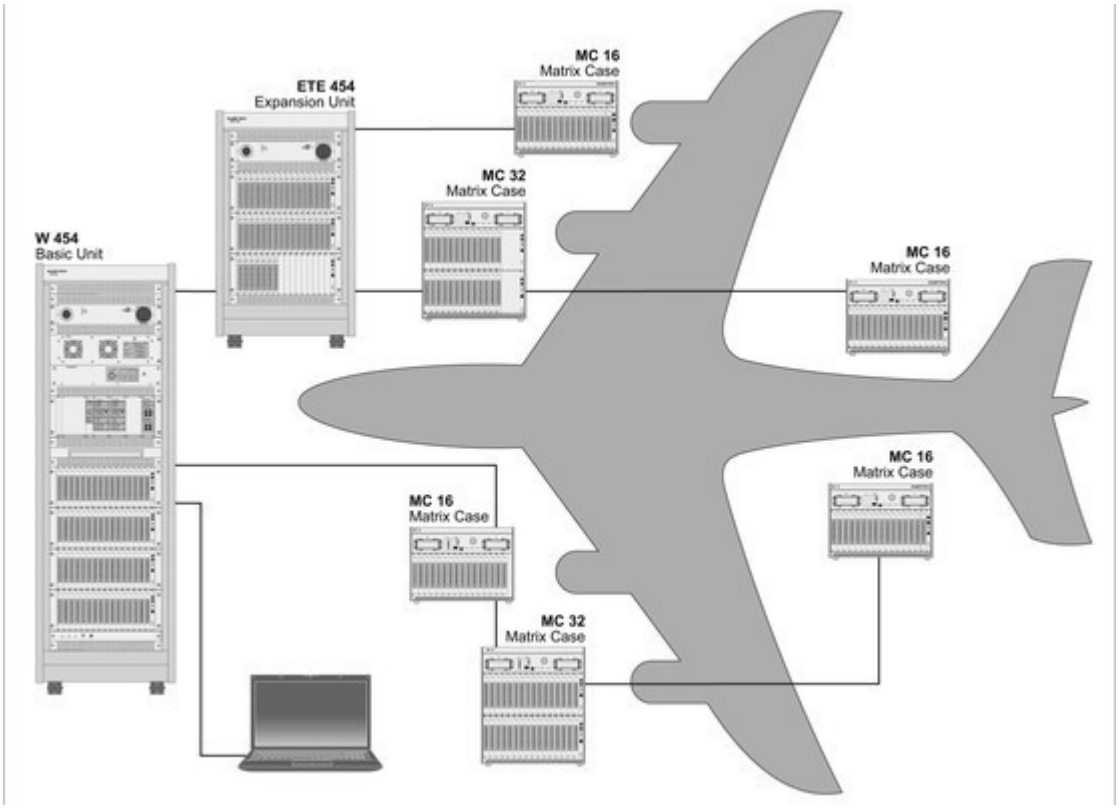
什么是分布式系统?

一个分布式系统包括一组相互独立的进程（比如计算机），它们互相传递消息并进行协作以完成一个共同的任务（比如解决一个计算问题）。

简单的说，一个分布式系统就是一组计算机，它们互相协作以完成一个共同的任务。虽然组成分布式系统的进程是相互独立的，但整个系统对于终端用户而言可以看成只是一台计算机。

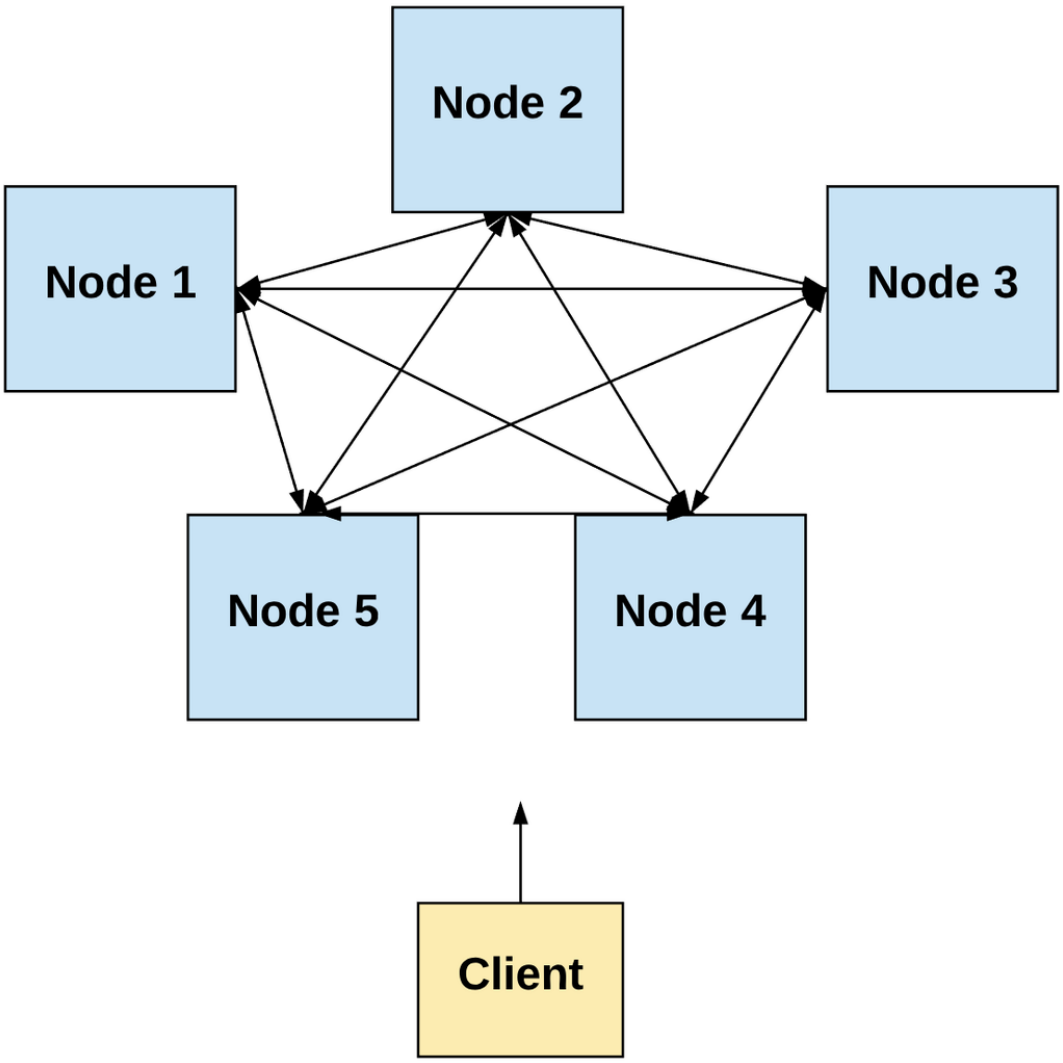
像我之前提到的，分布式系统可以有上百种架构。比如，一台普通的计算机也可以看成是一个分布式系统：中央处理器，内存，输入输出都是独立的进程，互相协作以完成共同的任务。

又比如飞机，下图中的不同组件协同工作，载你从A点飞到B点：



-图片来源：WEETECH -

在这篇文章中，我们将重点看看那些由空间上隔离的计算机组成的分布式系统。



-作者制图-

请注意：我可能会使用若干术语表达与""进程"相同的含义："节点"（node、peer），"计算机"（computer），"组件"（component）。在这篇文章里它们都是同义词。类似地，"网络"（network）和"系统"（system）在本文中也是同义词。

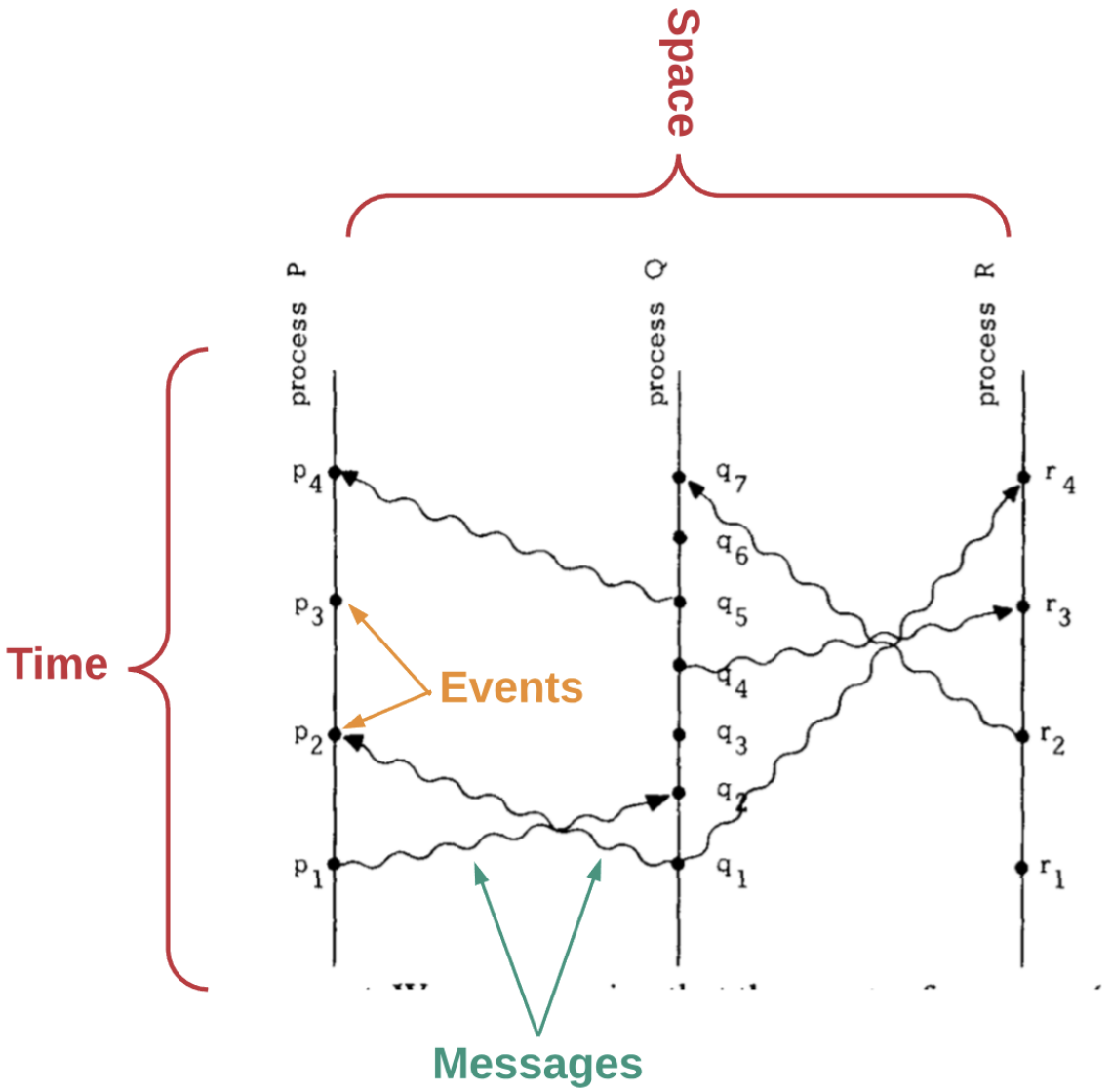
分布式系统的特性

每个分布式系统都有一组特性，如下：

A) 并发性

进程在系统中是并发运行的，即同一时刻有多个事件发生。换句话说，任一时刻，系统中每个节点的事件执行都是相互独立的。

这就需要协作。



-分布式系统中的时间，时钟和事件排序，来自 Lamport, L (1978)-

B) 全局时钟的缺失

分布式系统要能运转，我们就需要一个确定事件排序的方法。但是，由于分布式系统中的计算机是空间上隔离且并发运行的，有时候很难说清楚两个事件中哪个先发生。换句话说，对于系统中所有计算机而言，没有一个统一的全局时钟来决定事件发生的顺序。

Leslie Lamport 在他的论文《分布式系统中的时间，时钟和事件排序》里，展示了如何通过以下事实来决定事件的先后顺序：

1. 消息总是先发送，而后被接收。
2. 每台计算机都有事件的排序。

通过确定事件之间的相对顺序（哪个先发生，哪个后发生），我们就可以得到关于系统中事件的一个局部排序（partial ordering）。Lamport 的论文描述了一种算法，要求每台计算机都监听其它计算机。这样，所有事件就可以根据这个局部排序得到全局排序。

但是，如果完全基于每个独立计算机监听到的事件进行排序，有可能会碰到这种情况：得出的排序与外部用户的观察不一致。所以，论文提到该算法仍然无法完全排除异常情况。最后，Lamport 讨论了如何通过适当地同步物理时钟避免上述异常。

但是，等一下——这里有个很重要的前提：**协调独立的时钟并使之同步是一个非常复杂的计算机科学问题。即使你一开始把一堆时钟调成同步的状态，一段时间后它们的计时也会开始产生偏差。**这被称作"时钟偏移"，是一种时钟计时频率发生细微差别的现象。

实质上，Lamport 的论文证明了：在由空间上隔离的一组计算机组成的分布式系统中，时间和事件排序是导致系统可能出错的障碍根源。

C) 单点可能出错

理解分布式系统的一个关键点在于认识到，分布式系统中的组件是会出错的。这也是分布式系统被称作"容错分布式计算"的原因。**不出错的系统是不可能存在的。现实当中的系统总是暴露在许多可能的错误和故障风险之下**，这些故障包括程序崩溃；消息丢失、实真、重放；网络隔离导致的消息延迟或丢帧；甚至是进程完全失控、恶意发送消息。

错误可以被大致分为三类：

- **崩溃错误**：组件没有预警就停止工作（比如计算机崩溃了）。
- **遗漏错误**：组件发了一个消息但没有被别的节点收到（比如这条消息丢包了）。
- **拜占庭错误**：组件不按既定规则工作。这种类型的错误在可控环境中不会发生（比如谷歌和亚马逊的数据中心），因为一般认为那种环境中系统里不会发生恶意行为。与之相反，拜占庭错误会发生在所谓的"敌对环境"中。简单来说，当一组去中心化的独立参与者作为节点加入某个分布式网络后，这些参与者可以完全不按既定规则行动，也就是说它们可以恶意地更改消息、拦截消息或者根本不发送任何消息。

了解上述概念后，那分布式系统的目标就可以定义为：**为一个存在可能出错组件的系统设计协议，期望该系统仍然能完成组件的共同任务并对用户提供可用的服务。**

考虑到所有的系统都可能会出错，那我们建立一个分布式系统的核心考量就是，它是否可以在部分组件发生异常（无论是因为非恶意的行为比如崩溃-出错或者遗漏错误，还是因为恶意行为比如拜占庭错误）的情况下继续工作。

粗略地说，建立分布式系统有两种可考虑的模式：

1) 简单容错

在一个简单容错系统中，我们假设系统中所有组件都会处于以下两种状态中的一个：要么严格遵循协议工作，要么不遵循。这种类型的系统可以处理节点下线或者宕机的情况。但是它没法处理不按既定规则工作或者恶意工作的节点。

2A) 拜占庭容错

简单容错系统在非可控环境中用处不大。在一个节点由独立参与者控制，且通过无需权限的开放互联网进行通信的去中心化网络里，我们需要考虑到，网络中可能出现恶意节点（或者说"拜占庭式"节点）。因此，在拜占庭容错系统中，我们假设节点可能会宕机或者作恶。

2B) BAR容错

尽管大多数现实系统都被设计成可以容忍拜占庭错误，有些专家认为，这些设计太过宽泛，并没有考虑到所谓的"利己性"出错，即节点可能因为出于合理的自身利益而采取恶意行为。换句话说，根据不同的激励，节点可能诚实，也可能不诚实。所以如果激励足够多，甚至可能大部分节点都会不诚实。

更正式的定义一下，这就是 BAR 模式-同时考虑了拜占庭出错和利己性出错的情况。BAR 模式假设系统中有下列三种参与者：

- **拜占庭**：这种节点是恶意的并且就是试图让你玩完。
- **利他**：始终遵循系统协议的诚实节点。
- **利己**：这类节点只在遵循协议有利于自身时才会遵循。

D) 消息传递

如我前述，分布式系统总的计算机通过互相之间的“消息传递”来沟通和协作。消息可以通过任何一种消息协议传递，比如 HTTP 或者 RPC，亦或是为特定系统实现的一套定制协议。有两种消息传递的环境：

1) 同步

在同步系统中，我们假定消息会在一个固定且已知的时间范围内接收到。

同步消息传递在概念上更简单，因为用户会有一个时间上的保证：当他们发送某条消息后，接收方会在一定时间内收到它。这就使得用户可以为他们的协议设置一个消息传递的耗时上限。

但是，这种假设在现实生活当中的分布式系统中不切实际：因为组件计算机们可能会崩溃，或者下线，而且消息也可能丢包，重复，延时或者没有按照发送顺序被接收。

2) 异步

在异步消息传递系统中，我们假定系统可能导致某条消息一直被延误，导致消息重放，或者乱序发送消息。换句话说，在这种环境下，消息传递的预期耗时是没有上限的。

https://ethfans.org/posts/a-7-000-year-history-of-trust-with-mits-michael-casey)

在分布式系统中，达成共识意味着什么

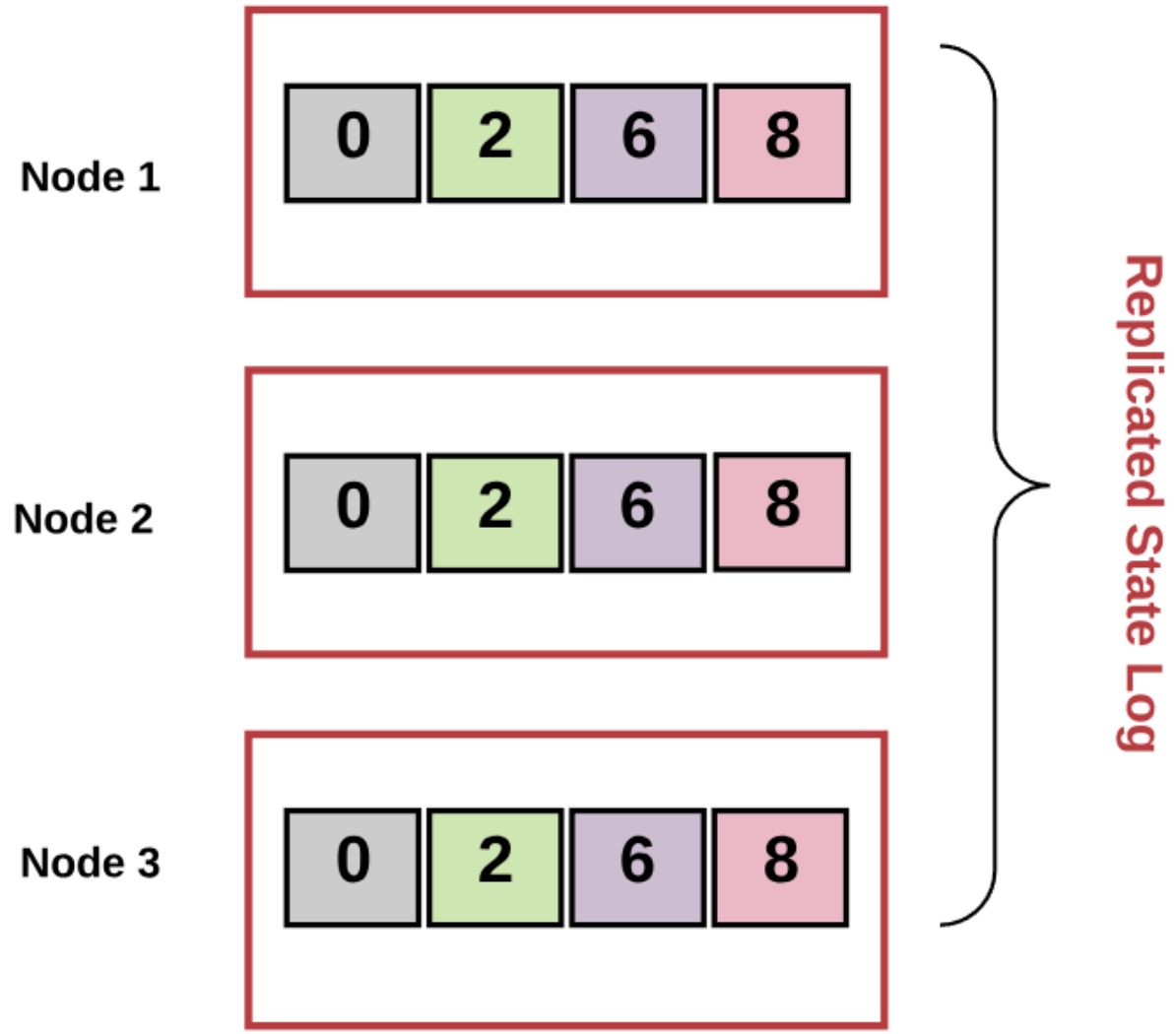
目前，我们已经知道分布式系统有以下特性：

- 进程的并发性
- 全局时钟缺失
- 组件可能出错
- 消息需时传递

接下来我们会聚焦分布式系统中，“达成共识”究竟是什么意思。首先有一件很重要的事情得重申一下，“分布式计算有数百种软硬件架构”；而最常见的形式称为复制状态机。

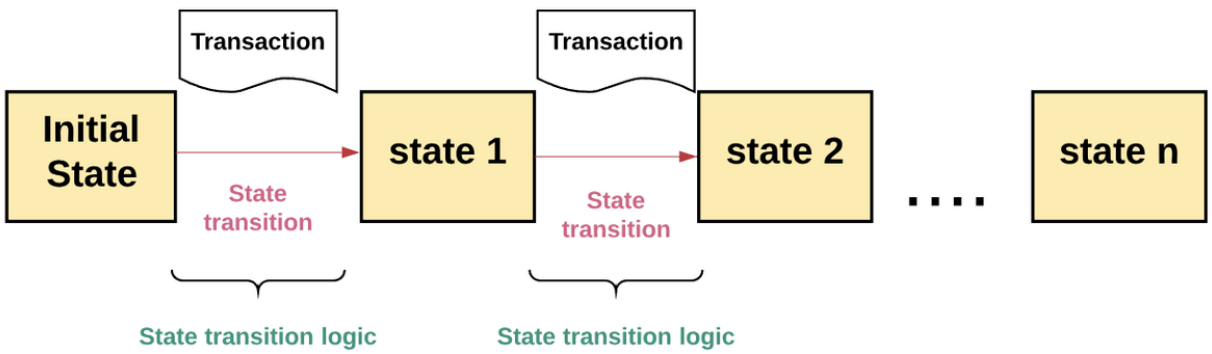
复制状态机

复制状态机是一种确定性状态机，它在许多计算机上存有副本，但整个系统就像单个状态机那样运行。任何一台计算机都有出错的可能，但状态机依然能正常运行。



-作者制图-

在复制状态机中，如果出现一个有效事务（transaction），则事务的输入集会使得系统的状态转变为下一个状态。**事务对数据库进行原子性的操作，这意味着操作要么整个完成，要么等于完全没发生。**在复制状态机内维护的事务集合又称为"事务日志"；从一个有效状态转变为下一个有效状态的逻辑，称为"状态转变逻辑"。



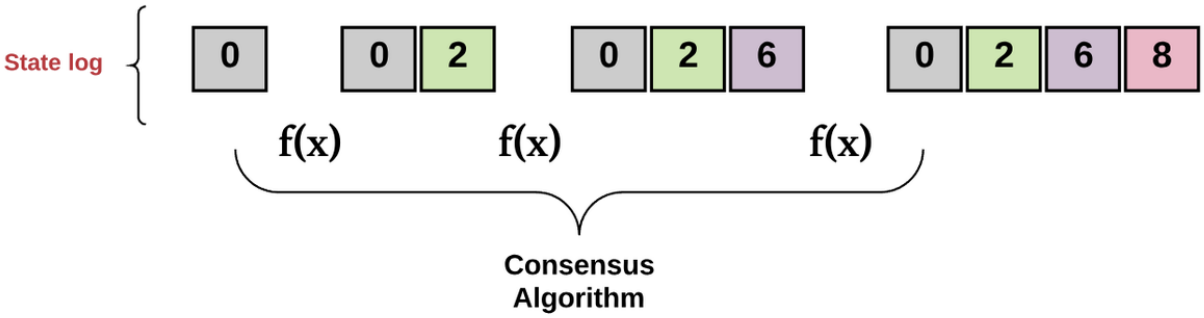
-作者制图-

换言之，复制状态机是一个分布式计算机的集合，这些计算机都有相同初值。每一次状态的转变方式、下个状态是什么，都由相关的进程决定。**所谓“达成共识”，意思是全体计算机都同意某个输出的值。**

反过来说，“达成共识”也意味着让系统中每一台计算机的事务日志保持一致（也就是它们“达成了共同目标”）。合格的复制状态机，即使发生以下情况，仍必须不断将新的事务接收进日志（即“提供有效服务”）：

1. 有些计算机发生故障。
2. 网络不可靠，消息可能出现延迟、传送失败，或排序混乱。
3. 没有全局时钟来确定事务顺序。

朋友们，这就是所有共识算法的基本目标。



-作者制图-

共识问题的定义

只要能满足以下条件，我们就说某算法可以实现分布式共识：

1. **Agreement**（一致性）：所有非故障节点，都会选择相同的输出值。
2. **Termination**（可终止性）：所有非故障节点，最终都选定了某些输出值，不再反悔。

注意：不同的算法会满足上述条件的不同变体。举例来说，某些算法会将 Agreement 拆分为一致性和总体性；一些算法还加入了有效性（validity）、完整性（integrity），或效率的概念。这些细微的区别不在本文讨论范围。

广义来说，共识算法一般会假设系统中有三种角色：

1. **提案者（Proposers）**：常被称为领导者或协调者。
2. **接收者（Acceptors）**：进程，监听提案者的请求并给出响应值。
3. **学习者（Learners）**：系统里的其他进程，接收最终决定的值。

因此，正常情况下，我们能通过三个步骤定义来共识算法：

第一步：选举 Elect

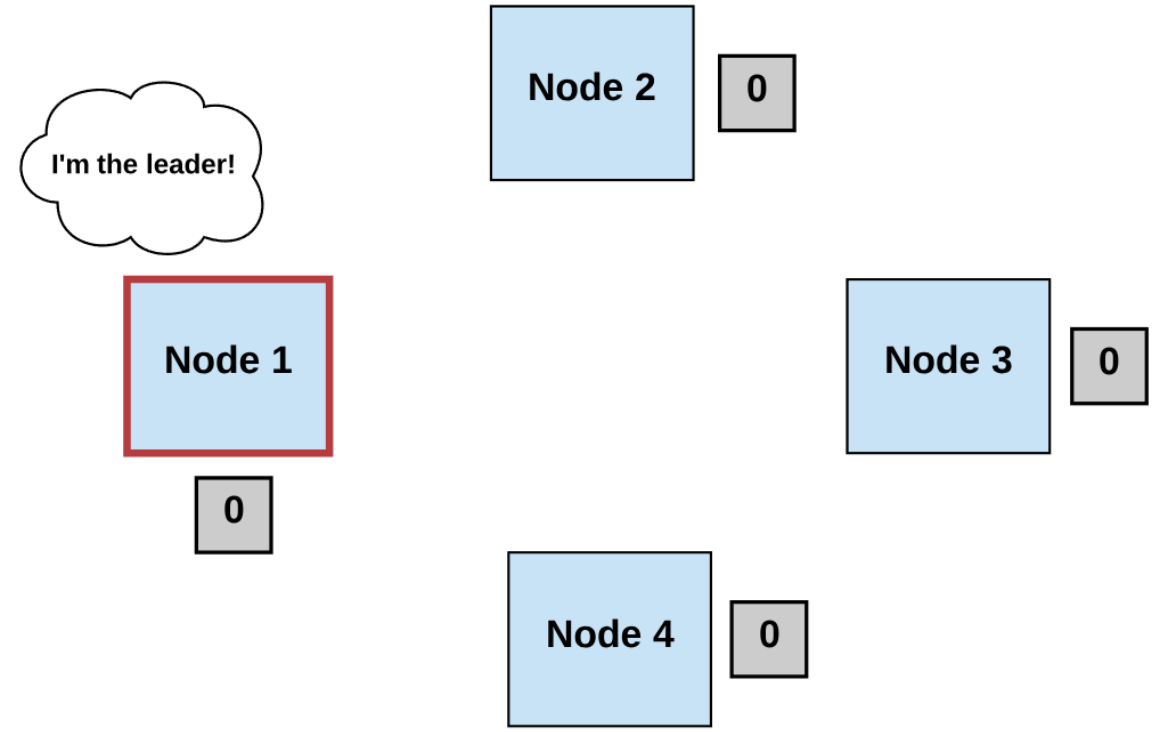
- 所有进程推举出某个进程（领导者）来做决策。
- 领导者提出下一个有效的输出值。

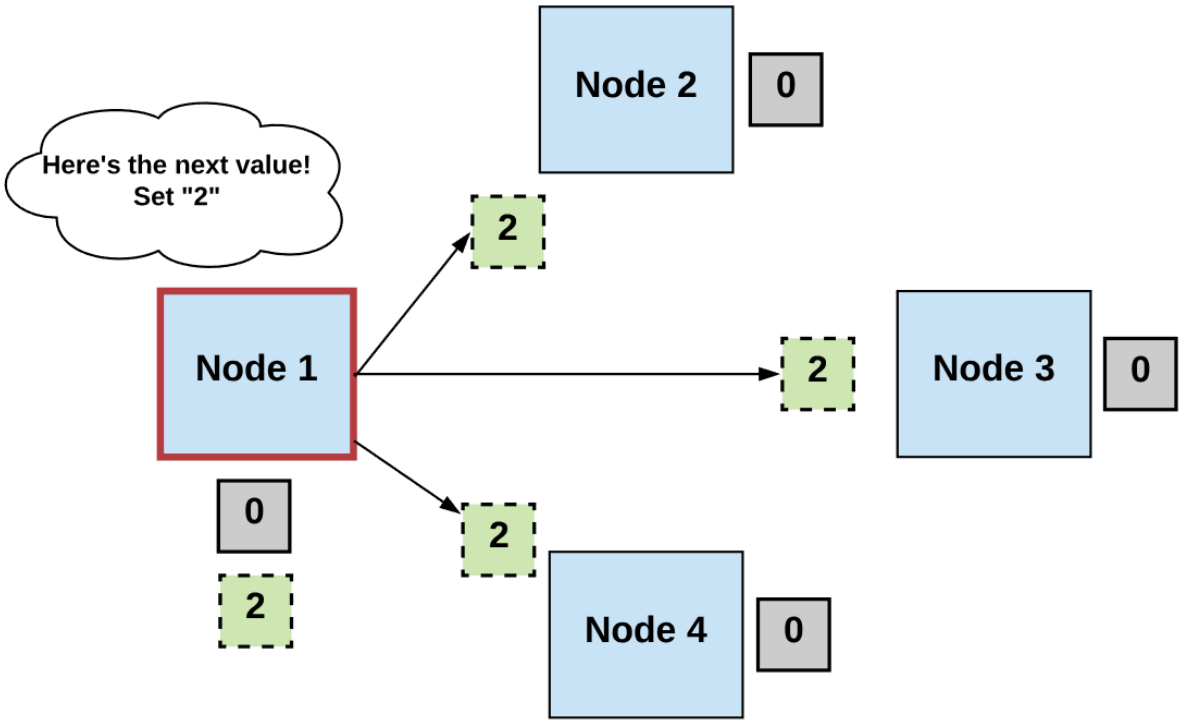
第二步：投票 Vote

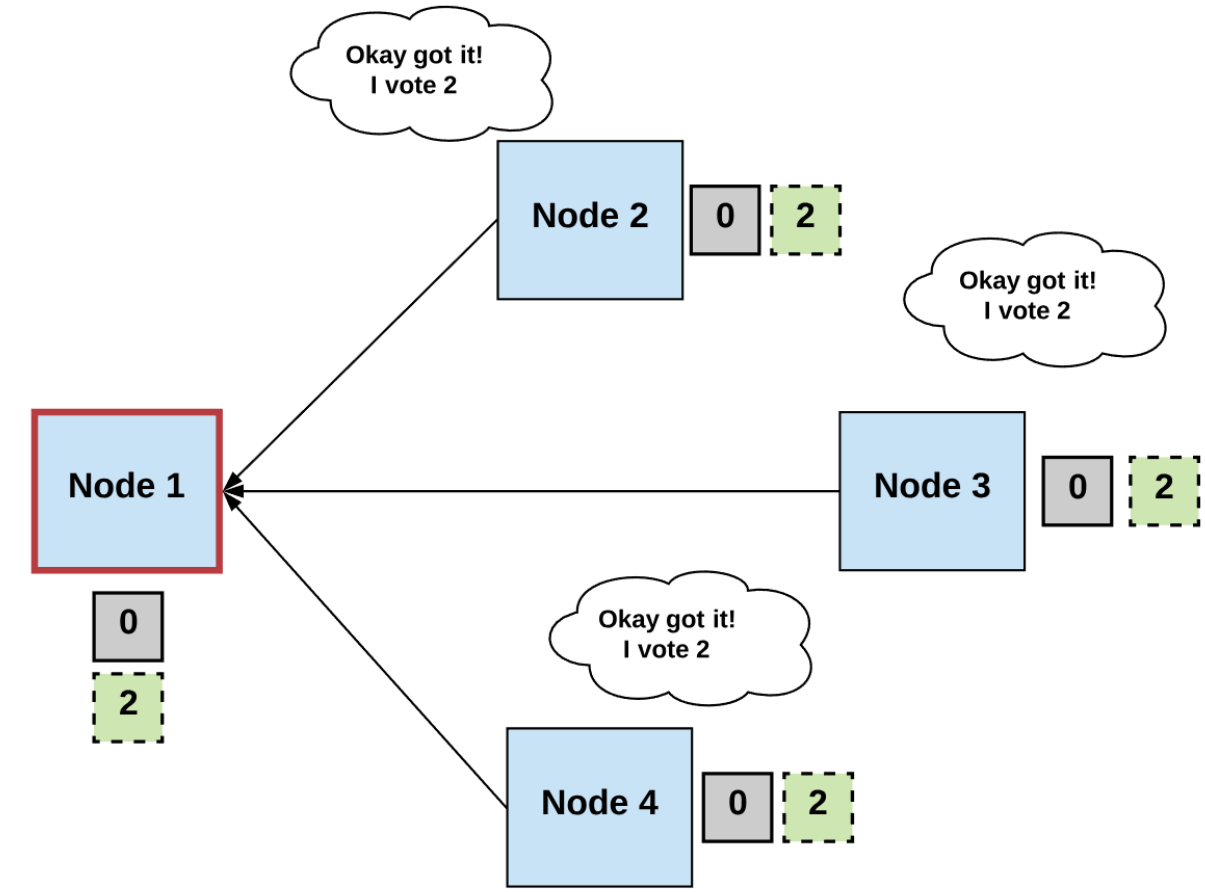
- 非故障进程会监听领导者进程提出的值，并验证这个值。然后将它作为下一个有效值提出。

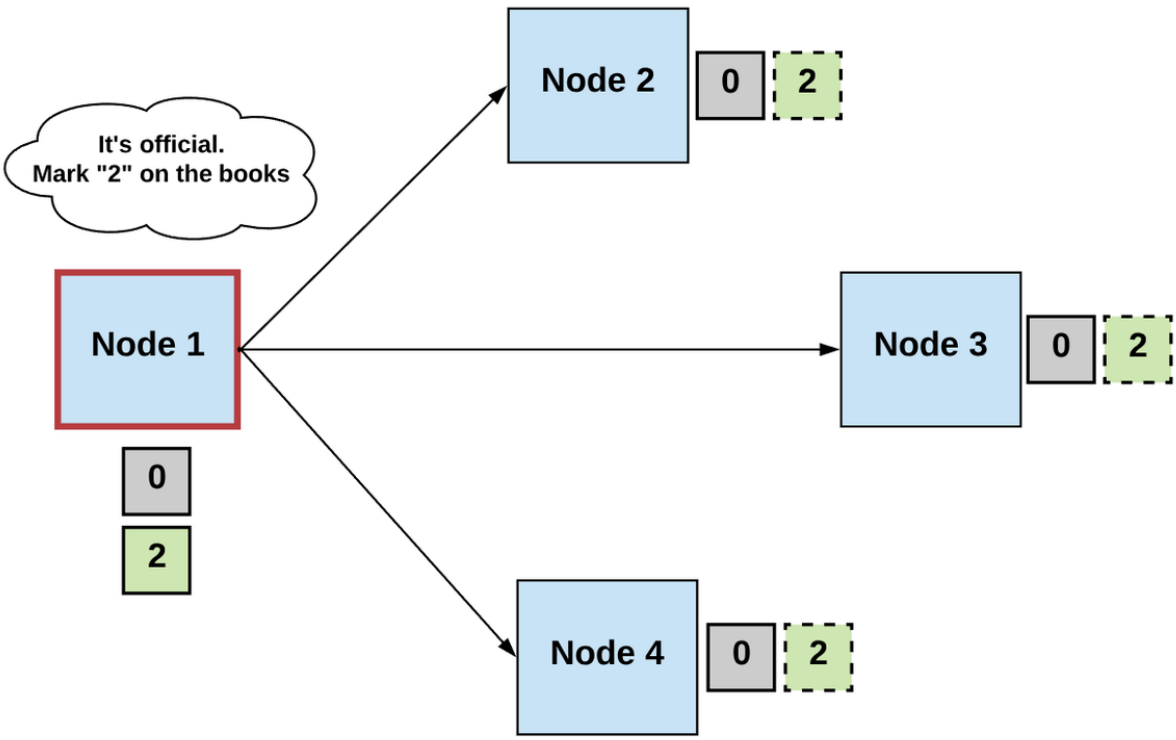
第三步：决定 Decide

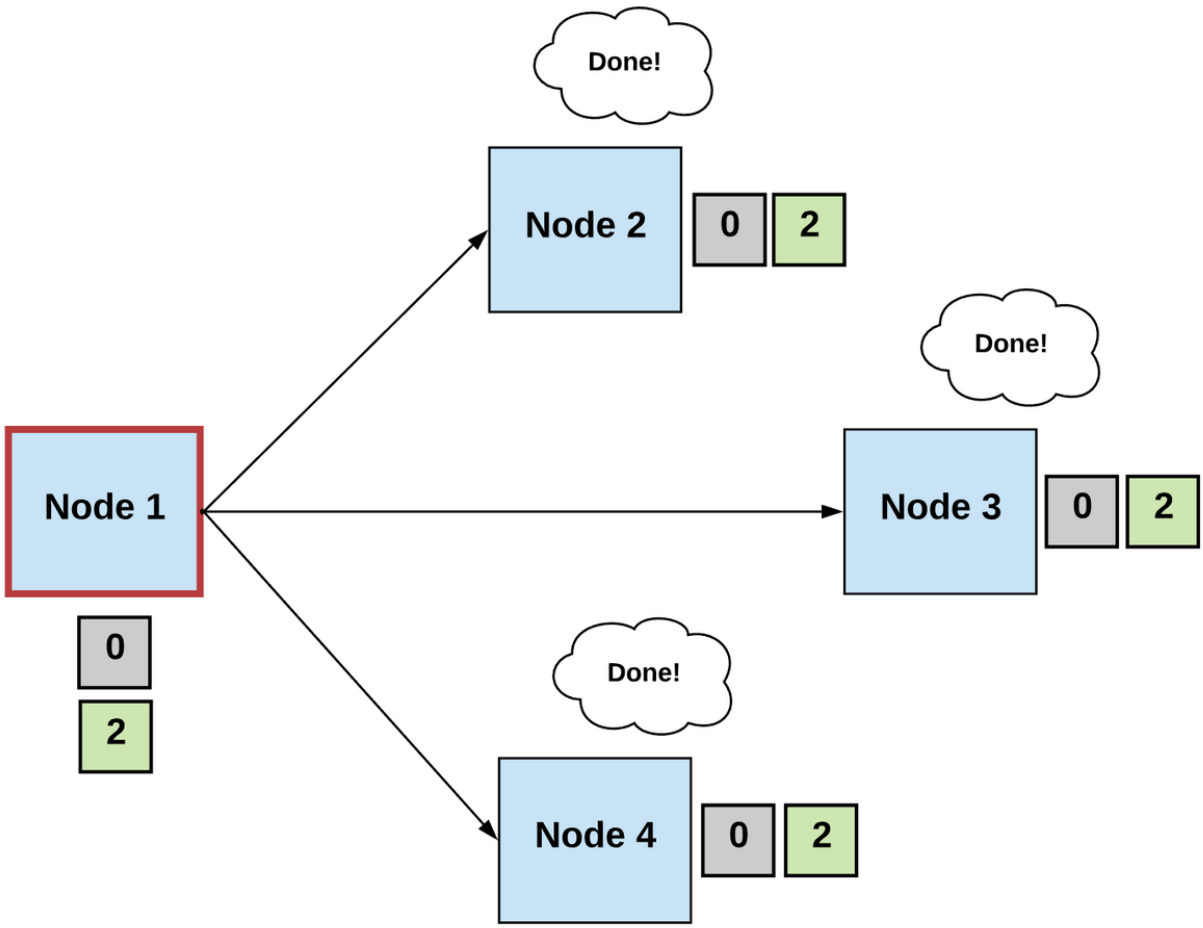
- 非故障节点必须就单个正确的值达成共识。如果这个值收到满足某个阈值条件的投票数，那么全体进程就会同意这个值为最终共识值。
- 如果没有达到条件，则重新进行上面步骤。






Okay got it!
I vote 2
Node 2
0
2
Okay got it!
I vote 2
Node 1
Node 3
0
2
0
2
Okay got it!
I vote 2
Node 4
0
2


Node 2
0
2
It's official.
Mark "2" on the books
Node 1
Node 3
0
2
0
2
Node 4
0
2



-作者制图-

值得注意的是，每个共识算法都会有些许不同，比如：

- 术语（如，轮次（round）、阶段（phase））
- 投票如何进行的，以及
- 决定最终值的标准（例如：验证条件）。

尽管如此，如果我们能以通用的过程构造共识算法，并保证满足上述的基本条件，那我们就有了一个能够达成共识的分布式系统。

你以为非常简单，对吧?

FLP 不可能定理

……并非如此，说不定你已经预见到了！

回想一下，我们是如何描述同步系统和异步系统之间的区别的：

- 同步通信情况下，消息会在固定时间范围内发送。
- 异步通信情况下，无法保证消息的传递。

这个差异非常重要。

在同步通信环境下有可能达成共识，因为我们能够假设消息传递需要的最长时间。因此在同步消息系统，我们允许不同的节点轮流成为提案者、进行多数投票，并跳过那些没有在最长等待时间内提交提案的节点。

但正如我们前面提到的，跳脱出可控（即消息延时可以预测）的环境（比如，具有同步原子钟的数据中心之类的可控环境），假设操作发生在同步通信环境里是不切实际的。

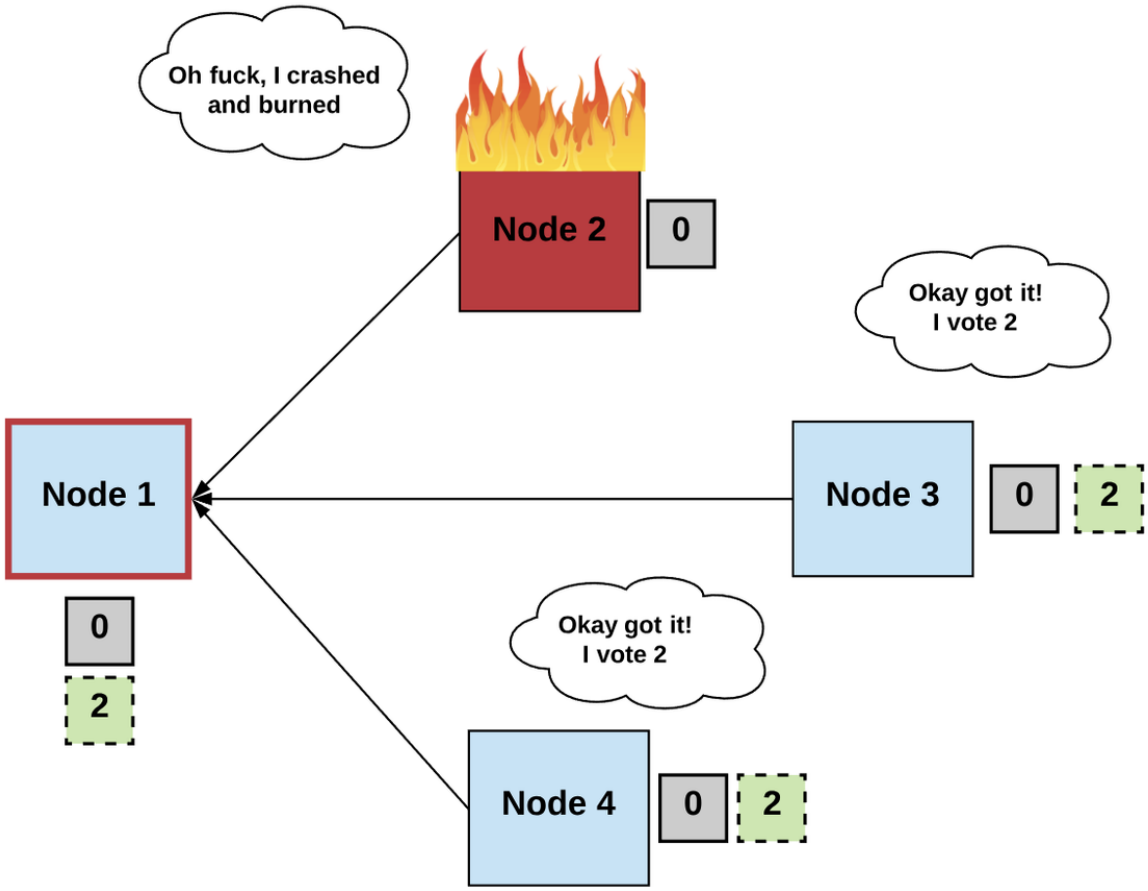
事实上，多数时候我们无法假设同步通信，所以我们的设计必须面向异步通信环境。

在异步通信环境中，因为我们无法假设一个最大的消息传递时间，想要达成 termination 条件是非常困难的。回忆下，要达成共识的其中一项条件是"可终止性"，也就是说每个非故障节点最终都必须选定某组相同的输出值，而不能永远迟疑不决。

这也就是大家熟知的" FLP 不可能性 "。它是怎么得到这个名字的？我很高兴你这么问了。

1985年，研究员 Fischer、Lynch 和 Paterson （FLP）在他们的论文《Impossibility of Distributed Consensus with One Faulty Process》中描述了为什么在异步通信环境下，单个进程故障也会导致共识无法达成。

简单来说，因为进程可能在任何时间出错，所以也有可能发生在正好会影响共识达成的时间点。



-亮点自寻-

这个结果对分布式计算领域带来重大打击，但即使如此，科学家仍在不断尝试避开 FLP 不可能性问题的方法。

抽象一点来说，有两个方法能够避开 FLP 不可能性问题：

1. 使用同步性假设
2. 使用非确定性机制

方法1：使用同步性假设

我知道你们在嘀咕：这究竟是啥？

让我们回头审视一下 FLP 不可能性问题。这里有个新的视角：FLP 不可能性表明，如果我们无法推进整个系统，就无法达成共识。换言之，如果消息是异步发送的，termination 条件就无法得到保证。回忆下，要达成共识的其中一项条件是“ termination ”，也就是说每个非故障节点最终必须选定某个输出值。

但是在异步通信网络里，我们不知道消息何时会被发送，我们怎么保证每个非故障进程都选定了一个输出值？

要澄清一下，这个发现没有证明共识不能达成。而且——因为消息异步传递，所以“不可能”在固定时间达成共识——这里说的“不可能”，指的是共识“并非总能”达成。这是个微妙但至关重要的细节。

这个规避 FLP 不可能问题的方法就是引入超时（timeout）的概念。如果在确认下个值的过程没有进展，我们会等到超时，然后重新进行共识的步骤。如我们所见，Paxos 和 Raft 使用的就是这类共识算法。

Paxos

Paxos 于 90 年代提出，是首个应用在真实世界、实用的、容错的一致性共识算法，它的正确性（correctness）被 Leslie Lamport 所证明。许多跨国互联网公司，如 Google 和 Amazon 也用其构建分布式服务，是个被广泛采用的共识算法之一。

Paxos 工作方式如下：

阶段1：准备请求

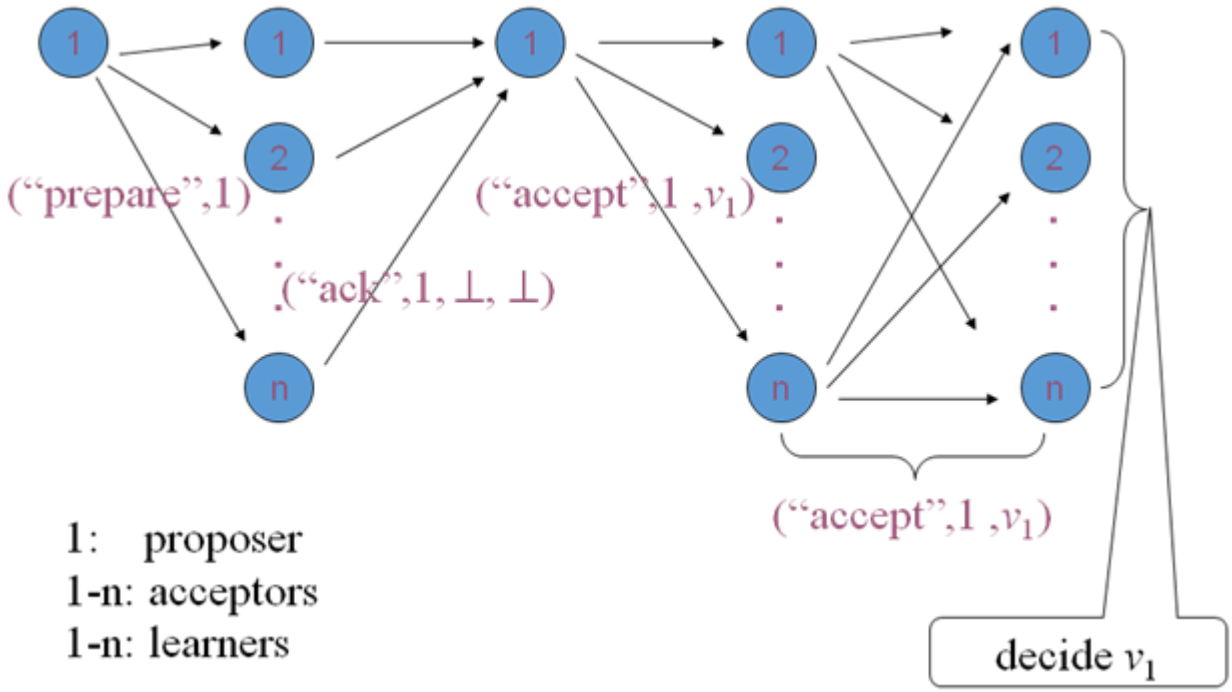
1. 提案者选择新的提案号（n），然后向接收者发送“准备请求”。
2. 当接收者收到准备请求 (“prepare,” n)，且 n 比已经接收到的其他准备请求大，接收者会发出消息 (“ack,” n, n’, v’) 或 (“ack,” n, ^ , ^)。
3. 接收者承诺不会再接纳任何提案号小于 n 的准备请求。
4. 如果有的话，接收者会发出具有最高提案号的提案值（v），反之则发出 ^ 。

阶段2：接收请求

1. 如果提案者收到大多数接收者的响应，则可以发出带有提案号 n 和价值 v 的接收请求 (“accept,” n, v) 。
2. n 是准备请求中出现的数字。
3. v 是响应中具有最高提案号的提案值。
4. 当收到接收请求(“accept,” n, v)，接收者就会采纳这个请求。除非接收者已经采纳了大于 n 的提案号。

阶段3：学习阶段

1. 每当接收者采纳了一个提案，就会向所有学习者返回 (“accept,” n, v)。
2. 学习者从多数接收者那儿得到 (“accept,” n, v)，选定最终的值 v，然后向其他学习者发送 (“decide,” v)。
3. 学习者收到 (“decide,” v) 和最终决定的值 v。



-来源：myassignmenthelp.net -

哈！你被绕晕了没？现在你有许多信息要消化，但...稍等一等。

我们知道，每个分布式系统都有故障。在这个算法中，如果提案者故障（比如，出现丢包错误），则最终决定允许被延后。面对此类问题，Paxos 会重新在阶段 1 提交新的提案号，即使之前的操作还没有终止。

想要从上述错误中将程序引导回正常流程是一件非常复杂的事，因为进程会插手并驱动这整个流程，所以我不想深入讨论。

Paxos 之所以晦涩难懂的主因，就是它将许多实现细节留给读者自行想象：我们如何知道提案者发生故障？是否使用同步时钟设置超时条件，来判别提案者故障与否，让整个系统能进入下个循环？

为了提供部署的灵活性，Paxos 描述的许多主要特性都是开放式的。比如领导者选举、侦错、日志管理等等内容，都非常模糊甚至完全没有提到。

后来，这种设计选择成为 Paxos 最大的缺点之一。它不仅难以理解，而且难以实施。反过来说，这使得分布式系统的领域非常难以驾驭。

现在你可能开始好奇：似乎上面完全没用到同步假设，那为什么把 Paxos 归为这一类算法呢？

虽然在 Paxos 算法中没有明确的定义超时，但在实际部署时，发生超时之后选择新的提案者是满足 termination 的必要条件。否则，我们无法保证接收者会输出下一个值，这会导致系统停止运转。

Raft

2013 年，Ongaro 和 Ousterhout 为 Raft 复制状态机提出了一种新型共识算法，其核心目标是可理解性（不像 Paxos 那样难懂）。

我们从 Raft 中学到的一个非常重要且新颖的东西是使用共享超时机制处理 termination 的概念。在 Raft 中，如果节点宕机并重启后，在试图声明成为领导者（Leader）之前，需要至少等待一个超时周期，并且一定会有所进展。

等等.....那么在 “拜占庭” 环境下又如何呢？

虽然诸如 Paxos 和 Raft 的传统共识算法能够使用某种程度的同步假设（即超时机制），在异步环境下正常运行，但是它们并不是拜占庭容错的。它们仅仅能够容忍宕机故障。

因为我们能够将处理过程设置/建模为工作或宕机（即，0 或 1 ）的状态，使得宕机故障能够更容易处理。进程不能作恶或说谎。因此，在一个宕机容错的系统中，分布式系统使用简单的“少数服从多数”的方式，就能够达成共识。

在一个开放且去中心化的系统中（例如：公有链），用户无法控制网络中的节点。相反，每个节点根据自己的目标行事，并且可能与其他节点的目标相冲突。

在拜占庭系统中，节点有着不同的动机，可以撒谎、协作或者任意行事，不能假设简单的“少数服从多数”的方式就足以达成共识。一半或者超过一半的节点可以联合起来作恶。

如果一个当选的领导人是拜占庭式的（即作恶节点），并且与其他节点的网络连接很通畅，那么它可能危及系统。回想一下我们刚刚说过的，我们必须为我们的系统建模，以容忍简单的错误或拜占庭错误。Raft 和 Paxos 是简单的容错共识机制，并不能容忍拜占庭错误。它们不是为了容忍恶意行为而设计的。

“拜占庭将军问题”

试图建立一个可靠的计算机系统，使其能够协调提供冲突信息的进程达成共识，便是所谓的“拜占庭将军问题”。一个拜占庭容错协议应该在部分节点作恶的情况下，能够达成其共同的目标（共识）。

Leslie Lamport、Robert Shostak 和 Marshall Pease 撰写的这篇 “*拜占庭将军问题（Byzantine General's Problem）*” 的论文，提出了第一种解决拜占庭将军问题的证明：该论文指出一个有 **x** 个拜占庭节点的系统中，必须总共至少有 **3x+1** 个节点才能够达成共识。

具体原因如下：

如果 **x** 个节点有错误，那么系统需要在 **n - x** 个节点间通过协作的方式正确运行（因为 **x** 个节点可能有错误或者为拜占庭式节点，并且没有响应）。然而，我们必须能够处理没有响应的 **x** 个节点可能不是出错的情况，这 **x** 个节点可能*确实*有响应。如果我们希望非故障节点数量大于故障节点数量，那么我们至少需要 **n - x - x > x**。因此，**n > 3x + 1** 是最优值。

然而，这篇论文所描述的算法仅适用于同步环境。真是懒蛋！看来我们只能让其中一个（拜占庭式或异步）成立，能在同时满足异步和拜占庭假设的分布式系统中运行的算法比这个还要更难设计。

为什么呢?

简而言之，构建一个既能满足异步环境又能保证拜占庭容错的共识算法……emmm，就像创造奇迹一样。

共识算法有像 Paxos 和 Raft 这样知名且广泛使用的，但也有大量的学术研究更多地关注解决拜占庭+异步环境的共识问题。

So……系好安全带！

我们要去实地考察一下……

到……

理论学术论文去！

Okay, Okay——我很抱歉这么说。但你不觉得很有意思吗！还记得我们之前提到的“创造奇迹”吗？我们来看看两种算法（DLS 和 PBFT），它们让我们比以往任何时候都更接近于打破拜占庭+异步环境困局。

DLS算法

Dwork、Lynch 和 Stockmeyer （因此该算法叫 “DLS” 算法）的论文 “部分同步环境下的共识算法（Consensus in the Presence of Partial Synchrony）” 介绍了一种拜占庭容错共识主要进展：其定义了在“部分同步系统”中达成共识的模型。

你还可能记得，在同步系统中，消息从一个处理器发送到另一个处理器所需的时间有一个已知的固定上界。在异步系统中，并没有消息传递的固定上界，而部分同步系统介于这两种极端之间。

这篇论文解释了两种部分同步系统：

- 假定消息传播存在固定上界，但上界并不是初始确定的（即，先验上界）。系统目标是在不获知实际上界的情况下达成共识。
- 假定消息传播的上界是已知的，但是只保证从某个未知的时间点（也称作 “全局标准时间（Global Standardization Time，GST）”）开始保持不变。目标是设计一个无论何时都能够达成共识系统。

DLS 算法是这样工作的：

系统运行的每一轮被分为 “尝试” 和 “锁定释放” 阶段。

1. 每一轮都有一个提议者，并且在每一轮开始后，进程之间互相通讯，发布它们认为正确的数值。
2. 如果至少有 **n - x** 个进程向提议者发送某个相同的值，则提议者向全网“发布”这个数值。
3. 当一个进程收到提议者提出的数值时，它必须将这个值锁定，并且广播这个数值给其他进程。
4. 如果提议者从 **x+1** 个进程中收到它们已经锁定了某个值，则提议者将该值提交为最终数值。

DLS 是一项重大突破，因为它创造一种新型网络假设类型（即，部分同步），并且证明在这种假设情况下能够达成共识。DLS 论文的另一个重要结论是将在拜占庭和异步设置中达成共识的关注点分为两部分：安全性（safety）和活性（liveness）。

（译者注：

- 安全性：错误的事情永远不会发生
- 活性：正确的事情一定会发生（尽管不知道何时发生））

安全性

这是描述我们之前讨论的 “一致性” 属性的另一个术语，即所有没有错误的进程能够就相同的输出达成共识。如果我们能够保证安全性，我们就能够保证整个系统保持同步。尽管存在故障节点或恶意节点，我们依旧希望所有节点就事务（或者说区块链中的交易）日志的总顺序能够达成共识。违反安全性意味着我们最终将得到两个或多个有效的事务日志。

活性

这是描述我们之前讨论的 “可终止性” 属性的另一个术语，即每个没有错误的节点最终就某个输出值达成共识。在区块链场景中，“活性” 意味着区块链持续增加合法的新区快。活性是非常重要的是，因为它是网络持续有效的唯一途径 —— 否则区块链系统将停滞。

从 FLP 不可能性问题我们知道，完全异步的系统无法达成共识。DLS 论文认为，假设系统环境为部分同步，那么就足以克服 FLP 不可能性问题，从而达到活性状态。

因此，该文证明 *DLS 算法不需要任何同步假设就可以达到安全性状态。*

很简单，对吧？如果还觉得困惑也不用担心。让我们再深入一点。

需要记住的是，如果系统中节点没有决定某个输出数值，那么系统就会停止。因此，使用同步假设（即超时机制）来保证 terminaton 是有意义的，因为只要其中一个同步假设被打破，网络就将中止。

但是如果我们设计一个假设超时（以保证正确性）的算法，那么如果同步假设失效，就有产生两个合法事务日志的风险。

这比之前的选择危险的多。如果服务被破坏（即不安全），那么提供有用的服务（即活性）是没有意义的。基本上，有两个不同的区块链比让整个区块链网络停止更加糟糕。

分布式系统设计总是需要权衡利弊。如果想克服某些限制（例如，FLP 不可能问题），就必须在其他地方做出牺牲。在这种情况下，将关注点划分为安全性与活性是非常机智的。它使得我们能够构建一个在异步环境中安全的系统，但仍然需要某些超时机制以保证能够持续产生新的数据。

尽管 DLS 论文解决了各方面问题，但是 DLS 算法并没有在真实拜占庭环境中广泛实现或使用。**这可能是由于 DLS 算法中的一个核心假设是使用同步处理器时钟来同步时间。**事实上，同步时钟很容易受到攻击，并且在拜占庭容错环境中表现不佳。

PBFT 算法

另一种拜占庭容错算法，由 Miguel Castro 和 Barbara Liskov 于 1999 年发表，叫做 “实用拜占庭容错（Practical Byzantine Fault-Tolerance，PBFT）”。对于具有拜占庭行为的系统，PBFT 被认为是一种更实用的算法。

从这个意义上说，“实用” 意味着它能够在类如互联网的异步环境中工作，算法中的部分优化，使其比先前的共识算法更加快速。该论文指出，先前的共识算法虽然被证实 “理论上可行”，但要么太慢无法使用，要么需要为了安全性而假设完全同步。

正如我们所讲到的，先前共识算法的设计在异步环境下是非常危险的。

简而言之，PBFT 算法假设在 **(n-1)/3** 个节点发生故障的情况下，它依旧能够保障安全性与活性。正如我们之前讨论的，这是容忍拜占庭故障所需的最小节点数。因此，该算法的弹性是最优的。

只要小于一定比例，无论有多少节点出错，该算法都能够保障安全性。**换句话说，它无需为保障安全性而假设同步环境。然而，PBFT 算法的活性依赖于网络同步。**最多容许 **(n-1)/3** 个节点出错，且消息延迟的增长速度不会超过某一具体时间限制。因此，PBFT 通过使用同步假设保证活性来绕过 FLP 不可能性问题。

PBFT 算法通过一系列 “视图” 运行，每个视图有一个 “主” 节点（即，领导者），其他节点称为 “备份”节点。下面是它具体工作流程：

1. 在一台客户端上发生了一个新事务，并被广播给主节点。
2. 主节点将该事务广播给所有 “备份” 节点。
3. 各备份节点执行该事务，并给客户端发送一条回复消息。
4. 客户端期望从备份节点收到 **x + 1** 个相同回复。这个相同的回复就是最终结果，并且系统的状态会发生相应转变。

如果领导者没有故障，那么协议就能正常工作。然而，检测恶意/坏的主节点并重新选取主节点的过程（也叫做“视图更改”）非常低效。**例如，为了达成共识，PBFT 的信息交换量将达到 O(n^2) 级别，即每台计算机都需要与网络中其他所有计算机通讯。**

插播：我会用另一篇独立的博客完整解释 PBFT 算法！以后再说;)。

尽管 PBFT 算法相较于之前的算法有了一定的改进，但在具有大量参与者的实际用例（例如：公有链）中，它仍不够实用。但是，与 Paxos 相比，至少在故障检测以及领导者选举这类事情上，它要更具体，更实用。

需要承认 PBFT 的贡献。它包含了一些重要的突破性思想，这些思想是新型共识协议（特别是在后区块链世界中）可以借鉴和使用的。

例如，Tendermint 就是一种受 PBFT 影响很深的新型共识算法。在它们的 “验证” 阶段，Tendermint 使用与 PBFT 相似的两次投票方式决定最终值。Tendermint 算法最主要的不同是比 PBFT 更加实用了。

Tendermint 每轮都会换一个新的领导者。如果本轮领导者在一段时间内没有响应，那么将跳过该领导者，算法将简单地进入具有新领导者的下一轮。实际上，这比每次改变视图或选新领导者都需要使用点对点连接通信的方式更有意义。

方法二：非确定性算法

如前所述，为了解决 FLP 不可能性，大部分拜占庭容错协议最终都要基于同步性假设的前提。然而，解决 FLP 不可能性还可以另辟蹊径：非确定性算法。

当当当当，中本聪共识！

认真听讲的同学应该还记得，在传统的共识中，状态转变函数 **f(x)** 是这样定义的：提案者和一众接收者必须全体协作决定下一状态的值。

这种做法未免过于复杂了，因为这要求我们明晰每一个节点的状态，*并且* 每一个节点都要与其余所有的节点交换数据（也就是说信息交换量达到 O(n^2) ）。由此显而易见，传统共识不具备良好的可拓展性，且无法在开放、用户能无需特别许可，随时加入随时退出的分布式系统中达成。

中本聪共识的精妙之处在于把上述问题转变成一个概率问题。与其让所有节点对一个确定的值达成一致，不如构造 f(x) 来使所有节点对某一输出值正确性的概率达成一致。

等等，你是不是又觉得我没在说人话了？

拜占庭容错

和传统共识先选本轮领导、再节点协作的模式不同，中本聪共识是基于哪一个节点最快计算出难题达成的。比特币区块链中每一个新区快都由本轮最快计算出数学难题的节点添加。整个分布式网络持续不断地建设这条有时间戳的区块链，而承载了最多计算量的区块链正是达成了共识的主链（即累积计算难度最大）。

最长链不仅是区块顺序的明证，也是 CPU 算力最大消耗的明证。因此，只要绝大多数的算力掌握在诚实的节点手中，这些节点继续帮助建设最长链，就能保证最长链始终比攻击者的区块链更长，共识不被破坏。

区块奖励

中本聪共识之所以行之有效，是基于这样一个合理假设：节点会为了下一个区块的记账权而不断扩充自己的算力。它的巧妙之处在于设计了经济激励，节点需要不断解决大计算量难题来获取赢得大额奖励的机会（即区块奖励）。

防女巫攻击

工作量证明需要解决计算难题的特点天生可以抵抗女巫攻击。整个系统不需要任何 PKI 或是其它花哨的身份认证策略。

点对点 gossip

应用 gossip 协议 是中本聪共识的一大创举。在无法假设节点之间连接状态是否良好的分布式系统中，gossip 协议更适合点对点数据传输。与其他协议不同，我们假定节点只和有限的几个其他节点相连，然后在节点之间利用点对点协议以 gossip 协议传输消息。

在异步环境下不是“严格”安全的

在中本聪共识中，安全性是一定概率下的安全性——在我们构建最长链的过程中，每一个新区块的加入都在降低攻击者伪造替代链得手的概率。

因此，中本聪共识在异步网络的前提下不具备严格安全性。下面我们好好掰扯一下为什么。

在异步通信的前提下，一个分布式系统的安全性体现在它能排除异步网络的影响，能维持一份一致的事务日志。换言之，就是一个节点在随时离开网络后，无论在何种网络状况下再次加入系统，都能仅仅根据区块链的创世块找到最近一个正确的区块。新加入的诚实节点能查询到过去任意高度上的区块，而恶意节点无法伪造可信的虚假区块信息。

本文前面提到的传统共识算法满足这种严格安全性，是因为每一个事务都有一个确定值。只要保证了每个值的确定性，就能知道过去任意时刻的状态。但比特币不满足"严格"异步安全，因为攻击者可能屯聚巨量算力，盘踞于网络一隅，以超过诚实节点生成区块的速度制造自己的替代链，从而在替代链上成功实施双花攻击。

诚然，攻击者不仅要耗费巨量哈希算力，也要付出一大笔资金。

本质而言，比特币区块链的不可篡改性就是因为绝大多数矿工不想切换到另一条替代链。为什么呢？因为即使相互串谋，动员到能让区块链网络切换成替代链的算力也太难了。因此可以这么说，由于切换替代链攻击的可能性过于低，在实际应用中可以把这种风险性忽略不计。

中本聪共识 vs. 传统共识

从应用角度出发，中本聪共识具备拜占庭容错性。但因为它显然无法达成共传统共识研究者们所认同的共识，所以起初学界并没有将它归入拜占庭容错协议的家族中去。

从设计上讲，中本聪共识成就了任意节点加入分布式网络的可能性，使节点无需知道其他节点的全集就能正常工作。做出这样伟大的设计要我怎么夸你呢！感谢你，中本聪。

中本聪共识较传统共识更为简单，它消除了点对点通信、领导人选举、平方级通信复杂度等等复杂的操作。

这一切都使得它能在现实条件下轻松部署，不愧是比 PBFT 更"实用"（"practical"）的好弟弟。

Conclusion结语

以上便是分布式系统和共识知识的基础概述。探索之路漫长而曲折，布满荆棘，而今时今日分布式计算的发展已然这般巧妙。希望这篇文章能为你在这个领域的研究带来一点帮助。

中本聪共识的创新确真伟大，它推动了一股新的技术潮流，从研究员、科学家到开发者和工程师，他们都在积极探索，尝试打破共识协议研究领域旧有的边界。

随着中本聪共识的落地，各种各样新的共识协议在它的基础之上百花齐放：除了 Proof-of-Steak ，你还能列举出哪些呢？;) 这里我先卖个关子，欲知后事如何，且听下回分解！

注意：为了不至于把文章写成一本书，本文仍有许许多多重要的论文和算法未曾提及。例如，Ben Orr 的 Common Coin 项目也用了概率手段解决共识问题，但不具备最优范围弹性。其他诸如 Hash Cash 等等的算法也使用了 PoW 协议，但却仅用于限制垃圾邮件和拒绝服务攻击问题。不仅如此，关于传统共识协议我也略去了很多！我认为上述内容足够为你留下共识领域传统和中本聪模式之间差异的一瞥。下篇文章，我们不见不散！

向 Zaki Manian 致以衷心的感谢，他在本文成文过程中解答了我关于分布式共识的各种问题。

（全文完）