第3章

共识算法

第2章介绍了作为区块链核心技术的密码学的一些重要知识，在本章中将为读者介绍区块链技术的另一个重要核心技术——分布式共识算法。分布式共识算法是分布式系统中保证系统状态一致性的重要技术，是分布式文件系统、分布式数据库的重要基础。区块链系统采用分布式共识算法在无中心节点控制、同时又可能存在破坏节点的环境下确立系统状态，从而建立信任。区块链因此也被称为“信任机器”，由此可见，共识算法是区块链技术中需要重点掌握的关键技术。共识算法因不同的应用场景而有所不同，从区块链的部署架构来看，有适合于公有链的共识算法，也有适合于联盟链或私有链的共识算法。公有链上的共识算法一般需要支持高扩展性，能够在共识节点动态出入网络的情况下保证共识流程的节奏，同时又要防止拜占庭节点对网络的攻击。受限于FLP定理和CAP定理，公有链的共识算法一般不能保证强一致性。而另一方面，联盟链的共识算法一般需要支持强一致性、高性能，但对高扩展性和防拜占庭节点攻击方面的要求往往没有公有链的要求那么高。

不同的共识算法在一致性、正确性、可终止性、交易吞吐量、记账频率、扩展性和安全性各有不同。在实际应用中，需要根据应用场景，选择能满足应用需求的共识算法。下面我们来介绍分布式共识算法的背景和典型的共识算法。

3.1 分布式共识算法背景

在数据库系统中，两阶段提交(Two-Phase Commit)是一个普遍使用的保证一致性的方法。如图3.1所示，两阶段提交由一个协调节点(Coordinator)居中控制。

第一阶段，协调节点会向所有节点发送一个询问“是否准备好”的消息，各节点会响应该消息；

第二阶段，如果各节点的响应都是准备好的回复，协调节点会向各节点发“提交”指令，否则会发“回滚”指令。

这样所有节点的状态都会保持一致，提交新的状态，或者回滚成以前状态。两阶段提交

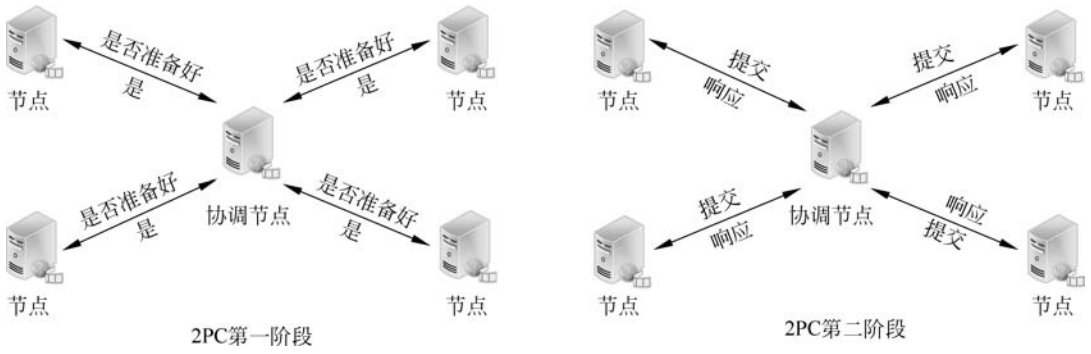

图 3.1　两阶段提交协议

协议在没有出现节点故障的情况下可以保证系统状态的一致性，但是当协调节点和另外一个节点同时出现故障的时候，系统没有办法知道故障节点的状态，因此无法决定是提交还是回滚。

为了解决这个问题，三阶段提交(Three-Phase Commit)协议被提出来(见图 3.2)，即在正式提交前，协调节点再发一次“准备提交”的消息，在收到所有节点的回复后，协调节点才最后发“提交”的指令。三阶段提交能部分解决两阶段提交中协调节点和另一节点同时出故障的问题，但是系统过分依赖一个协调节点，另外其余节点也没有办法形成对协调节点是否出现故障的共识，因此，无论两阶段提交或三阶段提交，都不能作为一个分布式共识算法，只能在通常情况下保障分布式系统的一致性。

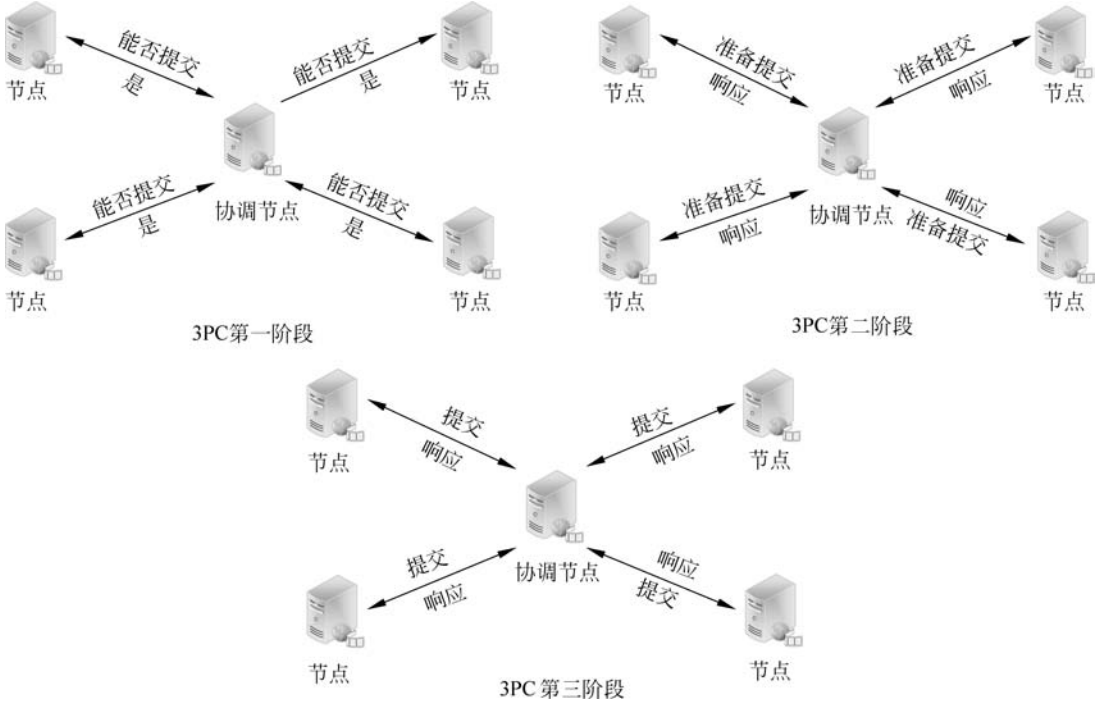

图 3.2　三阶段提交协议

两阶段提交或三阶段提交多数用在同步通信的分布式系统中，而通常的分布式系统是由多个主机通过异步通信方式组成网络集群。在这样的一个异步系统中，需要主机之间进行状态复制，以保证每个主机达成一致的状态共识。然而，异步系统中，可能出现无法通信的故障主机，而主机的性能可能下降，网络可能拥塞，这些可能导致错误信息在系统内传播。因此，需要在默认不可靠的异步网络中建立容错协议，以确保各主机达成安全可靠的状态共识。

利用区块链构造基于互联网的去中心化账本，需要解决的首要问题是如何实现不同账本节点上的账本数据的一致性和正确性。这就需要借鉴已有的在分布式系统中实现状态共识的算法，确定网络中选择记账节点的机制，以及如何保障账本数据在全网中形成正确、一致的共识。

在 20 世纪 80 年代出现的分布式系统共识算法，是区块链共识算法的基础。下面从基本的拜占庭将军问题入手，逐步介绍适合于联盟链/私有链和公有链的共识算法。

3.1.1 拜占庭将军问题

拜占庭将军问题是区块链平台的共识机制需要解决的一个核心问题。学习共识算法首先要理解拜占庭将军问题。

1. 拜占庭将军问题提出与约束条件

拜占庭将军问题是 Leslie Lamport 在 20 世纪 80 年代提出的一个假象问题。拜占庭是东罗马帝国的首都，由于当时拜占庭罗马帝国国土辽阔，各支军队的驻地分隔很远，将军们只能靠信使传递消息。发生战争时，将军们必须制订统一的行动计划。然而，这些将军中有叛徒，叛徒希望通过发布错误消息来影响忠诚将军们制订统一的行动计划，以达到破坏忠诚的将军们统一行动的目的。因此，将军们必须有一个预定的方法协议，使得满足两个条件：

A：使所有忠诚的将军能够达成一致；

B：少数几个叛徒不能使忠诚的将军做出错误的计划。

条件 B 比较难用形式化方式描述，这里需要了解的是将军们怎么做决策。一般来说，每个将军都会把他观察到的敌情派通信员告知其他将军。假设 $v(i)$ 是第 i 个将军通知其他将军的消息，每个将军会采用某种方法把所有收到的消息 $v(1)v(2)\cdots v(n)$ 组成一个单一的行动计划，其中 n 是将军的数目。条件 A 可以通过采用同一方法来组成行动计划来达到，而条件 B 则可以通过某种鲁棒性方式来达到，例如，通过多数人的意见来决定最后的行动计划。

如果要满足条件 A，下列条件必须满足：

(1) 每个忠诚的将军必须收到相同的命令值 $v(1),v(2),\cdots,v(n)$（$v(i)$ 是第 i 个将军的命令）。

(2) 如果第 i 个将军是忠诚的，那么他发送的命令和每个忠诚将军收到的 $v(i)$ 相同。

Lamport 把以上所有将军互相发命令的问题简化成一个将军发指令给 $n-1$ 个副官，从而得出拜占庭将军问题。

拜占庭将军问题———一个发送命令的将军要送一个命令给其余 $n-1$ 个副官，使得

IC1. 所有忠诚的接收命令的副官遵守相同的命令；

IC2. 如果发送命令的将军是忠诚的，那么所有忠诚的接收命令的副官将遵守所接收的

命令。

拜占庭将军论断——Lamport 指出在将军通过通信员口头传递信息的情况下，只要有 1/3 以上的叛徒，则没法保证忠诚的副官能达成一致的行动。

Lamport 采用一个简单的图来说明在一个将军两个副官的情况下，只要有一个叛徒存在，就不可能达成共识。如图 3.3 所示，当将军是叛徒的情况下，忠诚的副官 1 和副官 2 没法遵从同样的命令，因此没办法保障 IC1 的成立。

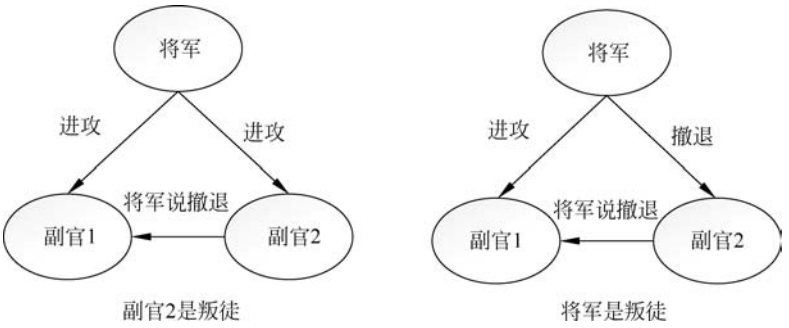

图 3.3　三个拜占庭将军，一个叛徒

Lamport 对拜占庭将军问题的研究表明，当 $n>3m$ 时，即叛徒的个数 m 小于将军总数 n 的三分之一时，通过口头同步通信（假设通信是可靠的），可以构造同时满足 IC1 和 IC2 的解决方案，即将军们可以达成一致的命令。但如果通信是可认证、防篡改伪造的（如采用 PKI 认证、消息签名等），则在任意多的叛徒（至少得有两个忠诚将军）的情况下都可以找到解决方案。

Lamport 对口头同步通信的定义如下：

A1. 每个发送的消息都会被正确地接收；

A2. 接收消息的人知道谁发送的消息；

A3. 消息没有发送可以被检测到。

Lamport 利用递归的方式定义了一个能在系统中叛徒少于 1/3 的将军总数的情况下，采用口头同步信息达成共识的算法，简单表示成 OM(m)，即在最多 m 个叛徒的情况下，该算法能使总共超过 $3m+1$ 的将军解决拜占庭将军的问题。该算法假设有一个 majority 的多数函数，当大多数的 $v(i)$ 值等于 v，则 majority($v(1),\cdots,v(n-1)$) 等于 v。如果 $v(i)$ 不存在，该函数值是 RETREAT（默认值是撤退）。

2. 算法与定理

1) M(0)（$m=0$，没有叛徒的情况）

(1) 将军发他的值（相当于命令）给每一个副官。

(2) 每个副官用他接收到将军的值作为当前值，如果没有接收到，用 RETREAT（默认值撤退）。

2) 算法 OM(m)，$m>0$

(1) 将军发他的值给每一个副官。

(2) 对每个 i，让 $v(i)$ 等于副官 i 从将军那里接收到的命令，或者如果没有接收到指令的话，值就是 RETREAT（默认值撤退）。副官 i 作为算法 OM($m-1$) 的将军，发 $v(i)$ 给其他 $n-2$

个副官。

(3) 对每个 i,以及每个 $j \neq i$,让 $v(j)$等于步骤(2)中采用算法 OM($m-1$)副官 i 从副官 j 接收到的值,或者是没有收到值则默认为 RETREAT。副官 i 取 majority($v(1), v(2), \cdots, v(n-1)$)的值。

为了证明算法 OM(m)对任意 m 能解决拜占庭将军问题,先证明以下引理。

引理 1:对任意 m 和 k,如果有多于 $2k+m$ 个将军和最多 k 个叛徒,算法 OM(m)满足 IC2。

证明:证明采用归纳法。IC2 只是规定在将军是忠诚的情况下需要满足的条件。根据 A1,可以容易看出当将军是忠诚的时候,OM(0)成立,所以引理 1 在 $m=0$ 情况下成立。现在假设在 $m>0$ 的情况下成立,只需要证明在 $m-1$ 情况下也成立,通过数学归纳法,可以得出引理 1 成立。在算法 OM(m)第一步,忠诚的将军发一个值 v 给所有 $n-1$ 个副官。在第(2)步,每个忠诚的副官递归调用 OM($m-1$),总共 $n-1$ 个将军。

根据假设 $n>2k+m$,可以得到 $n-1>2k+(m-1)$,所以可以采用归纳法来得出每个忠诚的副官从第 j 个忠诚的副官得到 $v(j)=v$。因为最多只有 k 个叛徒,并且 $n-1>2k+(m-1)>2k$,所以 $n-1$ 个副官中的多数是忠诚的。因此,每个忠诚的副官都在 $n-1$ 个副官的多数中收到 $v(i)=v$,所以在第(3)步会得到 majority($v(1), v(2), \cdots, v(n-1)$)$=v$,证明了 IC2,即当将军是忠诚的情况下,所有忠诚的副官都执行接收的命令。

以下定理确认算法 OM(m)能解决拜占庭将军问题。

定理 1:对任意 m,如果在至多 m 个叛徒,而将军数超过 $3m$ 个的情况下,算法 OM(m)满足条件 IC1 和 IC2。

证明:采用归纳法来证明。如果没有叛徒,很容易得出 OM(0)满足 IC1 和 IC2。假设定理在 OM($m-1$)时成立,只需要证明在 OM(m),$m>0$ 时也成立。

首先考虑将军是忠诚的情况。在引理 1 中,使 $k=m$,可以得出 OM(m)满足 IC2。如果将军是忠诚的,IC1 也会自动满足。所以只需要在将军是叛徒的情况下验证 IC1。最多只有 m 个叛徒,由于将军是其中一个,因此有 $m-1$ 个副官是叛徒。因为有超过 $3m$ 个将军,会有超过 $3m-1$ 个副官,而且 $3m-1>3(m-1)$。因此采用递归来得出 OM($m-1$)满足条件 IC1 和 IC2。所以,对每个 j,任何两个忠诚的副官在第三步的时候会得到相同的 $v(i)$(如果两个副官中的一个是副官 j,可以从 IC2 中得出,否则可以从 IC1 中得出)。因此,任意两个忠诚的副官可以得到一样的向量 $v(1), v(2), \cdots, v(n-1)$,因此会在步骤(3)获得一样的 majority($v(1), v(2), \cdots, v(n-1)$),证明 IC1 成立。

综上所述,拜占庭将军问题的实质就是要寻找一个方法,使得在有故障情况下能建立系统的共识。在分布式系统中,特别是在区块链网络中的环境,也和拜占庭将军的环境类似,有运行正常的服务器(类似忠诚的拜占庭将军),有故障的服务器,还有破坏者的服务器(类似叛变的拜占庭将军)。共识算法的核心是在正常的节点间形成对网络状态的共识,也就是解决拜占庭将军问题。

3.1.2 共识系统的基本定义

从 3.1.1 节可见,拜占庭将军问题在一个分布式系统中,是一个非常有挑战的问题。在这里,先给出分布式计算中有关拜占庭缺陷和故障的 4 个定义。

定义 1：拜占庭缺陷(Byzantine Fault)。任何从不同观察者角度看表现出不同症状的缺陷。

定义 2：拜占庭故障(Byzantine Failure)。在需要共识的系统中由于拜占庭缺陷导致丧失系统服务。

定义 3：宕机缺陷(Fail-Stop Fault)。在需要共识的系统中导致进程停止运行发生的缺陷，该缺陷不对系统产生其他副作用。

定义 4：宕机恢复故障(Fail-Recovery Failure)。在需要共识的系统中导致进程停止运行发生的故障，该故障在重启进程后恢复，不对系统产生其他副作用。

在分布式系统中，不是所有的缺陷或故障能称作拜占庭缺陷或故障。像宕机、丢消息等缺陷或故障不能算为拜占庭缺陷或故障。拜占庭缺陷或故障是最严重缺陷或故障，拜占庭缺陷有不可预测、任意性的特点，例如遭黑客破坏、中木马病毒的服务器就是一个拜占庭服务器的例子。

在一个分布式系统中，所有的进程都有一个初始值。在这种情况下，共识问题(Consensus Problem)，就是要寻找一个算法和协议，使得该协议满足以下三个属性。

(1) **一致性(Agreement)**：所有的非缺陷进程都必须同意同一个值。

(2) **正确性(Validity)**：所有非缺陷的进程所同意的值必须来自非故障进程所提议的值。

(3) **可结束性(Termination)**：每个非缺陷的进程必须最终确定一个值。

通常也把满足一致性和正确性称为安全性(Safety)，而满足可结束性称为活性(Liveness)。

根据 Fischer-Lynch-Paterson 的理论(见 3.1.3 节)，在异步通信的分布式系统中，只要有一个拜占庭缺陷的进程，就不可能找到一个共识算法，可同时满足上述要求的一致性、正确性和可结束性。也就是说，不能同时满足安全性(Safety)和活性(Liveness)。

在实际情况下，根据不同的假设条件，有很多不同的共识算法被设计出来，这些算法各有优势和局限。算法的假设条件有以下几种情况：

故障模型：非拜占庭故障/拜占庭故障。

通信类型：同步/异步。

通信网络连接：节点间直连数。

信息发送者身份：实名/匿名。

通信通道稳定性：通道可靠/不可靠。

消息认证性：认证消息/非认证消息。

通常实用的共识算法，如果按照严格的 FLP 异步通信的假设，往往只满足安全性而牺牲活性。也就是说，大部分实用的异步通信网络下的共识算法，往往不能保证在有限的时间内，总能达成共识。

3.1.3　Fisher-Lynch-Paterson 定理

1983 年，Fischer、Lynch 和 Paterson 三位科学家给出以下定理。

Fischer-Lynch-Paterson(简称 FLP)定理：在一个多进程异步系统中，只要有一个进程不可靠，那么就不存在一个协议，此协议能保证有限时间内使所有进程达成一致。

FLP 定理不考虑拜占庭故障，而仅仅是一般的宕机故障。它假设异步通信是可靠的，也就是说异步通信能成功传递消息，而且能保证只传递一次消息而不会传递重复消息。在这里，异步通信的意思是对进程处理速度和发送消息的延迟不做任何假设，也就是说进程处理速度有可能非常慢，消息的延迟也有可能非常大。同时假设进程没有一个同步的时钟可以访问，也就是说不能依靠超时来做判断。最后，也假设没法通过监控来判断进程是否出现故障。也就是说，没法分别一个进程是出现故障还是运行得非常慢。

先给出 FLP 定理中共识协议的一些基本定义和假设。

定义 1：一个共识协议 P 是一个有 N 个进程($N \geqslant 2$)的异步通信系统。每个进程 p 有一个一位的输入寄存器 xp，一个输出寄存器 yp，它们的取值范围是$\{b,0,1\}$。每个进程有无限的内存。p 的内部状态由它的输入/输出寄存器、内存、程序计数器决定。p 是一个基于转换函数的确定性(deterministic)的进程，也就是说，如果收到一个消息，p 的状态就会转到一个确定的状态。初始状态(initial state)是不包含输入寄存器的所有内部状态。输出寄存器的值如果是 0 或者 1，这个视为决定状态。如果进程 p 进入决定状态，那么转换函数就不能再改变输出寄存器，也就是说，输出寄存器是仅“一次写”。当所有的非故障进程都进入一个相同值(0 或 1)的决定状态，那么系统就达成了共识。整个 P 系统是由每个进程的转换函数和其相应的输入寄存器的初始值决定的。

定义 2：系统中的进程通过互相发消息来进行通信。一个消息可以表示成(p,m)，其中 p 是接收消息的进程，m 是消息的值，$m \in \boldsymbol{M}$，$\boldsymbol{M}$ 是一个固定的消息集合。消息在发送后并在没有接收前，存在消息缓存中。消息的操作支持以下两个抽象函数。

send(p,m)：把(p,m)放入消息缓存；

receive(p)：把(p,m)从消息缓存中删除，同时返回 m，在这种情况下(p,m)已经送到；否则返回特殊的空集$\varnothing$，同时不改缓存。

由此可见，虽然每个进程的转换函数是决定性的，但在异步通信环境下，由于存在不确定的延迟，接收消息是非确定性的。也就是说，即使有消息在缓存，调用 receive(p)也可能多次返回空集$\varnothing$。

定义 3：共识系统的配置(configuration)由所有进程的内部状态，加上消息缓存组成。初始配置(initial configuration)是每个进程从初始状态开始，同时消息缓存是清空的状态。

定义 4：一个步骤(step)是一个单一进程的一次执行，把一个配置状态转到另一个配置状态。假设 C 是一个配置，一个步骤的执行应有两阶段。第一阶段是进程 p 调用 receive(p)接收信息 $m \in \boldsymbol{M} \cup \{\varnothing\}$。然后取决于 p 的内部状态和 m，p 进入一个新的内部状态，并发有限的消息给其他进程。因为 p 是确定性的，所以步骤完全取决于 $e=(p,m)$，e 称一个事件。$e(C)$表示结果配置，同时我们称 e 可以作用于 C。需要了解的是事件$(p,\varnothing)$也总可以作用于 C，所以总是有可能一个进程会走另一个步骤。

定义 5：一个从 C 的计划(schedule)是一个可以作用于 C 的有限或无限的事件序列 u。相关的序列步骤叫作一个运行(run)。如果 u 是有限的，我们用 $a(C)$来表示结果配置，那么可以说结果配置可以由 C 到达(reachable)。一个从初始配置(initial configuration)能到达的配置叫作可访问(accessible)。以后，所有提到的配置假设都可访问。

计划(schedule)有互换性(commutativity)，见下面引理。

引理 1：假设从一个配置 C，两个不同的计划 σ_1 和 σ_2 能分别作用于 C，并分别得到结

果 C_1 和 C_2。如果 σ_1 和 σ_2 没有相同的进程，那么 σ_2 可以作用于 C_1，σ_1 可以作用于 C_2，同时结果配置是一样的。

证明：由于 σ_1 和 σ_2 没有重合的进程，且每个进程都是确定性的，其每一步骤只改变一个进程的内部状态，而且改变的状态也是确定的，另外根据 FLP 的假设，系统没有同步的时钟可以访问，因此状态的改变与时间无关。因此，$\sigma_1+\sigma_2$ 的结果状态和 $\sigma_2+\sigma_1$ 的状态将一致，如图 3.4 所示。

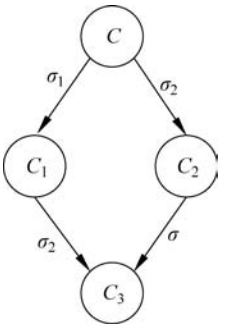

图 3.4　计划交换性

FLP 定理的作者采用反证法来证明 FLP 的正确性。他们首先假设在有一个进程出故障的情况下，异步系统能够达成共识状态，然后推出几个自相矛盾的引理。其中一个引理如下。

引理 2：任何一个异步通信共识协议 P，都有一个共识状态不确定的初始配置(initial configuration)。

这里所说的共识状态，指的是系统中所有非故障进程的 y 寄存器都进入全是“0”或者全是“1”的状态。如果全是“0”，FLP 定理作者把该状态定义成“0-valent”，如果全是“1”，则是“1-valent”。不确定状态，也就是说有些是“0”，有些是“1”，则定义成“bi-valent”，也就是说无法达成共识。

FLP 定理的作者采用反证法证明引理 2，其证明方式比较抽象难懂，这里用通俗语言来描述其证明。首先假设协议 P，只有一个共识状态确定的初始配置，要么是“0-valent”，要么就是“1-valent”。那么在所有可能的初始配置中，能够找到一个“0-valent”的初始配置，我们把它叫 C_0 的初始配置，C_0 通过一些步骤最后进入“0-valent”的决定状态，同时找到另外一个除进程 p 之外状态和 C_0 完全一样的 C_1 初始配置，该配置通过一些步骤最后进入“1-valent”状态。C_0 和 C_1 只是在进程 p 的状态不一样。如果进程 p 出故障，也就是说进程 p 不能接收或发送消息，那么 C_0 和 C_1 进入决定状态所运行的步骤就会一样。也就是说，在一个系统中，存在着一个运行计划(schedule)，有可能分别进入“0-valent”和“1-valent”的状态，因此和假设矛盾，从而证明了引理 2 的正确性。

在证明了引理 2 的基础上，FLP 定理作者更进一步证明引理 3。

引理 3：让 C 是 P 的一个不确定(bi-valent)配置，让 $e=(p,m)$ 是一个可以作用于 C 的事件，让 ζ 是一组不运行 e 而能从 C 达到的配置，同时让 $\partial=e(\zeta)=\{e(E)\mid E\in\zeta, e$ 作用于 E，那么 ∂ 包含有一个不确定(bi-valent)的配置。

FLP 作者也是采用反证法来证明该引理，其证明过程比较烦琐，这里不做展开。其证明过程主要采用了引理 2 的证明技巧，同时加上引理 1 即运行计划的可交换性，得出自相矛盾的结果。

在引理 2 和引理 3 正确的基础上，可以容易地用反证法来导出 FLP 定理的正确性。

FLP 定理是共识算法中一个非常重要的基础性定理。该定理的出现在共识算法领域具有里程碑的意义，它推翻了过去很多研究者在共识算法领域的一些结论。在使用 FLP 定理的时候，我们一定要了解它的系统模型，也包括它的假设条件。在 FLP 的假设条件中，异步通信系统没有对进程接收消息、处理消息、回复消息的时间做一个上限。所以无法判断一个进程是出了故障，还是因为长时间处理而无回应。FLP 中的故障不是拜占庭故障，而仅仅是宕机类的故障。理解 FLP 定理的关键在于，尽管构成共识系统 P 的每个进程 p 的状态转换函数是确定性的(deterministic)，但是触发状态转换函数的消息接收和处理是异步

的，也就是不确定的(un-deterministic)，因此，不可能在有一个缺陷进程的情况下达成整个系统的确定共识状态。

3.1.4 CAP 定理

在 2000 年的分布式计算原则大会(Symposium on Principles of Distributed Computing 2000)，Eric Brewer 做了一个主旨演讲，该主旨演讲分析了当时由于数据量的增长，新出现的一种数据库模式(BASE)。BASE 的意思是基本可用、软状态、最终一致性(Basically Available，Soft-State，Eventual Consistency)，这种新的模式是为了解决传统基于 ACID (Atomicity，Consistency，Isolation，Durability)的数据库在分布式环境扩展性差而产生的。Brewer 的分析结果中提出了 CAP(Consistency，Availability，Partition Tolerance)定理的假设。

CAP 定理：一个分布式数据库系统有三个需要的属性：①对网络分隔的容忍(Partition Tolerance)；②一致性(Consistency)；③可用性(Availability)。在任意一个数据共享系统，最多只能在三个属性中选择满足两个属性。

Brewer 当时并没有给出证明，而是在两年后由 Gilbert 和 Lynch 给出了正式的证明。在做该定理的证明前，Gilbert 和 Lynch 先对三个属性进行了严格的定义。

Consistency(数据对象原子性)：在所有的操作中，必须有一个完整的顺序存在，使得每个操作就像在单个实体上完成。在一个分布式多节点环境中，这意味着所有发生在一个写操作完成之后的读操作都必须返回该写操作写入的值。

Availability：每个非故障节点在收到请求后必须有回应。这意味着服务用的任何算法最后必须能终止。

Partition Tolerance：可以容忍网络丢失从一个节点发到另一节点任意多的消息。

Gilbert 和 Lynch 采用反正法来证明 CAP 定理。首先假设所有的条件(Consistency，Availability，Partition Tolerance)能同时成立。因为任意网络至少两个节点可以分别处在两个非空、不重合的网络节点集合$\{G_1,G_2\}$。一个原子对象 o 有一个初始值 v_0。我们定义 α_1 是一个执行步骤，该步骤把在 G_1 的 o 的值改成 v_1，$v_1 \neq v_0$。假设 α_1 是唯一的一个用户请求。另外，假设 G_1 和 G_2 中的节点互相没有收到来自对方的消息。因为可用性的要求，我们知道 α_1 是完成的，也就是说 o 现在在 G_1 有 v_1 的值。类似地，α_2 是一个在 G_2 的一个单一的读 o 的执行步骤，在 α_2 的执行过程中，G_1 和 G_2 之间没有消息的传递。由于可用性的要求，我们知道 α_2 也是完成的。如果我们执行 α_1 和 α_2，G_2 只能看到 α_2(因为 G_2 没有收到任何从 G_1 来的消息，也看不到 α_1 的请求)，那么从 α_2 读来的结果还是 v_0。但由于 α_2 读的操作是在 α_1 写的操作完成之后开始的，一致性的要求被违反。这就证明了我们不能同时保证三个属性都满足。

CAP 定理对后来的分布式系统产生了深远的影响。它告诉人们，在三个属性中必须做取舍，只能同时满足两个属性。而分布式在网络环境下，由于分区容错是一个常态。因此在设计分布式系统的时候，需要在可用性和一致性上做权衡。AWS 后面推出的分布式数据库 Dynamo，就是舍掉一致性的一个例子。而谷歌的 GFS，则是在保证一致性前提下，舍弃一些可用性。

CAP 定理和 FLP 定理虽然在某些方面有些相似，都是研究在分布式系统中的一些不能解决的问题，但并不直接相关。也就是说，两个定理并不是等价关系。它们之间的区别在于：FLP 的假设是网络的异步通信是可靠的，消息虽然会有延迟，但不会丢失。CAP 中的网络分区则会导致消息丢失。另外，FLP 中的故障节点完全从网络中分隔，不会响应任何请求；而 CAP 中的可用性要求系统能响应请求。FLP 是关于分布式系统的共识问题，这个和 CAP 中的一致性有些相似，但 CAP 主要讨论关于原子性数据存储的一致性，以及和可用性的关系。

分布式共识算法的设计需要考虑 FLP 和 CAP 定理的限制，因为分布式系统都会有对一致性、可用性、可终止性的要求，重要的是，在共识协议的设计上对这些属性的取舍，还有就是在假设条件上规避 FLP 不可能达成共识的一些前提条件，如采用同步或半异步通信方式。

3.2 强一致性非拜占庭共识算法

在很多分布式系统场景下，并不需要解决拜占庭将军问题。也就是说，在这些分布式系统的实用场景下，其假设条件不需要考虑拜占庭故障，而只是处理一般的宕机恢复故障。在这种情况下，采用非拜占庭容错共识协议往往效率更高。这种共识协议也应用广泛。在这部分，重点介绍三个处理宕机恢复故障的协议：Viewstamped Replication、Paxos 和 Raft 协议。

3.2.1 Viewstamped Replicaton

Viewstamped Replication（以下简称 VR）是于 1988 年由 Brian Oki 在导师 Barbara Liskov 的指导下，在其博士论文中发表的第一个分布式共识算法，主要用于在分布式系统状态复制服务。

1. VR 正常流程与视图更换

VR 基于状态机复制技术，在很多节点运行副本（replicas），维护状态并给客户端提供服务。VR 的特点如下：

（1）VR 主要是一个复制协议，但它也能提供共识功能。

（2）VR 只处理宕机故障，一个节点应该是完全运行良好，或者是停机。

（3）VR 的运行环境是一个异步网络环境，消息可能丢失，接收顺序可能不能保证，同一消息可能会重复接收。

VR 副本的总数设成 $2f+1$ 个，这个是在异步通信网络下存在宕机故障的最少副本配置。VR 保证在不超过 f 个副本出故障的情况下整个系统的可靠性和可用性，这个可以简单地通过投票中占多数的非故障节点来证明其可靠性和可用性。这个多数组 $f+1$ 个副本通常称为“法定人数（quorum）”。

VR 的架构如图 3.5 所示。

（1）客户端的代码运行在 VR 代理。

（2）VR 代理和副本通信，将副本的计算结果返回给客户端。

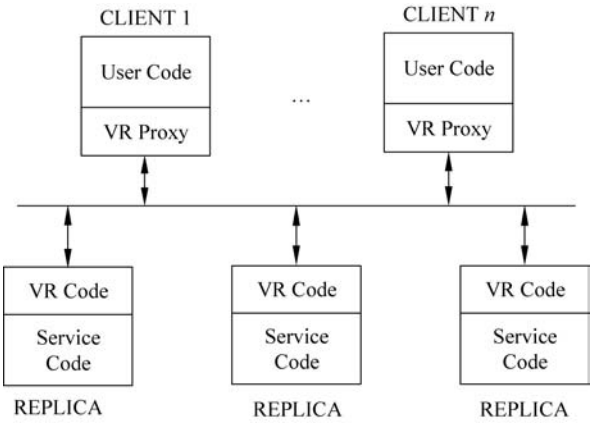

图 3.5 VR 架构

(3) VR 代码接受客户端的请求,调用服务代码执行协议。

(4) 服务代码返回结果给 VR 代码,VR 代码再把结果返回给客户端上的 VR 代理。

VR 共识协议的目的是为了保证在客户端并行发请求,同时存在着副本故障的情况下,所有非故障副本必须能按同一个顺序来执行。VR 中的副本分为主副本(primary)和从副本(secondary)。主副本决定客户请求执行的顺序,从副本执行主副本选择的执行顺序。如果主副本出故障,VR 允许不同的副本来替代主副本。这种情况下,VR 系统会发生视图转换,每个视图有一个副本做主副本的角色。其他副本都会关注主副本,如果主副本出现故障,其他副本可以通过一个"改变视图"的步骤来选举新的主副本。

因此,VR 协议有三种场景。第一个是正常情况下处理用户的请求;第二个场景是视图改变时新主副本的选举;第三个是故障恢复后的副本重新加入。

先来看 VR 副本的一些状态参数。VR 的配置(configuration)保存所有副本($2f+1$)的 IP 地址。参数 replicate_number 保存指向副本的索引,当前视图号保存在 view_number,初始值是 0。当前状态保存在 status,取值是 NORMAL、VIEW-CHANGE 和 RECOVERING。op-number 保存当前的请求号,初始值为 0。日志(log)保存一个存放 op-number 的数组,按收到的请求顺序存放。commit_number 是最近提交的 op-number。client_table 是一个保存每个客户最新请求的表,里面包括给客户的回复。

主副本的视图是通过视图号和配置表计算出来,不需要保存主视图的标志。每个副本按 IP 地址来排序,最小的 IP 地址定为 1,逐渐增加。

在客户端的代理会维护一些状态,包括配置(configuration)当前的视图号以跟踪主副本、客户端的 ID 和客户请求号码。

1) 正常情况的 VR 协议

图 3.6 给出了 VR 正常的流程。

具体步骤如下:

(1) 客户端发一个请求(Request)消息给主副本,让其执行某个操作,包括客户端 ID(client_id)和请求号 request_number。

(2) 主副本会检查 client_table,如果 request_number 小于在 client_ table 表中的请求

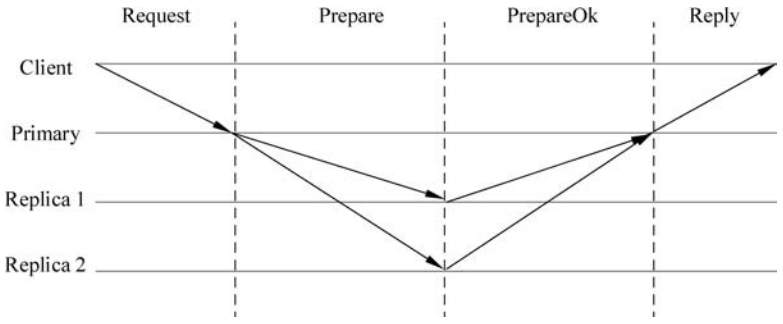

图 3.6　VR 流程

号，主副本会忽略该请求，并发送回复；如果该请求是最近被执行的一个请求，则会拒绝执行该请求。

(3) 主副本在 op-number 增加 1，把请求加到日志(log)并用新的请求号更新 client_table。然后发一个 Prepare 消息给其他副本，连同当前的视图号 view_number、op-number、客户的消息和 commit_number(最近提交的 op_number)。

(4) 副本按顺序来接收消息，然后把消息加进日志，更新 client_table 并发一个 PrepareOk 的消息给主副本。这个消息表示该操作以及以前的操作都准备好了。

(5) 主副本收到 f 个副本的回复之后才提交操作，把 commit_number 加 1。当它确认所有在此之前的操作都执行后，它会调用服务代码执行当前的操作。一个 Reply 的消息会发给客户端，包含有 view_number、request_number 以及服务代码返回的结果。

(6) 其他副本执行请求时，只要保证在此请求之前的请求都被执行，然后执行该请求，在 commit_number 上增加 1，更新 client_table，但并不需要直接回复给客户端，只有主副本回复客户端。

(7) 如果一个客户端在一定时间内没有收到回复，它将重发请求给所有的副本。如果 VR 已经转到一个新的视图，这个消息将最后到达新的主副本。副本会忽略客户请求，只有主副本处理客户请求。

2) 视图转换协议

视图转换是在主副本出现异常情况时需要更换主副本的协议。该协议也是其他很多共识机制常用的异常处理机制，例如后面的 PBFT 算法的视图更换，也是采用相同的协议。

副本监控主副本，它们预期是间断性地能接收到主副本消息。一般情况下主副本发 Prepare 消息，但如果是空闲时段(没有客户请求)，它发 Commit 消息给副本。如果在一个超时发生而没有接收到主副本消息的情况下，副本会执行一个视图转换以选择新的主副本。主副本是通过轮流选出。每个副本有一个单一的 IP 地址，下一个主副本是具有最小 IP 地址并正常工作的副本。在 VR 组里的成员都能预计哪一个副本会成为下一个主副本。

视图转换的算法如下：

(1) 一个副本注意到需要更换视图，它把 view_number 增加 1，把状态改成 view_change，同时发一个 START-VIEW-CHANGE 消息。每个副本根据自身的定时器，或者因为接收到其他副本发送的 START-VIEW-CHANGE 或 DO-VIEW-CHANGE 中高于自身 view-number 的请求来决定是否更换视图。

(2) 当一个副本接收到 f 个 START-VIEW-CHANGE 消息回应它的 view-number 后,它发 DO-VIEW-CHANGE 给将要成为主副本的节点。该消息包含副本的状态:日志(log)、最近的 op_number、commit_number 和上次状态是正常状态的视图号。

(3) 新的主副本等待接收 $f+1$ 个 DO-VIEW-CHANGE 来自其他副本的消息(包括自己)。然后它根据副本的消息将自己状态更新,它把自己的号码作为消息中的 view-number,把状态改成正常(normal)。它发一个 STARTVIEW 消息,连同最新的包括日志(log),commit-number,和 op-number 给所有的副本节点。

(4) 主副本节点可以接收客户端请求。它执行请求并发回复给客户端。

(5) 当副本收到 STARTVIEW 消息,它们将根据消息来更新自身的状态。状态更新后,它们对在日志(log)中没有提交的操作,发 PrepareOK 消息给新主副本。然后执行这些操作以和新主副本同步。

(6) 当主副本接收到 f 个 PrepareOK 消息后,主副本将提交操作,并调用服务代码。

2. VR 副本节点故障恢复协议

当一个副本从宕机恢复时,在它恢复到宕机之前的状态前,它不能参与处理请求和视图转换,否则系统就会出错。VR 副本需要记住它所执行的操作。VR 副本节点的状态是保存在别的副本节点上,不需要通过磁盘存储来保持状态。VR 副本的状态可以通过恢复协议来获得。

当一个节点从宕机恢复的时候,它把它的状态置为恢复(RECOVERYING),同时执行恢复协议。恢复协议步骤如下:

(1) 恢复节点发一个 RECOVERY 消息给所有其他副本节点,连同一个随机数。

(2) 收到消息的副本节点,如果自身是处于正常状态,它会回复恢复节点一个 RECOVERY-RESPONSE 的消息,包含它的视图号码和收到的随机数。如果是主副本节点,它将把自己的日志(log)、op-number 和 commit-number 一同发回。

(3) 当恢复副本收到 $f+1$ 个 RECOVERY-RESPONSE 消息,包括一个从主副本收到的消息,它将更新自己的状态并把状态设置成正常状态。随机数的作用是保证恢复副本节点只接收对这次恢复的 RECOVERY-RESPONSE 消息,而不是以前的。

3. VR 共识协议小结

VR 是第一个正式发表的共识协议,它主要用于主从节点状态的复制,同时也提供一个可行的共识协议。它处理的是在异步通信下存在宕机恢复故障的一致性共识问题。只要在总数不超过一半的故障节点情况下,VR 系统能够达成共识。需要注意的是,VR 虽然也是一个异步通信环境的共识协议,但是与 FLP 中的异步通信假设还是不一样。在 FLP 的异步通信假设中,没有一个同步的时钟可以访问,也就是说不能依靠超时来做判断。同时,假设没法通过监控来判断进程是否出现故障,因此有 FLP 不可形成共识的结论。而 VR 则可以依靠超时来判断一个节点是否出现故障。因此,VR 可以达成共识与 FLP 的不可能达成共识的结论并不冲突,只是各自的假设条件不一样。如果严格按照 FLP 的条件来衡量,那么 VR 可以保证安全性(safety),但 VR 不能保证活性(liveness),也就是说不能保证在任何情况下,不出现不断更换视图的情况而无法进入正常共识流程。

3.2.2 Paxos 共识算法

与 VR 算法类似，Paxos 共识算法也是一个在异步通信环境的共识算法，能容忍半数以下的宕机恢复(非拜占庭故障)节点并达成共识。Paxos 由 Leslie Lamport 在 1998 年发表，成为强一致性共识算法的代表。大部分后来的强一致共识算法都是 Paxos 的变种。

Paxos 共识系统中有五种角色，每个节点可以同时扮演多种角色。

(1) Client(客户端)：客户端向分布式系统发出请求并等待回复，例如更新一个分布式文件系统。

(2) Proposer(提议者)：一个 Proposer 是接收客户端请求并按其要求提议系统接受特定值的节点。在任何时候可以有多个提议者。

(3) Acceptor(接受者)：一个从提议者那里接收提议的投票节点。接受者可以接受或拒绝提议。

(4) Learner(学习者)：一个不参与决策流程，但是必须从提议者和接受者那里了解最后的选择结果，然后采取行动，例如执行请求、返回结果给客户端。

(5) Leader(领导者)：在所有提议者中选出一个领袖，由该领袖来主导提议的发送、投票和最后选择。

Paxos 的提议 Proposal(x,n)由提议值 x 和一个提议号 n 构成。当一个提议者发出一个新的提议，它会选择一个更大的唯一的提议号。

一个接受者看到一个比任何它原有接收到的提议都大的 n，就会接受该提议。接受者可以接收任意多的提议。接收到的提议不一定会被选择，可以选择多个提议，但这些选择的提议中，提议值必须相同。Paxos 正常流程如图 3.7 所示。

Paxos 协议分以下两个阶段。

1. 准备阶段(Prepare)

(1) 提议者发一个 Prepare(x,n)消息给接受者的多数集，其中 n 必须比提议者以前用过的提议号大。该消息告知所有的接受者希望提议 Propose(x,n)。多数集由提议者选择，保证至少有 $f+1$ 个接受者接收(假设接受者至少 $2f+1$，f 为可能出现的最多故障数)。

(2) 接受者收到该消息后，如果 n 是大于它原来接收到的所有的提议号，接受者返回一个过去接收到的提议 Propose(y,m)给提议者，m 是过去接收到的最大的提议号。同时保证将不接收任何小于 n 的提议。如果接受者过去从来没收到提议，则返回(ϕ,0)。如果 n 不是最大的提议号，接受者可以不回答提议者，也可以发送拒绝消息(Nack)表示不同意对该提议的共识，否则接受者将把 n 作为最大提议号，x 作为最后收到的提议值记录下来。

2. 提议阶段(Propose)

(1) 如果提议者至少收到 $f+1$ 个回复，它将发正式提议。如果前面只接收到 Acc(ϕ,0)，它只发它自己提议的 x 值，否则的话将采用接收到的最大的提议号 m 中的 y 值。在发出的提议中，它还是采用自己的提议号 n，也就是说发出 Propose(y,n)的消息给接受者的多数集。

(2) 接受者收到提议后，如果它在过去没有向其他提议者保证过只接收大于 n 的提议，

那该接受者必须接受该提议。接受者把 y 值记录下来，并发一个接受 Ack(y,n)消息给提议者(Proposer)和学习者(Learner)，表示接受该提议。在表示接受提议时，接受者实际上承认该提议者为 Leader(领袖)。如果接受者在过去曾经向其他提议者保证过只接受大于 n 的提议，那么该接受者将忽略这次收到的提议。

(3) 提议者在收到 $f+1$ 个 Ack(y,n)回复，则该轮共识达成。否则需要用一个更大的提议号来发起新一轮共识。如果多个提议者发出互相冲突的提议，也会使得共识难以达成。

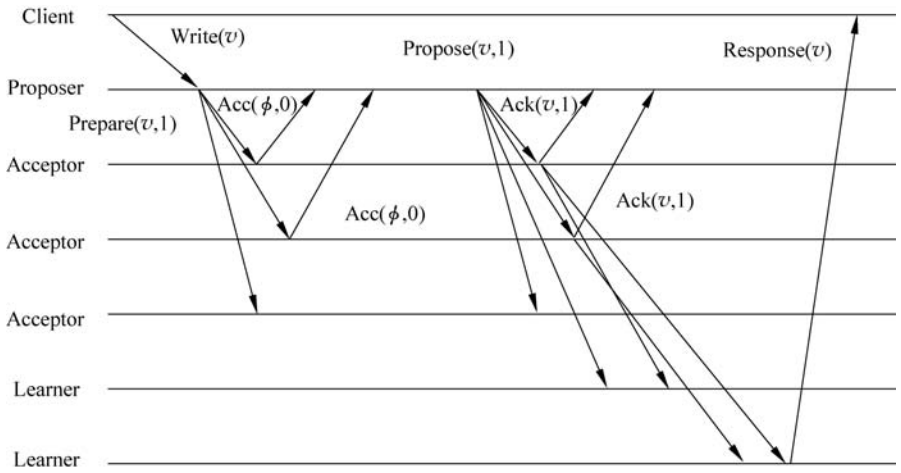

图 3.7 Paxos 正常流程

3. Paxos 共识算法小结

Paxos 是强一致、非拜占庭故障的典型共识算法，它在异步环境下能保证在故障节点不超过二分之一的情况下实现共识的安全性(Safety)，也就是说，能保证如果一个值被选择，其他节点只能选择相同的值。但却不能保证活性(Liveness)，这意味着在某些特殊情况，可能不能在有限时间内达成共识。例如，由于网络分隔的原因使得提议者得不到多数接受者的回复的情况，因此，Paxos 和 FLP 并不冲突。

Paxos 协议和 VR 非常类似，不同的地方在于 VR 是一个被动状态机复制协议，它关注的是相同顺序的操作指令被各副本按顺序执行，VR 是由主节点主导，只有主节点有决定性(Deterministic)状态机，从节点“被动”听从主节点指令的复制协议；而 Paxos 则更多是一个“主动”状态机复制共识协议，各节点自带决定性状态机，各节点同意同一个值或同一个执行顺序。

3.2.3 其他类 Paxos 共识协议

Paxos 共识协议是一个典型的高效非拜占庭故障的共识协议，但 Paxos 的缺点是协议比较复杂，比较难以理解，特别是在实施上会碰到不少的问题。因此后面很多共识协议的设计是基于 Paxos，但目的是设计适合于不同场景、比较容易理解、容易实现的共识协议。例如，像 Raft、Chubby、Zookeeper、etcd 等都是应用比较广泛的基于 Paxos 的共识协议。下面重点介绍 Raft，它在联盟链和私有链上得到比较广泛的应用。

1. Raft 共识协议

Raft 协议是近来比较流行的一个基于 Paxos 的共识协议。Raft 最初是一个用于管理复制日志的共识算法，也是基于状态机复制的协议。它是一个为真实世界应用建立的协议，主要注重协议的落地性和可理解性。Raft 是在非拜占庭故障下达成共识的强一致协议。

在区块链系统中，使用 Raft 实现记账共识的过程可以描述如下：首先选举一个首领 Leader，接着赋予 Leader 完全的权力管理记账。Leader 从客户端接收记账请求，完成记账操作，生成区块，并复制到其他记账节点。有了 Leader 简化了记账操作的管理。例如，Leader 能够决定是否接收新的交易记录项而无须考虑其他的记账节点，Leader 可能失效或与其他节点失去联系，这时，系统就会选出新的 Leader。

给定 Leader 方法，Raft 将共识问题分解为三个相对独立的子问题：

(1) Leader 选举：现有的 Leader 失效时，必须选出新 Leader。

(2) 记账：Leader 必须接收来自客户端的交易记录项，在参与共识记账的节点中进行复制，并使其他的记账节点认可交易所对应的区块。

(3) 安全：若某个记账节点对其状态机应用了某个特定的区块项，其他的服务器不能对同一个区块索引应用不同的命令。

1) Raft 基础

一个 Raft 集群通常包含 5 个服务器，允许系统有两个故障服务器。每个服务器处于三个状态之一：Leader、Follower 或 Candidate。正常操作状态下，仅有一个 Leader，其他的服务器均为 Follower。Follower 是被动的，不会对自身发出请求而是对来自 Leader 与 Candidate 的请求做出响应。Leader 处理所有的客户端请求(若客户端联系 Follower，则该 Follower 将转发给 Leader)。Candidate 状态用来选举 Leader。

Raft 阶段主要分为两个，首先是 Leader 选举过程，然后在选举出来的 Leader 基础上进行正常操作，如日志复制、记账等。

2) Leader 选举

当 Follower 在选举超时时间范围内未收到 Leader 的心跳消息，则转换为 Candidate 状态。为了避免选举冲突，这个超时时间是一个 150～300ms 之间的随机数。

一般而言，在 Raft 系统中，Leader 的选举步骤如下：

(1) 任何一个服务器都可以成为一个候选者 Candidate，它向其他服务器 Follower 发出要求选举自己的请求；

(2) 其他服务器同意了，发出 OK。注意，如果在这个过程中，有一个 Follower 宕机，没有收到请求选举的要求，此时候选者可以自己选自己，只要达到 $N/2+1$ 的大多数票(N 是所有节点的总数)，候选人还是可以成为 Leader 的。

(3) 这样，这个候选者就成为了 Leader 领导人，它可以向选民也就是 Follower 们发出指令，如进行记账。

(4) 以后通过心跳进行记账的通知。

(5) 如果一旦这个 Leader 宕机了，那么 Follower 中又有一个成为候选者，发出邀票选举。

(6) Follower 同意后，其成为 Leader，继续承担记账等指导工作。

3）记账过程

Raft 的记账过程按以下步骤完成：

（1）假设 Leader 领导人已经选出，这时客户端发出增加一个交易记录的记账要求。

（2）Leader 要求 Follower 遵从他的指令，都将这个新的交易记录追加到他们各自的账本中。

（3）大多数 Follower 服务器将交易记录写入账本后，确认追加成功，发出确认成功信息。

（4）在下一个心跳中，Leader 会通知所有 Follower 更新确认的项目。

对于每个新的交易记录，重复上述过程。

如果在这一过程中，发生了网络通信故障，使得 Leader 不能访问大多数 Follower 了，那么 Leader 只能正常更新它能访问的那些 Follower 服务器。而大多数的服务器 Follower 因为没有了 Leader，他们重新选举一个候选者作为 Leader，然后这个 Leader 作为代表与外界打交道，如果外界要求其添加新的交易记录，这个新的 Leader 就按上述步骤通知大多数 follower，如果这时网络故障修复了，那么原先的 Leader 就变成 Follower，在失联阶段这个老 Leader 的任何更新都不能算确认，都回滚，接受新的 Leader 新的更新。

2. 其他类 Paxos 共识协议

其他基于 Paxos 的共识算法有 Chubby、Zookeeper、etcd 等。这些共识算法广泛用于各种分布式系统。例如，谷歌公司的 GFS 和 BigTable 使用 Chubby 的分布式锁协议；Yahoo 公司的 Hadoop 系统和 Openstack、Mesos 等采用了 Zookeeper；Kubernetes 和 CoreOS 采用了 etcd 协议等。除了超级账本的 Fabric1.0 用了基于 Zookeeper 的 Kafka 做排序引擎外，这些共识协议用在区块链的场景目前不是很多见，因此我们这里不做详细叙述。

3.2.4 强一致性非拜占庭共识算法小结

共识问题是分布式系统中一个最重要的问题。在很多场景，包括服务器副本复制、日志复制、同步服务、分布式锁、领袖选举、元数据管理、消息队列中都会有共识的需求。分布式系统之所以能协同工作，归根结底有赖于共识协议的协调。拜占庭将军问题是一个形象的比喻，说明在一个同步通信系统中，当有超过三分之一系统节点总数的节点出现拜占庭故障时，该分布式将不可能达成共识。而 FLP 定理告诉我们在异步通信情况下，即使有一个非拜占庭故障（如宕机、宕机恢复故障）的存在，就不能保证系统能在有限的时间内达成共识，也就是说，不能同时满足安全性和活性的要求。分布式系统中著名的 CAP 定理也从另一个角度说明，分布式系统不能同时满足一致性、可用性和网络分区容错性，只能在三者中取其二。

因此，在设计共识算法的时候，要根据实际情况来规避以上的限制。大部分的分布式共识算法，都是在保证安全性，也就是一致性和正确性的前提下牺牲活性。

Viewstamped Replication 是第一个公开发表的共识协议。后面的共识协议大多基于 Paxos。Paxos 是一个高效而复杂的协议，比较难以理解，也难以实现。因此后来出现了各种不同的变种。例如，谷歌公司的 GFS、Bigtable 就采用了基于 Paxos 的 Chubby 的分布式锁协议；Yahoo 公司的 Hadoop 系统采用了类似 Paxos 协议的 Zookeeper 协议。Raft 也是

为了避免 Paxos 的复杂性而专门设计成易于理解的一个分布式一致性算法。大部分强一致性的非拜占庭故障共识协议都能容忍少于系统总节点二分之一的宕机恢复故障。

在私有链和联盟链的场景下，通常共识算法有强一致性要求，同时对共识效率要求高。另外，一般安全性要比公有链场景高，一般来说不会经常存在拜占庭故障。因此，在一些场景下，可以考虑采用非拜占庭容错的分布式共识算法。在 Hyperledger 的 Fabric 项目中，未来共识模块被设计成可插拔的模块，支持 PBFT、Raft 等共识算法。目前在区块链项目中比较常见的非拜占庭容错共识算法是 Raft。

3.3 强一致性拜占庭容错共识算法

FLP 定理给出了在异步通信系统中对分布式共识的理论限制，但由于同步通信的可扩展性小，大部分分布式系统不能依靠同步通信，否则性能和效率将非常低。因此寻找一种实用的在异步通信环境下解决拜占庭将军问题的算法一直是分布式计算领域中的一个重要问题。

在区块链网络中，由于应用场景的不同、所设计的目标各异，不同的区块链系统采用了不同的共识算法。一般来说，在私有链和联盟链情况下，对一致性、正确性有很强的要求，要采用强一致性的共识算法。而在公有链情况下，对一致性和正确性通常没法做到百分之百，通常采用最终一致性(Eventual Consistency)的共识算法。

下面先来介绍适合私有链和联盟链场景的拜占庭容错系统。

区块链网络的记账共识和拜占庭将军问题是相似的。参与共识记账的每一个记账节点相当于将军，节点之间的消息传递相当于信使，某些节点可能由于各种原因而产生错误的信息传达给其他节点。通常，这些发生故障的节点被称为拜占庭节点，而正常的节点即为非拜占庭节点。

拜占庭容错系统是一个拥有 n 台节点的系统，整个系统对于每一个请求，满足以下的条件：

(1) 所有非拜占庭节点使用相同的输入信息，产生同样的结果。

(2) 如果输入的信息正确，那么所有非拜占庭节点必须接受这个信息，并计算相应的结果。

与此同时，在拜占庭系统的实际运行过程中，还需要假设整个系统中拜占庭节点不超过 m 台，并且每个请求还需要满足两个指标：

① 安全性：任何已经完成的请求都不会被更改，它可以被以后的请求看到。

② 活性：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。

拜占庭系统普遍采用的假设条件包括：

(1) 拜占庭节点的行为可以是任意的，拜占庭节点之间可以共谋。

(2) 节点之间的错误是不相关的。

(3) 节点之间通过异步网络连接，网络中的消息可能丢失、乱序并延时到达，但大部分协议假设消息在有限的时间里能传递到目的地(该假设是避免 FLP 的不可能共识条件)。

(4) 服务器之间传递的信息，第三方可以探测到，但是不能篡改、伪造信息的内容和验证信息的完整性。

3.3.1 实用的拜占庭容错系统

原始的拜占庭容错系统由于需要展示其理论上的可行性而缺乏实用性，另外需要额外的时钟同步机制支持，算法的复杂度也是随节点增加而指数级增加。Castro and Liskov 在1999 年提出实用拜占庭容错系统(Practical Byzantine Fault Tolerance，PBFT)，降低了拜占庭协议的运行复杂度，从指数级别降低到多项式级别(Polynomial)，使拜占庭协议在分布式系统中应用成为可能。

PBFT 是一类状态机拜占庭系统，要求整个系统共同维护一个状态，所有节点采取的行动一致。为此，需要运行三类基本协议，包括一致性协议、检查点协议和视图更换协议。我们主要关注支持系统日常运行的一致性协议。

一致性协议要求来自客户端的请求在每个服务节点上都按照一个确定的顺序执行。这个协议把服务器节点分为两类：主节点和从节点，其中主节点仅一个。在协议中，主节点负责将客户端的请求排序，从节点按照主节点提供的顺序执行请求。每个服务器节点在同样的配置信息下工作，该配置信息被称为视图，主节点更换，视图也随之变化。

一致性协议至少包含 5 个阶段：请求(Request)、序号分配(Pre-Prepare)、交互(Prepare)、序号确认(Commit)和响应(Reply)等阶段。

PBFT 的一致性协议正常流程如图 3.8 所示。PBFT 系统通常假定故障节点数有 f 个，而整个服务节点数为 $3f+1$ 个。每一个客户端的请求需要经过 5 个阶段，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，PBFT 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。视图更换协议与 VR 的视图更换协议一样，请参考 3.2.1 节。

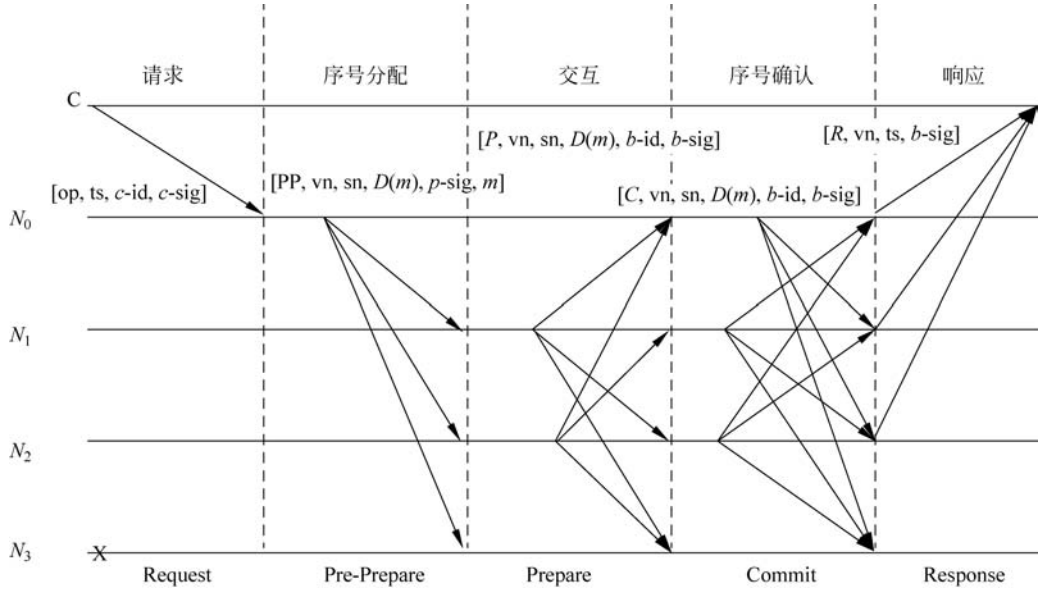

图 3.8 PBFT 协议正常模式

图 3.8 显示了一个简化的 PBFT 协议正常模式，其中 C 为客户端，$N_0 \sim N_3$ 表示服务节点，N_0 为主节点，X 表示 N_3 为故障节点。虚线是各阶段边界，箭头表示消息从消息源节点发送到接收节点。整个协议的基本过程如下。

1. Request(请求)阶段

客户端发送请求给主节点，请求消息 m=[op,ts,c-id,c-sig]，其中 op 是需要执行的操作，ts 是时间戳，c-id 是客户端 ID，c-sig 是客户端签名。时间戳是为了保证命令只被执行一次，客户端的签名是为了客户认证和权限控制。

2. Pre-Prepare (序号分配)阶段

当主节点接收请求后，主节点给请求赋值一个序列号 sn，广播序号分配消息和客户端的请求消息 m，并将构造 Pre-Prepare 消息[PP,vn,sn,$D(m)$,p-sig,m]给各从节点，其中 PP 表示 Pre-Prepare 消息，vn 是视图号，$D(m)$是客户消息的摘要(哈希值)，p-sig 是主节点的签名，m 是客户消息。序列号是为了保证命令执行的顺序，视图号让从节点记录当前视图。主节点签名是为了让从节点认证主节点身份。消息摘要保证消息没有被篡改。

3. Prepare(交互)阶段

从节点接收 Pre-Prepare 消息，向其他服务节点广播 Prepare 消息[P,vn,sn,$D(m)$,b-id,b-sig]，其中 P 表示 Prepare 消息，b-id 是从节点 ID，b-sig 是从节点签名。

4. Commit(序号确认)阶段

从节点在收到 $2f+1$ 个 Prepare 消息后，对视图内的请求和次序进行验证，广播 Commit 消息[C,vn,sn,$D(m)$,b-id,b-sig]，其中 C 表示 Commit 消息。执行收到客户端请求并给客户端以响应。

5. Response(响应)阶段

(1) 当各节点收到 $2f+1$ 个 Commit 消息后，它将执行操作并提交，同时把回复返回给客户端。回复消息是[R,vn,ts,b-sig]，其中 R 表示回复消息。

(2) 客户端等待来自不同节点的响应，若有 $f+1$ 个响应相同，则该响应即为运算的结果。

PBFT 能在异步环境下，容忍不足三分之一的总数节点是拜占庭节点，并能够达成共识。表面看起来这和 FLP 定理的不可能共识结论相违背，但其实如果检查其假设条件，可以看到 PBFT 的异步通信环境假设和 FLP 的异步通信假设不一样。PBFT 中有超时机制，而 FLP 中假设没有超时机制。在超时机制下，PBFT 能够保证 safety 和 liveness。另外，在 PBFT 中，消息的发送者是经过认证的，发送消息的身份可以被确定，节点有没有发消息也可以验证。

PBFT 在很多场景都有应用，在区块链场景中，一般适合于对强一致性有要求的私有链和联盟链场景。例如，在 IBM 主导的区块链超级账本项目中，PBFT 是一个可选的共识协议。Fabric 0.6 版本自带 PBFT 共识算法。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，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，即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，为此这个自有共识协议会在 PBFT 实现的基础之上，引入代码执行结果签名进行验证。

3.3.2 强一致性拜占庭容错共识算法小结

本节介绍了强一致性拜占庭容错共识算法中最为典型的实用拜占庭容错(PBFT)算法。在区块链的企业级应用,例如大部分联盟链或私有链,通常需要保证系统各节点的强一致性,也就是说,保证系统各节点按相同的顺序执行相同的命令,形成相同的结果。这些结果一旦形成共识,就不可改变。在特别有拜占庭故障的环境,需要有强一致的拜占庭容错共识算法,而 PBFT 是这一类共识算法的基础。其他的拜占庭容错共识算法还有 Fab Paxos、XFS、Zyzzyva、SBFT 等。

3.4 非强一致性共识算法 PoW 机制

比特币系统的重要概念是一个基于互联网的去中心化分布式账本,该分布式账本以区块链形式保存,每个区块相当于账本页,区块中记录的信息主体,即为相应的交易内容。传统账本为了保证交易内容的唯一性,一般要求记账行为是中心化的行为。然而,中心化所引发的单点失败,可能导致整个系统面临危机甚至崩溃。另外,中心化的记账要求所有交易相关方信任中心化记账机构。这种信任很多时候需要很高的成本才能维持,例如银行等金融机构需要有雄厚的资本来维持信任。

去中心记账可以克服中心化记账成本高、安全脆弱的缺点,但需要解决记账行为的一致性问题。从去中心化账本系统的角度,每个加入这个系统的节点都要保存一份完整的账本,但每个节点却不能同时记账,因为节点处于不同的环境,接收到不同的信息,如果同时记账,必然会导致账本的不一致,造成混乱。因此,需要达成由哪个节点有权记账的共识。比特币区块链通过基于算力的随机性竞争记账的方式,选出一个节点来生成一个区块的交易信息,向其他节点同步这个新增的区块信息,解决去中心化的记账系统的一致性问题。

比特币系统设计了以每个节点的计算能力(即“算力”)来竞争记账权的机制。在比特币系统中,大约每 10 分钟进行一轮算力竞赛,竞赛的胜利者,就获得一次记账的权力,并向其他节点同步新增账本信息。然而,在一个去中心化的系统中,谁有权判定竞争的结果呢? 比特币系统是通过一个称为“工作量证明(Proof of Work,PoW)”的机制完成的。简单地说,PoW 就是一份确认工作端做过一定量的工作的证明。PoW 系统的主要特征是计算的不对称性。工作端需要做一定难度的工作得出一个结果,验证方却很容易通过结果来检查出客户端是不是做了相应的工作。

举个例子,对于给定的字符串 blockchain,我们给出的工作量要求是,可以在这个字符串后面连接一个称为 Nonce 的整数值串,对连接后的字符串进行 SHA-256 哈希运算,如果得到的哈希结果(以十六进制的形式表示)是以若干个 0 开头的,则验证通过。为了达到这个工作量证明的目标,需要不停地枚举 Nonce 值,然后对得到的新字符串进行 SHA-256 哈希运算。按照这个规则,需要经过 2688 次计算才能找到前 3 位均为 0 的哈希值; 而要找到前 6 位均为 0 的哈希值,则需进行 620 969 次计算。

```
    blockchain1 →
4bfb943cba9fb9926df93f33c17d64b378d56714e8a29c6ba8bdc9690cea8e27
    blockchain2 →
01181212a283e760929f6b1628d903127c65e6fb5a9ad7fe94b790e699269221
    … …
    blockchain515 →
0074448bea8027bebd6333d3aa12fd11641e051911c5bab661a9b849b83958a7
    … …
    blockchain2688 →
0009b257eb8cf9eba179ab2be74d446fa1c59f0adfa8814260f52ae0016dd50f
    … …
    blockchain48851: 00000b3d96b4db1a976d3a69829aabef8bafa35ab5871e084211a16d3a4f385c
    … …
    blockchain6200969: 000000db7fa334aef754b51792cff6c880cd286c5f490d5cf73f658d9576d424
```

通过上面这个计算特定 SHA-256 运算结果的示例，我们对 PoW 机制有了一个初步的理解。对于特定字符串后接随机 Nonce 值所构成的串，要找到一个 Nonce 值，满足前 n 位均为 0 的 SHA-256 值，需要多次进行哈希值的计算。一般来说，n 值越大，需要完成的哈希计算量也越大。由于哈希值的伪随机特性，要寻求 4 个前导 0 的哈希散列，预期大概要进行 216 次尝试，这个数学期望的计算次数，就是所要求的“工作量”。

比特币网络中任何一个节点，如果想生成一个新的区块并写入区块链，必须解出比特币网络出的 PoW 问题。这道题关键的三个要素是工作量证明函数、区块及难度值。工作量证明函数是这道题的计算方法，区块决定了这道题的输入数据，难度值决定了这道题的所需要的计算量。

1. 工作量证明函数

比特币系统中使用的工作量证明函数正是 SHA-256。

SHA 是安全哈希算法(Secure Hash Algorithm)的缩写，是一个密码散列函数家族。这组函数是由美国国家安全局(NSA)设计，美国国家标准与技术研究院(NIST)发布的，主要适用于数字签名标准。SHA-256 就是这个函数家族中的一个，是输出值为 256 位的哈希算法。到目前为止，还没有出现对 SHA-256 算法的有效攻击。

2. 区块

比特币的区块由区块头及该区块所包含的交易列表组成。区块头的大小为 80 字节，由 4 字节的版本号、32 字节的上一个区块头的哈希值、32 字节的默克尔根哈希值、4 字节的时间戳(当前时间)、4 字节的当前难度值、4 字节的随机数组成。区块包含的交易列表，也叫区块体，附加在区块头后面，其中的第一笔交易是 coinbase 交易，这是一笔为了让矿工获得奖励及手续费的特殊交易。

区块的大致结构如图 3.9 所示。

拥有 80 字节固定长度的区块头，就是用于比特币工作量证明的输入字符串。因此，为了使区块头能体现区块所包含的所有交易，在区块的构造过程中，需要将该区块要包含的交易列表，通过默克尔树算法生成默克尔根哈希值，并以此作为交易列表的哈希值存到区块头中。其中默克尔树的算法图解如图 3.10 所示。

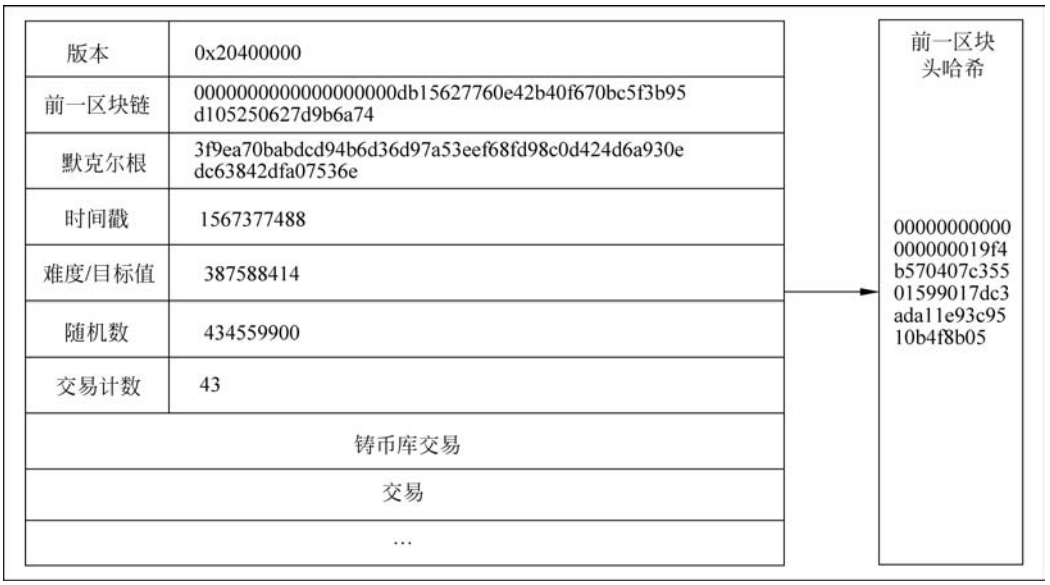

图 3.9 区块的结构

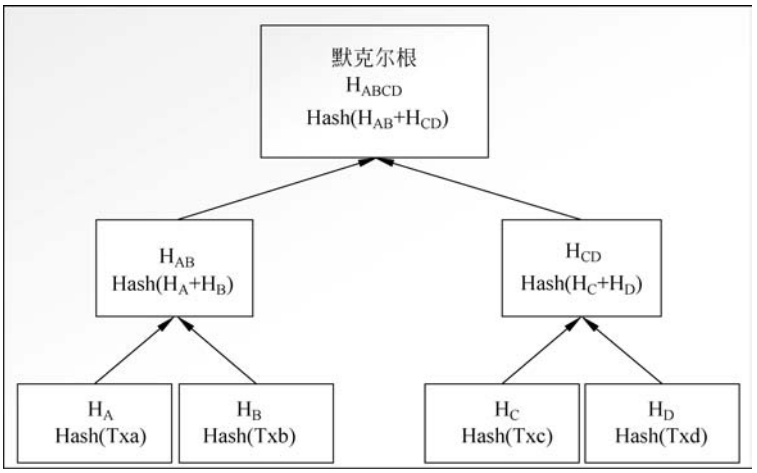

图 3.10 带 4 个交易记录的默克尔树的根哈希值的计算

图 3.10 展示了一个具有 4 个交易记录的默克尔树的根哈希值的计算过程。首先以这 4 个交易作为叶子节点构造一棵完全二叉树，然后通过哈希值的计算，将这棵二叉树转化为默克尔树。

首先对 4 个交易记录(Txa～Txd)，分别计算各自的哈希值 H_A～H_D，然后计算两个中间节点的哈希值 $H_{AB}=Hash(H_A+H_B)$和 $H_{CD}=Hash(H_C+H_D)$，最后计算出根节点的哈希值 $H_{ABCD}=Hash(H_{AB}+H_{CD})$。

而构造出来的区块链如图 3.11 所示。

由图 3.11 所示简化的区块链结构可以发现，所有在给定时间范围需要记录的交易信息被构造成一个默克尔树，区块头中包含了指向这个默克尔树的哈希指针，关联了与该区块相关的交易数据；另一方面，区块中也包含了指向前一区块的哈希指针，使得记录了不同交易

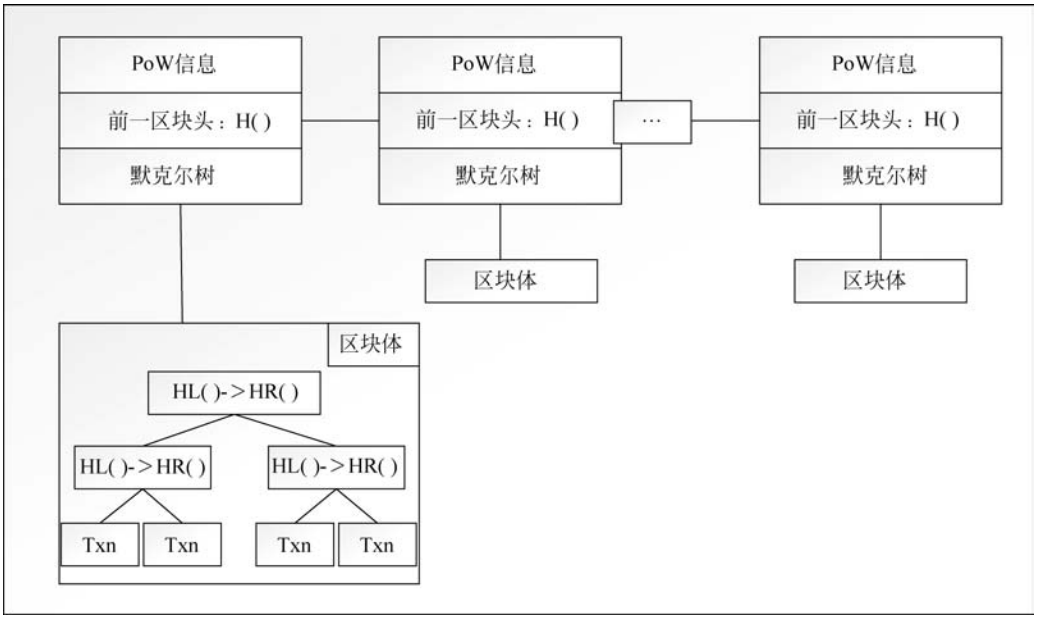

图 3.11　区块链的简化结构

的单个区块被关联起来，形成区块链。

3. 难度值

难度值是比特币系统中的节点在生成区块时的重要参考指标，它决定了节点大约需要经过多少次哈希运算才能产生一个合法的区块。比特币的区块大约每 10 分钟生成一个，如果要在不同的全网算力条件下，新区块的产生都基本保持这个速率，难度值就必须根据全网算力的变化进行调整。简单地说，难度值根据全网节点计算能力变化来设定，使得新区块产生速率能都保持在大约 10 分钟一个。

难度的调整是在每个完整节点中独立自动发生的。每 2016 个区块，所有节点都会按统一的公式自动调整难度，这个公式是由上个 2016 个区块的花费时长与期望时长（期望时长为 20 160 分钟，即两周，是按每 10 分钟一个区块的产生速率计算出的总时长）比较得出的，根据实际时长与期望时长的比值，进行相应调整（或变难或变易）。也就是说，如果区块产生的速率比 10 分钟快则增加难度，比 10 分钟慢则降低难度。

这个公式可以总结为如下形式：

新难度值＝旧难度值×（过去 2016 个区块花费时长/20 160）分钟

在比特币中，难度值是一个 32 字节的数，它将被压缩成一个 4 字节的难度位存在区块头中。目标值和难度位的换算关系如下：

目标值＝难度位后三字节$\times 2^{8\times(\text{难度位第一字节}-3)}$

目标值＝最大目标值/难度值

其中最大目标值为一个恒定值：

0x00000000FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF

目标值的大小与难度值成反比。比特币工作量证明的达成就是矿工计算出来的区块哈希值必须小于目标值。

4. PoW 的过程

比特币 PoW 的过程，可以简单理解成通过尝试不同的 Nonce 值作为输入，进行 SHA-256 哈希运算，找出满足给定数量前导 0 的哈希值的过程。而要求的前导 0 的个数越多，代表难度越大。工作量证明流程如图 3.12 所示。

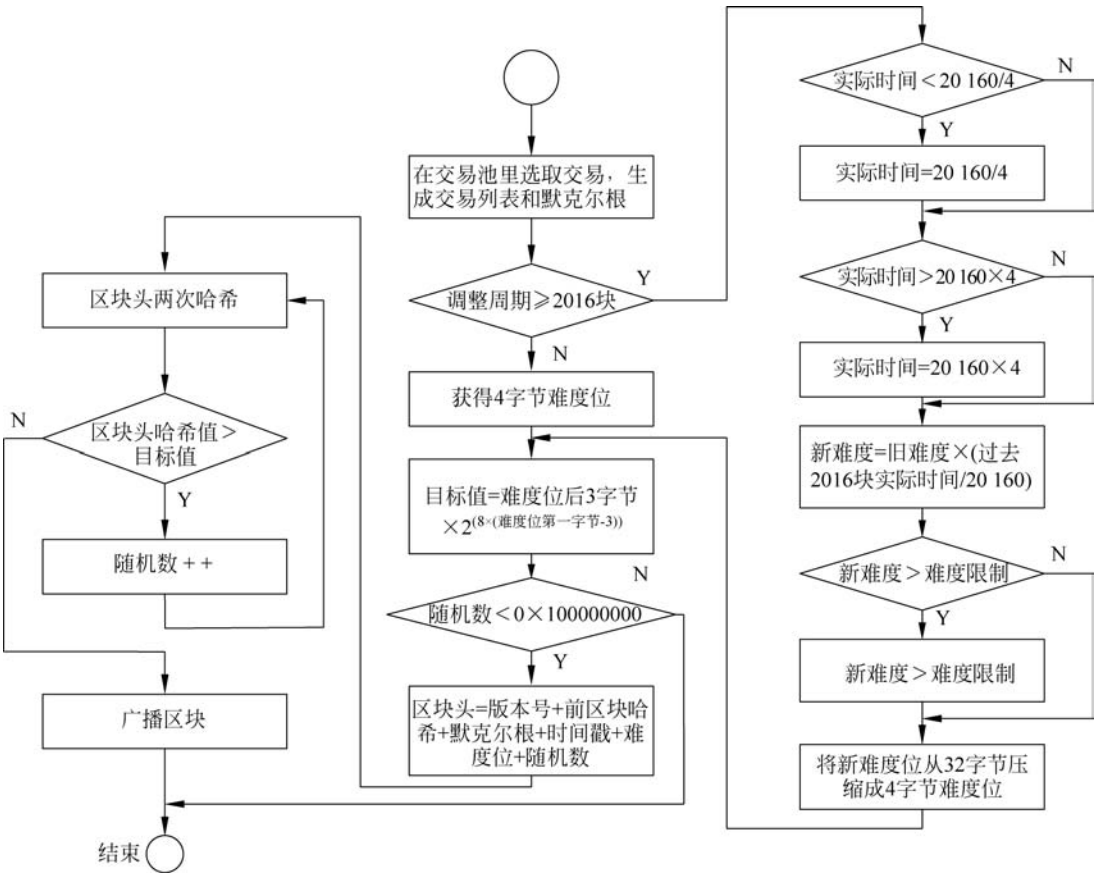

图 3.12　比特币 PoW 共识算法流程

可以把比特币节点求解工作量证明问题的步骤归纳如下：

(1) 生成铸币(Coinbase)交易，并与其他所有准备打包进区块的交易组成交易列表，通过默克尔树算法生成默克尔根哈希；

(2) 判断距上次难度调整是否到达 2016 区块，如果是，进行难度调整，获得新的难度位；

(3) 通过难度位计算目标值；

(4) 把版本号、前一区块头哈希值、默克尔根哈希值、时间戳、难度位和随机数字段组装成区块头，将区块头的 80 字节数据作为工作量证明的输入；

(5) 不停地变更区块头中的随机数，即 Nonce 的数值，并对每次变更后的区块头做双重 SHA-256 运算(即 SHA-256(SHA-256(Block_Header)))，将结果值与当前网络的目标值做比较，如果小于目标值，则解题成功，工作量证明完成。

比特币的工作量证明，就是俗称“挖矿”所做的主要工作。理解工作量证明机制，将为进一步理解比特币区块链的共识机制奠定基础。

5. 基于 PoW 的共识记账

以比特币网络的共识记账为例，来说明基于 PoW 的共识记账过程。

(1) 客户端产生新的交易，向全网进行广播，要求对交易进行记账。

(2) 每一个记账节点一旦收到这个请求，将收到的交易信息纳入一个区块中。

(3) 每个节点都通过 PoW 过程，尝试在自己的区块中找到一个具有足够难度的工作量证明。

(4) 当某个节点找到了一个工作量证明，它就向全网广播符合难度的区块。

(5) 当且仅当包含在该区块中的所有交易都是有效的且之前未存在过的，其他节点才认同该区块的有效性。

(6) 其他节点表示它们接受该区块，而表示接受的方法，则是在跟随该区块的末尾，制造新的区块以延长该链条，而将被接受区块的哈希值视为下一新区块的前项哈希值。

(7) 比特币系统默认最长的区块链为共识区块链，被最长区块链包含的区块，就是系统对区块内交易达成的共识的交易。

通过上述的记账过程，客户端所要求记录的交易信息被写入了各个记账节点的区块链中，形成了一个分布式的高概率的一致性账本。注意，比特币的工作量证明机制是最终一致性共识而不是强一致的共识算法，因为理论上存在一个概率，只要有足够的算力，形成一个更长的区块链，就可以把过去的共识结果推翻。当然，随着区块链的延长，要推翻过去共识的概率会呈指数型减少，因此可以达到最终一致性。

6. 关于比特币 PoW 能否解决拜占庭将军的问题

关于比特币 PoW 共识机制能否解决拜占庭将军问题一直在业界有争议。2015 年，Juan Garay 对比特币的 PoW 共识算法进行了正式的分析，得出的结论是比特币的 PoW 共识算法是一种概率性的拜占庭协议(Probabilistic BA)。Garay 对比特币共识协议的两个重要属性分析如下。

1) 一致性(Agreement)

在不诚实节点总算力小于 50%的情况下，同时每轮同步区块生成的概率很少的情况下，诚实的节点具有相同的区块的概率很高。用数学的严格语言应该是：当任意两个诚实节点的本地链条截取 K 个节点，两条链条剩下的最后一个区块不相同的概率随着 K 的增加呈指数型递减。

2) 正确性(Validity)

大多数的区块必须由诚实节点提供。严格地说，当不诚实算力非常小的时候，才能使大多数区块由诚实节点提供。

可以看到，当不诚实的算力小于网络总算力的 50%时，同时挖矿难度比较高，在大约 10 分钟出一个区块情况下，比特币网络达到一致性的概率会随确认区块的数目增多而呈指数型增加。但当不诚实算力具一定规模，甚至不用接近 50%的时候，比特币的共识算法并不能保证正确性，也就是，不能保证大多数的区块由诚实节点来提供。

因此，比特币的共识算法不适合于私有链和联盟链。其原因首先是它是一个最终一致

性共识算法，不是一个强一致性共识算法。企业应用需要有强一致的共识算法来保证交易的正确性，而不能依赖概率来决定。第二个原因是其共识效率低，提升共识效率又会牺牲共识协议的安全性。强一致的非拜占庭容错共识算法也不适合像比特币这样的公有链环境，因为在公有链上一定存在拜占庭节点。而像 PBFT 那样的强一致的拜占庭容错共识算法也由于扩展性有限，不能支持成千上万个记账节点。

另一方面，比特币通过巧妙的矿工奖励机制来提升网络的安全性。矿工挖矿获得比特币奖励以及记账所得的交易费用，使得矿工更希望维护网络的正常运行，而任何破坏网络的非诚信行为都会损害矿工自身的利益。因此，即使有些比特币矿池具备强大的算力，它们都没有作恶的动机，反而有动力维护比特币的正常运行，因为这和它们的切实利益相关。

3.5 PoS 机制

PoW 背后的基本概念很简单：工作端提交难以计算但易于验证的已知计算结果，而其他任何人都能够通过验证这个答案，就确信工作端为了求得结果已经完成了相当大量的计算工作。然而 PoW 机制存在明显的弊端。一方面，PoW 的前提是，节点和算力是均匀分布的，因为通过 CPU 的计算能力来进行投票，拥有钱包（节点）数和算力值应该是大致匹配的，然而随着人们将 CPU 挖矿逐渐升级到 GPU、FPGA，直至 ASIC 矿机挖矿，节点数和算力值也渐渐失配。比特币网络每秒可完成数百万亿次 SHA-256 计算，但这些计算除了使恶意攻击者不能轻易地伪装成几百万个节点打垮比特币网络，并没有更多实际或科学价值。由于 PoW 非常耗能，因此比特币的 PoW 也因为巨大的能耗浪费被很多人诟病。当然，如果从另一方面来看，相对于允许世界上任何一个人在瞬间就能通过去中心化和半匿名的全球货币网络，给其他人几乎没有手续费地转账所带来的巨大好处，它的浪费也许只算是很小的代价。

鉴于此，人们提出了一些工作量证明的替代者。权益证明（Proof of Stake，PoS）就是其中的一种方法。

3.5.1 点点币 PoS 机制

权益证明要求用户证明拥有某些数量的货币（即对货币的权益），点点币（Peercoin）是首先采用权益证明的虚拟货币，它采用一种混合共识机制，既用像比特币的 PoW 工作量证明挖矿机制，同时又用权益证明 PoS 机制。点点币中有两种块，一种是像比特币挖矿工作量证明产生的区块（区块中含有 Coinbase 交易），另一种是权益证明产生的区块（区块中含有 Coinstake 区块）。

点点币在 SHA-256 哈希运算的难度方面引入了币龄的概念，使得工作量难度与交易输入的币龄成反比。在点点币中，币龄被定义为币的数量与币所拥有的天数的乘积，这使得币龄能够反映交易时刻用户所拥有的货币数量。

实际上，点点币的权益证明机制结合了随机化与币龄的概念，未使用至少 30 天的币可以参与竞争下一区块，越久和越大的币集有更大的可能去签名下一区块。然而，一旦币的权益被用于签名一个区块，则币龄将清为零，这样必须等待至少 30 日才能签署另一区块。同时，为防止非常老或非常大的权益控制区块链，寻找下一区块的最大概率在 90 天后达到最大值，这一

过程保护了网络，并随着时间逐渐生成新的币而无须消耗大量的计算能力。点点币的开发者声称这将使得恶意攻击变得困难，因为没有中心化的挖矿池需求，而且购买半数以上的币的开销似乎超过获得51%的工作量证明的哈希计算能力。在一定程度上避免比特币的51%攻击。另外，攻击者的币龄会在攻击中受到消耗，所以对攻击者持续攻击的能力有限制。

在点点币的设计中，点点币的权益证明产生币的比率是每个币年消耗产生1分币，以保证未来的低通胀。为了避免出块率的陡然提升，工作量证明机制的目标值和权益证明机制的目标值是不停地调整的，不像比特币那样大约两周才调整一次。

和比特币中最大工作量的链被选为共识区块链不同，点点币中最大的消耗币龄的链条被选为共识区块链。

另一方面，用权益证明机制也使得改变区块链历史的攻击成本降低，同时双花攻击的成本也会降低。攻击者可以积累一定数量的币龄来强制重组区块链。为了应对这种威胁，点点币引入一个中心化Checkpoint广播机制，每天几次广播Checkpoint消息，以确认区块链中的交易。确认后的区块不能再被推翻，确认后的区块中的交易将被视为最终交易。

权益证明必须采用某种方法定义任意区块链中的下一合法区块，依据账户结余来选择将导致中心化，例如单个首富成员可能会拥有长久的优势。为此，人们还设计了其他不同的方法来选择下一合法区块。

未币(NXT)和黑币(Blackcoin)采用随机方法预测下一合法区块，使用公式查找与权益大小结合的最小哈希值，由于权益公开，每个节点都可以合理地准确预计哪个账户有权建立区块。详细的未币PoS机制见3.5.2节。

瑞迪币(Reddcoin)引入权益速度证明，即鼓励钱币的流动而非囤积。通过给币龄引入指数衰减函数，使得1个币的币龄不会超过2币月。

3.5.2 NXT PoS机制

与点点币不同，NXT币是一个完全使用PoS权益证明的虚拟货币，同时它也不采用“币龄”的概念。NXT一次性产生10亿个币，平均每分钟出一个块。每10个块后，交易就采用SHA-256和Curve25519签名，Curve25519是椭圆曲线中提供128位安全的一种签名机制。它采用椭圆曲线Diffie-Hellman(ECDH)密钥协议机制，也是最快的ECC曲线之一，同时该曲线是公开的，没有专利保护。

在NXT中，三个数值决定哪个账户有资格竞争生成区块，哪个账户获得产生区块的资格，以及在有冲突时，哪个区块链被选为合法区块。这三个数值为基础目标值、目标值和累积难度值。

(1) 基础目标值。为了获得区块产生权，所有活跃的NXT账户竞争生成一个小于给定基础目标值的哈希值。这个基础目标值在每个区块都不一样，它是由上一个区块链的基础目标值乘以需要生成一个区块的时间来获得。

(2) 目标值。每个账户基于当前有效的权益来计算它自身的目标值：

$$T = \mathrm{Tb} \times S \times \mathrm{Be}$$

其中，T是新的目标值，Tb是基础目标值，S是从上个区块生成以来的以秒为单位的时间，Be是当前有效的权益。

从公式可以看出，目标值随着上个区块时间戳后的时间流逝而增加，最大的目标值是

1.537 228 67×1017。最小的目标值是上个区块的基础目标值的一半。

所有竞争生成区块的账户的基础目标值和时间都一样，唯一一个与账户相关的参数是余额。

(3) 累积难度值。是通过以下公式从基础目标值算出：

$$Dcb = Dpb + 264/Tb$$

其中，Dcb 是当前区块的难度，Dpb 是上个区块的难度，Tb 是当前区块的基础目标值。

区块产生算法如下。

链上的每个区块有一个区块产生签名参数。为了参与区块产生流程，一个活跃的账户用它的公钥对上个区块的产生进行签名，这个生产一个 64 字节的签名，然后对这个签名进行 SHA-256 哈希，结果的头 8 字节是一个数值，叫作账户的击中值。这个击中值将与当前目标值来比较，如果击中值比目标值小，那么下一个区块就可以产生。从以上公式可以看出，目标值随着时间的推移而增加，那么即使只有几个活跃账户，其中一个必然会生产一个区块，因为目标值会越来越大。这样一来，通过比较一个账户的击中值和目标值，就能预测任何一个账户生成区块所需的大致时间。由于任意一个节点可以查询任何一个活跃账户的余额，这样就可以在比较高的精确度下预测哪一个账户会产生下一个区块。

当一个活跃账户赢得生成区块的权利，会把 255 个尚未确认的交易打包进一个区块，并将这个区块广播到网络作为候选区块。当网络有多个候选区块时，具有最高累积难度值的区块将被选为当前区块链的新区块。

3.5.3 Tendermint PoS 机制

Tendermint 区块链针对比特币挖矿机制的能耗问题以及交易确认慢的问题，提出一个不需要挖矿，从而能提升交易确认性能，而又在理论上能保证拜占庭容错的 PoS 机制。传统的 PoS 机制存在着 nothing at stake problem(无成本作恶问题)，也就是说参与共识的节点可以不用担心损失而同时在两个分叉的链上投票。Tendermint 的 PoS 机制采用押金投票机制避免了这个问题，同时也能够在有不超过 1/3 的拜占庭节点存在情况下保证共识的一致性和正确性，是第一个能在理论上证明拜占庭容错的 PoS 共识协议。

Tendermint 区块链由 Validator(验证者)投票来验证交易。验证者需要在投票期中把一定的币作为抵押物，然后通过广播带有签名的投票交易投票选出下一个区块来参与共识。验证者的投票权重相当于他们抵押的资金。验证者如果要退出共识流程，可以发一个取消抵押的交易来申请取回押金。在验证者发出取消抵押的交易后，抵押的资金还会被锁定一段设定的期限(抵押锁定期)，等该期限过后验证者才能取回押金。一个共识区块需要有超过 2/3 的投票权重的验证者签名投票同意，才能被确认为共识区块。

当一个验证者故意同时投票给两个不同区块，任何其他节点可以发一个短的证据交易，并把该验证者的两个签名投票的冲突交易作为证据附上，这个交易会被记录在区块链上，这个证据将使得作弊的验证者的抵押资金被没收。只要证据在抵押锁定期过期前提交，作弊的验证者就会被惩罚。需要注意的是，这个并不能完全避免超过 2/3 的验证者投票产生一个分叉的区块链，因为有可能作弊者已经取回押金并将资金转给其他节点。这种攻击叫长范围双花攻击。用户可以通过在押金锁定期间同步区块链来避免这种攻击。短范围双花攻击发生在攻击者的押金还在锁定期。通过设定验证者获得的酬劳可以提供经济激励，让验

证者意识到作弊损失比诚实投票大。

Tendermint 共识假设网络是部分同步的，它假设网络的延迟有一个上限，同时非拜占庭节点可以访问在一个区块共识期间保持足够准确的内部时钟，这些内部时钟不一定要和全球时间同步。

Tendermint 的共识流程分三个步骤，如图 3.13 所示。验证者首先发 Propose（提议）、Prevote（预投票）和 Precommit（预确认），加上另外两个特殊步骤 Commit（确认）和 NewHeight（新高度）。

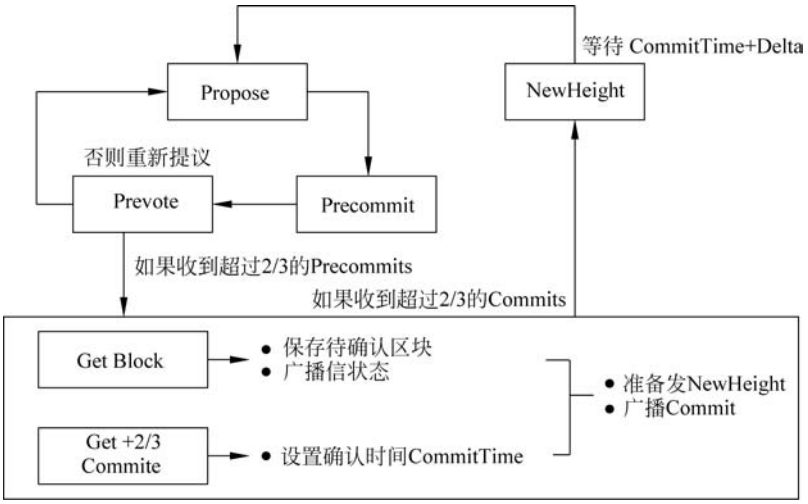

图 3.13　Tendermint 的共识流程

在每一轮共识中，验证者会采用轮询机制，按每个验证者押金的比例挑出一个验证者来做提议者，也就是押金多的验证者，被抽中做提议者的概率也相应比较大。

共识的第一阶段是提议（Propose）阶段。在提议的开始阶段，挑出来的提议者向其他邻节点广播一个提议，收到提议的节点也向它的邻节点广播该提议。

在预投票（Prevote）阶段，由每个验证者做决定，如果验证者在过去的投票阶段有一个提议，但还没被接收并处于锁住的区块，验证者可以对那个锁住的区块签名并广播一个 Prevote 交易。验证者如果接收当下这个提议者发出的提议，验证者也可以对目前的提议签名并广播一个 Prevote 交易。如果验证者没接到提议，或者这是一个不合规的提议，验证者需要签一个特殊的空 Prevote 交易并广播出去。

在 Precommit 阶段，每个验证者做一个决定，如果验证者收到对验证者提议的区块超过 2/3 的 Prevote 投票，验证者会对该区块签名并广播 Precommit 交易。验证者同时锁住该区块，并将过去锁住的区块解锁。每个验证节点每次最多能锁一个提议区块。如果一个验证者收到对验证者提议的区块超过 2/3 的空 Prevote 投票，验证者会简单地解锁。当锁或解锁提议区块，需要把 Prevote 交易的这些证据打包，放在一个 Proof-of-lock（锁证明）。如果一个验证者没有收到超过 2/3 的 Prevote 或空 Prevote，那么验证者不需要签名或锁提议区块。在 Precommit 阶段，所有的相邻节点互相通报 Precommit 交易。在 Precommit 阶段末，每个节点会做一个决定。如果节点收到对一个提议区块超过 2/3 的 Precommit 消息，那么就进入 Commit 阶段，否则的话会继续进入下一轮的提议阶段。即使一个节点还没

收到被提议的区块，都会进入 Commit 阶段。

Commit 阶段是一个特殊的阶段，有两个并行的条件需要同时满足才能最后确认区块。首先，节点必须接收到网络 Commit 的提议区块，如果接收到了，节点需要签名并广播 Commit 消息；然后，节点必须收到超过 2/3 的 Commit 消息。当这两个条件满足之后，该节点设一个 CommitTime，然后转到 NewHeight 阶段。异步通信和本地的 CommitTime 可以让网络即使在不同步的时钟情况下仍然能达成共识。

在区块链高度 H 前的 NewHeight 步骤会继续收集在高度 H-1 的提议区块的 Commit 消息，直到 CommitTime 加上一个固定的时间之后，这个允许提议的区块包括超过最小 2/3 的 Commit 消息，这样可以把比较慢的验证者的 Commit 记录在区块链上。在共识流程的任何阶段，如果一个节点收到对一个提议区块超过 2/3 的 Commit 消息，它会立即进入 Commit 阶段，所以有两种进入 Commit 步骤的途径。一个在 R 轮对提议区块的确认(Commit)投票，可以算成是对所有 R_0 轮($R < R_0$)的 Prevote 和 Precommits。确认(Commit)投票消息会在节点之间广播。

在共识流程中的任何时间，如果一个节点 R 轮在一个提议区块上锁住，但接收到一个 R_0 轮($R < R_0$)的锁证明(proof-of-lock)，这个节点将解锁。

1. Proof of Safety 安全性证明

Safety(安全性)包含一致性和正确性，也就是要证明其共识机制不会出现分叉。Tendermint 白皮书提供一个非正式的证明：

如果有低于 1/3 的拜占庭投票权重的验证者和至少一个诚实的验证者投票决定 B 区块，那么所有的诚实验证者都会投票决定 B 区块。考虑在最早的 R 轮，至少有一个诚实的验证者在 R 轮确认 B 区块，这个验证者在 R 轮会接收到超过 2/3 的 Precommit 消息。假设少于 1/3 的是拜占庭节点，那么至少有 1/3 的诚实验证者在 R 轮投了 B 区块 Precommit。这些诚实的验证者一定会在 R 轮对 B 区块加锁。其他的区块不可能被确认，除非一些诚实的验证者解锁 B 区块。因为一个验证者只能一次锁住一个区块。也就是说，不可能出现同时确认 B 区块和另一个区块的情况，因此不可能分叉。

2. Proof of Liveness 活性证明

Liveness(活性)指的是在有限的时间内能达成共识，而不出现死锁或永远达不到一致的情况。Tendermint 给出的非正式证明如下。

如果只有少于 1/3 的拜占庭投票权重，那么协议就不会死锁。只有两个不同的区块被一些不同的诚实验证者在不同的共识轮被锁住，共识流程才能进入死锁的情况。假设某些诚实验证者在 R 轮共识锁住了区块 B，一些别的诚实验证者在 R_0 轮锁住了区块 B_0，这里 $R < R_0$。在这种情况下，在 R_0 轮的一个诚实验证者的一个提议所包含的证明锁(proof-of-lock)总会有一个时间解锁，让那些在 R 轮锁住 B 区块的验证者能继续共识流程。

3.5.4 Ethereum Casper PoS 机制

以太坊在开始的版本规划的时候，就计划在未来的 Serenity 版本将共识机制从 PoW 切换到 Casper 的 PoS 共识机制，但后面制定 Casper 方案的工作一拖再拖，直到现在，并没有正式发布。下面从以太坊的一些论坛信息，和根据以太坊创始人 Vitalik Buterin 和 Virgil

Griffith 的网上白皮书 *Casper the Friendly Finality Gadget* 的资料总结出 Casper 共识机制的一些要点。

出于安全方面的考虑，Casper 的第一版本不会全部替换 PoW 共识算法，而是 PoW 和 PoS 的混合共识算法。具体说就是以太坊会继续使用 PoW 来验证绝大多数区块，PoS 将用来在每 100 个区块做 checkpoint，也就是永久确认。这个操作给以太坊区块链带来交易确认的最终性。

在 Casper 第一阶段，会在一个 CASPER_ADDR 地址上发布一个 Casper 合约。这个合约将让所有人打进以太币，并写上一个可用来签名的“验证代码”（可以想象成类似一个公钥），从而成为一个验证者。当一个用户进入活跃的验证池之后，就可以发消息来参与 PoS 的共识流程。一个验证者的大小指的是他在合约上存放多少以太币。PoS 共识的目的是为了“最终确认”叫 checkpoint 的关键区块。每第 100 个区块就是一个 checkpoint。为了使得一个区块被最终确认，在活跃验证池的至少占总验证池 2/3 的一个子集验证者（其实是相当于超过 2/3 的存储金所代表的投票）需要针对这个区块发 Commit 消息。当一个区块被最终确认后，那就意味着永远不能推翻。即使 99％的矿工开始支持一个不包含该区块的链，客户都仍然会认为该区块所包含的交易是最终确认的。

验证者广播用自己私钥签名投票消息，投票消息中包含 4 个信息，其中两个是 checkpoint s 和 t。s 是源 checkpoint，指的是被确认过的 checkpoint，t 是目标确认 checkpoint 区块。另外两个是 s 和 t 所对应的高度 $h(s)$ 和 $h(t)$。这里的高度指的是从创世区块开始顺着链下来 100 个区块的倍数。因为 Casper 只是用来验证并最终确认每 100 个区块（checkpoint），因此这里的高度是指距离创世区块 100 的倍数。

Casper 合约中的规则叫“苛刻条件（Slashing Conditions）”，这些规则用来防止不兼容的两个区块同时被确认，同时防止验证者同时投票确认两个区块。假设验证者广播两个投票消息：$<s_1,t_1,h(s_1),h(t_1)>$和$<s_2,t_2,h(s_2),h(t_2)>$，主要有两个苛刻条件：

第一个是 $h(t_1) \neq h(t_2)$，也就是说，一个验证者不能发布两个不同的投票信息，其中的两个 checkpoint 的高度一样。第二个是验证者不能发布一个嵌套于另一个 checkpoint 的 checkpoint，也就是说，不能发布两个满足下面条件的验证消息，使得 $h(s_1)<h(s_2)<h(t_2)<h(t_1)$。违反苛刻条件的验证者会被没收押金。

这两个苛刻条件是为了避免区块链的分叉。如果区块 A 和 B 都是 C 的子块，那么如果 A 和 B 都得到确认，那就说明至少存在着有 1/3 的活跃验证池的验证者，同时发了消息确认 A 块和 B 块。这些发双重验证消息的验证者的消息证据会被发到 Casper 合约，他们存的以太币押金的 96％会被销毁，而他们存的以太币的 4％会被奖励给提供证据的举报者。所以一旦推翻一个被确认的区块会带来非常昂贵的损失。

Casper 第一阶段主要是实施两部分工作，第一部分是“分叉选择规则（fork choice rule）”，该功能是把过去 PoW 中选用的“最长链”规则改成选用带最大高度的最终确认区块（checkpoint）的规则。第二部分是实现 Casper 验证逻辑的一个后台进程。

3.5.5 LPoS 机制

在传统的 PoS 机制，小份额的权益人只有非常小的机会获得出块记账机会。就像在比特币的 PoW 中算力小的矿工非常难挖到矿一样，小的权益人可能需要很多年才能有足够

的运气获得出块记账权。这就意味着很多小权益人没有动力运行节点，网络只由小部分的大玩家维持。这样的话，网络的安全性就比较差，因为一个安全的区块链网络需要很多参与者，所以需要激励小额权益者参与出块记账。LPoS(Leased Proof of Stake，租借权益证明)可以解决这个问题。它让权益人将自己的余额租借给别的节点来出块，租借的资金还是由权益人控制，同时在租借到期后可以花掉或者转走，出块后的收益由租借人和出块人共同分享。Waves 是采用 LPoS 的区块链项目。

3.5.6 DPoS 机制

PoW 机制和 PoS 机制虽然都能有效地解决记账行为的一致性共识问题，但是现有的比特币 PoW 机制纯粹依赖算力，导致专业从事挖矿的矿工群体似乎已和比特币社区完全分隔，某些矿池的巨大算力俨然成为另一个中心，这与比特币的去中心化思想相冲突。PoS 机制虽然考虑了 PoW 的不足，但依据权益结余来选择，会导致首富账户的权力更大，有可能支配记账权。DPoS(Delegated Proof of Stake，股份授权证明机制)的出现正是为了解决 PoW 机制和 PoS 机制的这类不足。

比特股(Bitshare)是一类采用 DPoS 机制的密码货币，它期望通过引入一个技术民主层来减少中心化的负面影响。

DPoS 是目前看到最快、最高效、也最去中心化和最灵活的共识模型。DPoS 利用权益人投票的权利来公平民主地解决共识问题。所有的网络参数，从交易费用、生成块的时间以及交易大小，都可以通过选出来的代理人来调整。确定性地选择出块人可以平均一秒的速率确认交易。比特股引入了见证人这个概念，见证人可以生成区块，每一个持有比特股的人都可以投票选举见证人。得到总同意票数中的前 N 个(N 在 Bitshares 中定义为 101，在 EOS 中定义为 21)候选者可以当选为见证人，当选见证人的个数(N)有以下条件确认：至少一半的参与投票者相信 N 已经充分地去中心化。

见证人的候选名单每个维护周期(1 天)更新一次。见证人随机排列，每个见证人按序有两秒的权限时间生成区块，若见证人在给定的时间片不能生成区块，区块生成权限交给下一个时间片对应的见证人。DPoS 的这种设计使得区块的生成更为快速，也更加节能。

DPoS 充分利用了持股人的投票，以公平民主的方式达成共识，他们投票选出的 N 个见证人，可以视为 N 个矿池，而这 N 个矿池彼此的权利是完全相等的。持股人可以随时通过投票更换这些见证人(矿池)，如果他们提供的算力不稳定、计算机宕机或者试图利用手中的权力作恶。

比特股还设计了另外一类竞选——代表竞选。选出的代表拥有提出改变网络参数的特权，包括交易费用、区块大小、见证人费用和区块区间。若大多数代表同意所提出的改变，持股人有两周的审查期，这个期间可以罢免代表并废止所提出的改变。这一设计确保代表技术上没有直接修改参数的权利以及所有的网络参数的改变最终需得到持股人的同意。

2017 年新出的 TeZoS 和 EOS 项目采用的是 DPoS 机制。

3.5.7 PoS 共识算法小结

PoS 共识基本上可以分两种类型：一种是用权益模仿 PoW 机制，通过伪随机数来模仿挖矿决定产生块的权利。这方面的例子像点点币(Peercoin)、黑币(Blackcoin)。还有在此基础上构造的 DPoS，它是一个更高效、更公平民主的 PoS 机制。另一种类型是把拜占庭容错(BFT)和 PoS 结合起来，可以有严格的数学证明保证只要满足超过 2/3 的诚实参与节点，共识协议不会确认不一致的区块。Tendermint 的共识算法是第一个实现 BFT 的 PoS 共识算法。以太坊未来的 Casper 的 PoS 机制也具有 BFT 容错能力。

3.6 Ripple 共识算法

1. Ripple 的网络结构

Ripple(瑞波)是一种基于互联网的开源支付协议，可以实现部分去中心化的货币兑换、支付与清算功能。在 Ripple 的网络中，交易由客户端(应用)发起，经过追踪节点(Tracking Node)或验证节点(Validating Node)把交易广播到整个网络中。追踪节点主要功能是分发交易信息以及响应客户端的账本请求。验证节点除包含追踪节点的所有功能外，还能够通过共识协议，在账本中增加新的账本实例数据。图 3.14 是 Ripple 共识过程中节点的交互示意图。

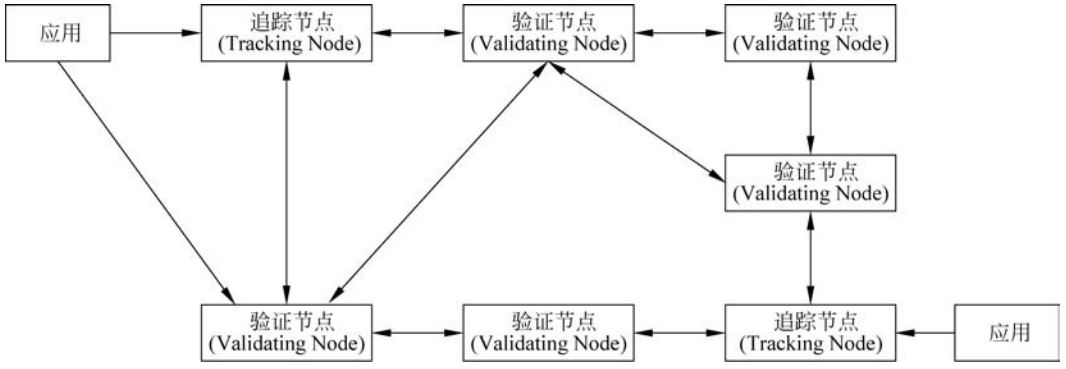

图 3.14 Ripple 共识过程中节点的交互示意图

2. Ripple 共识算法

Ripple 的共识达成发生在验证节点之间，每个验证节点都预先配置了一份可信任节点名单，称为 UNL(Unique Node List)。在名单上的节点可对交易达成进行投票。每隔几秒 Ripple 网络将进行如下共识过程。

(1) 每个验证节点会不断收到从网络发送过来的交易，通过与本地账本数据验证后，不合法的交易直接丢弃，合法的交易将汇总成交易候选集(Candidate Set)。交易候选集里面还包括之前共识过程无法确认而遗留下来的交易。

(2) 每个验证节点把自己的交易候选集作为提案发送给其他验证节点。

(3) 验证节点在收到其他节点发来的提案后，如果不是来自 UNL 上的节点，则忽略该

提案。如果是来自 UNL 上的节点，就会对比提案中的交易和本地的交易候选集，如果有相同的交易，该交易就获得一票。在一定时间内，当交易获得超过 50%的票数时，则该交易进入下一轮。没有超过 50%的交易，将留待下一次共识过程去确认。

(4) 验证节点把超过 50%票数的交易作为提案发给其他节点，同时提高所需票数的阈值到 60%，重复步骤(3)和步骤(4)，直到阈值达到 80%。

(5) 验证节点把经过 80%UNL 节点确认的交易正式写入本地的账本数据中，称为最后关闭账本(Last Closed Ledger)，即账本最后(最新)的状态。

图 3.15 是 Ripple 共识算法流程。

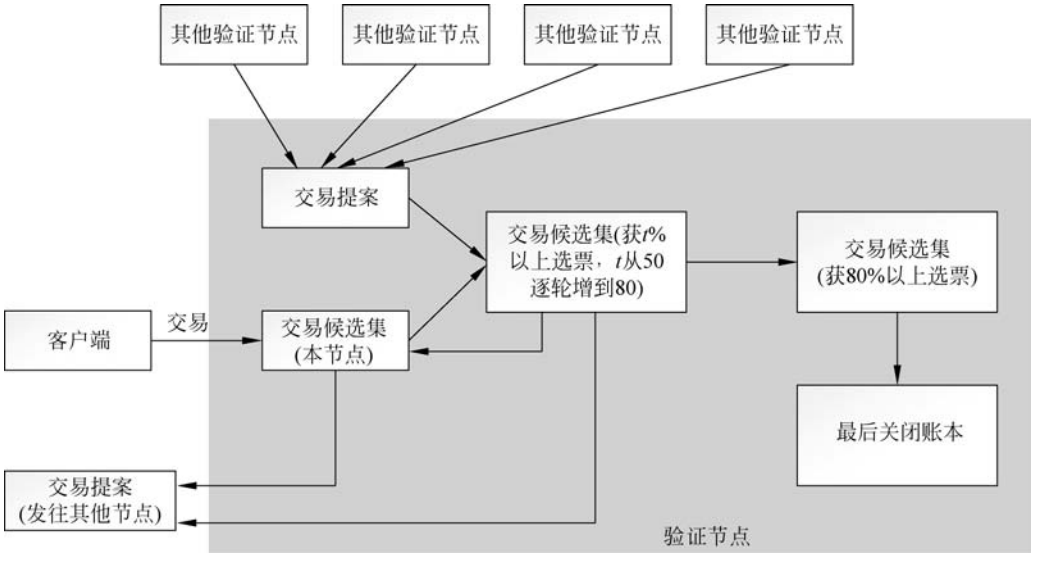

图 3.15 Ripple 共识算法流程

Ripple 共识算法的独特之处在于它不是在全网一次性达成共识，而是在 UNL 子网中达成共识。在 Ripple 的共识算法中 UNL 参与投票节点的身份是事先知道的，因此，算法的效率比 PoW 等匿名共识算法要高效，交易的确认时间只需要几秒钟。当然，这点也决定了该共识算法只适合于许可链(Permissioned Chain)的场景。同时 Ripple 共识算法假设拜占庭节点数少于所有节点数的 20%，需要任意两个 UNL 间重合的节点数至少占最大 UNL 节点数的 20%。另外，UNL 中的任意一个节点串通作恶的概率需要小于 20%。

Ripple 共识算法的拜占庭容错(BFT)能力为$(n-1)/5$，即可以容忍整个网络中 20%的节点出现拜占庭错误而不影响正确共识。

3.7 本章小结

本章主要讨论了共识机制。如何在分布式系统中高效地达成共识是分布式计算领域的重要研究问题，经典的拜占庭容错技术能够在拜占庭服务器不超过 1/3 以及同步通信的情况下，达成拜占庭系统中的共识。而在异步通信情况下，理论上只要有一个拜占庭故障服务器，就无法在全网中达成一致的共识。为了解决实际的分布式一致性问题，很多实用的共识

算法被设计出来。这些算法有不同的假设条件，具有不同的优点和局限。本章重点介绍了适用于私有链和联盟链环境的实用拜占庭容错（PBFT）协议，以及针对非拜占庭故障的Raft共识算法。Raft技术则只需要故障服务器不超过50%，就能实现交易记账的共识。

一般来说，PoW和PoS机制比较适合公有链环境，而PBFT和Raft则比较适合联盟链和私有链的分布式环境。比特币的PoW机制是一种概率性的拜占庭协议，能在一定程度上解决拜占庭问题。而PoS等其他机制，目前并没有严格的数学证明其具有拜占庭容错的属性。

参考文献

[1] Roger Wattenhofer，Strong Consistency. ETH Zurich—Distributed Computing[OL]. www.disco.ethz.ch.

[2] Lamport L，Shostak R，Pease M. The Byzantine generals problem[J]. ACM Trans. on Programming Languages and Systems，1982，4(3)：382-401.

[3] Fischer M J，Lynch N A. Impossibility of distributed consensus with one faulty process[J]. ACM 32 (2)，374-382 (1985).

[4] Seth Gilbert，Nancy A. Lynch. Perspectives on the CAP Theorem[OL]. Available on：https://core.ac.uk/download/pdf/16520017.pdf.

[5] Barbara Liskov and James Cowling. Viewstamped Replication Revisited[R]. MIT Computer Science and Artificial Intelligence Laboratory.

[6] Lamport L. The part-time parliament[J]. ACM Trans. Comput. Syst.，16(2)：133-169，1998.

[7] Ongaro D，Ousterhout J . In Search of an Understandable Consensus Algorithm[C]. In：Proc. of USENIX Annual Technical Conference 2014，305-319.

[8] Castro M，Liskov B. Practical Byzantine fault tolerance and proactive recovery[J]. ACM Trans. on Computer Systems，2002，20(4)：398-461.

[9] Satoshi N. Bitcoin：A peer-to-peer electronic cash system[OL]. Available on：http://bitcoin.org/bitcoin.pdf，2012.

[10] Garay J A，Kiayias A，Leonardos N. The Bitcoin Backbone Protocol：Analysis and Applications[J]. IACR Cryptology ePrint Archive 2014，765 (2014).

[11] Sunny King，Scott Nadal，PPCoin. Peer to Peer Cryptocurrency with Proof of Stake.[OL]. Available on：https://www.chainwhy.com/upload/default/20180619/126a057fef926dc286accb372da46955.pdf，2012.

[12] Nxt community，Nxt Whitepaper[OL]，Available on：https://www.jelurida.com/sites/default/files/Nxt whitepaper.pdf.

[13] Jae Kwon，Tendermint：Consensus without Mining[OL]. github.com/tendermint/tendermint/wiki，2014.

[14] Vitalik Buterin and Virgil Griffith：Casper the Friendly Finality Gadget[OL].

[15] Fabian Schuh，Daniel Larimer，Bitshares 2.0：General Overview[R]. Cryptonomex，Cryptonomex.com_Blacksburg (VA)，USA.

[16] David Schwartz，Noah Youngs，Arthur Britto. The Ripple Protocol Consensus Algorithm[R]. Ripple Labs Inc，2014.