第5章

区块链共识层

区块链在某种程度上是一个分布式系统，每个节点都有一份完整的账本，具有天然抵抗 DDoS 攻击的特性，并能够解决单点故障的问题。然而，区块链面临的一大难题就是账本的更新。账本的更新需要保障分布式系统的一致性，即区块链中各个节点的账本在更新后能够保持一致。区块链的共识层就是解决上述问题的层级：在去中心化且存在恶意节点的场景下维护区块链的全局账本。

5.1 一致性问题

一致性问题是分布式领域最为基础和最为重要的问题。如果分布式系统能够实现“一致”，对外便可以呈现为一个独立的、完美的、可扩展的“虚拟节点”，同时相对于物理节点具备更优越的性能以及稳定性。这也是分布式系统希望能实现的最终目标。

为什么一致性问题这么重要？下面举个例子，Rivest、Shamir 和 Adleman 三人是在一条供应链上，此时 Rivest 的账户上有 100 万美元，Shamir 账户上有 50 万美元，Adleman 账户上有 20 万美元。供应链上的三家开始货款结算，Rivest 需要给 Shamir 转账 100 万美元，Shamir 需要给 Adleman 转账 100 万美元。所以，此时存在两条交易：TX_RS：Rivest->Shamir100w；TX_SA：Shamir->Adleman100w。我们把这两笔交易发到系统中，假设系统上有节点 A 和节点 B。假设 A 先收到 TX_RS，后收到 TX_SA，A 的账本更新为{Rivest：0，Shamir：50，Adleman：120}；而 B 先收到 TX_SA，后收到 TX_RS，B 的账本更新为{Rivest：0，Shamir：150，Adleman：20}。此时，系统存在两个不一致的账本，这样的系统是很有问题的，所以一致性问题对于分布式系统来说是很必要的。

系统间一致性达成的程度也是不一样的，拜占庭容错算法达成的共识是确定性的，即共识生成高度为 n 的区块就是确定的。然而，工作量证明达成的共识是概率确定的，即共识生成高度为 n 的区块是不确定的（分叉），但随着区块链高度的增长，区块发生变化的概率越来越小，比特币的六区块确认说的就是这种现象。

5.1.1 问题与挑战

看似强大的计算机系统，实际上很多地方都比人类世界要脆弱得多。特别是在由多个计算机（节点）组成的分布式系统中，如下几方面很容易出现问题，从而极大地降低整个分布式系统的可用性。

（1）节点之间的网络通信是不可靠的。例如，消息延迟、乱序、出错，甚至出现消息丢失。

（2）节点的处理时间无法保障，处理结果可能错误，甚至节点自身可以出现系统中断。

（3）节点可以是恶意的，并通过各种手段破坏系统的一致性。

（4）同步调用可以简化设计，但会严重降低分布式系统的可扩展性，甚至使其退化为单点系统，带来单点故障的问题。仍然以上述的 Rivest、Shamir 和 Adleman 之间的转账为例，愿意思考的读者可能已经想到一些不错的解决方案。例如：

① 当收到一笔交易时，节点会先询问其他节点是否已经收到同样的交易以及交易的执行顺序以确保交易的执行结果与其他节点不冲突，即通过同步调用的方法来避免冲突。

② 所有节点提前约好某段时间交易执行顺序的决定权。比如一天之内 0～12 点发起的交易由节点 A 决定交易的执行顺序，而 12 点后到 24 点发起的交易由节点 B 负责，即通过令牌机制让节点 A 和节点 B 轮流决定交易的顺序，以避免冲突。

③ 成立一个第三方的机构，专门负责处理交易的执行顺序。节点 A 和节点 B 从该机构获得排序好的交易序列并执行，更新本地的账本。此时问题退化为中心化单点系统。

当然，还有很多的方案，这里将不再逐一列举。实际上，这些方案背后的思想，都可能引发不一致性的并行操作串行化。这也是现代分布式系统处理一致性问题的基础思路。只是因为现在的计算机系统应对故障还不够“智能”，例如上述的方案都没有考虑请求和答复消息出现失败的情况，因此实际可行的方案还需要更加全面和高效，才能保证系统更快、更稳定地工作。

5.1.2 一致性要求

规范地说，分布式系统达成一致性的过程，应该满足可终止性（Termination）、约同性（Agreement）和合法性（Validity）。

（1）可终止性。一致性的结果在有限时间内能完成。

（2）约同性。不同节点最终完成决策的结果是相同的。

（3）合法性。决策的结果必须是某个节点提出的提案。

可终止性、约同性和合法性分别对应着以下 3 个分布式系统的要求。

（1）活性（Liveness）。可终止性本质上是为了保证活性，即保证系统的可用性。如果达成一致性的过程无限长，意味着服务中断，这样系统将不能被正常地使用。活性的一种通俗的表述就是好事总会发生。注意：在现实生活中这点并不是总能得到保障的。例如，取款机有时会出现服务中断。

（2）安全性（Safety）。约同性其实是为了保证安全性，即算法要么不给出结果，任何

给出的结果必定是达成共识的。这个性质是区块链共识算法所重点关注的。在区块链系统中，一般归约成给区块内的交易定一个全局的序号。如上面的例子，A 和 B 要么同时认可收到 TX_SA 后收到 TX_RS，要么同时认可收到 TX_RS 后收到 TX_SA。安全性的一种通俗的表述就是坏事不会发生。

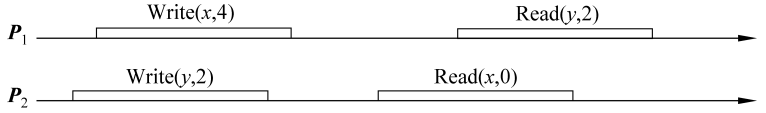
(3) 正确性(Correctness)。合法性其实是为了保证正确性。合法性相对于其他两个要求似乎不是必要的，但是如果没有这个约束，那么可以设计一个这样的共识：无论发生何种交易，都给所有的银行账户的余额增加 1。这样的共识具有强活性、强安全性，以及简单的特点，但是并不是“正确”的。所以可以看出没有合法性的约束，不能保证正确性的共识机制可以变得多荒谬。

5.1.3　不同的一致性要求

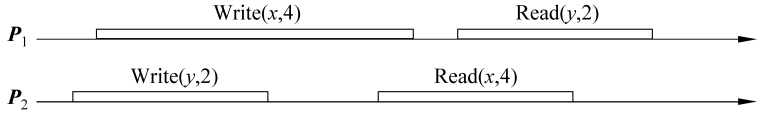
要实现绝对理想的严格一致性(Strict Consistency)代价很大。除非系统不发生任何故障，而且所有节点之间的通信无需任何时间，这时整个系统其实就等价于一台计算机了。实际上，越强的一致性要求往往会造成越弱的处理性能，以及越差的可扩展性。不同的一致性要求大概可以分成三类：线性一致性(强一致性)、顺序一致性和弱一致性。

线性一致性(Linearizability Consistency)的要求：①任何一次读都能读到某个数据的最近一次写的数据；②系统中的所有进程看到的操作顺序，都和全局时钟下的顺序一致。

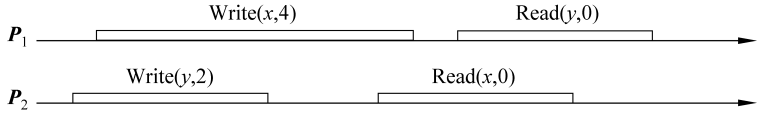
顺序一致性(Sequential Consistency)的要求：①任何一次读都能读到某个数据的最近一次写的数据；②系统中的所有进程看到的操作顺序一致，而且是合理的，即不需要和全局时钟下的顺序一致。下面以图 5.1 为例对顺序一致性进行说明(图 5.1 中的 P_1 和 P_2 分别代表系统中的进程 1 和进程 2，Write 和 Read 代表进程的读写操作)。



(a) 满足顺序一致性但不满足线性一致性



(b) 满足顺序一致性和线性一致性



(c) 不满足顺序一致性(因此不满足线性一致性)

图 5.1　线性一致性与顺序一致性

图 5.1(a)不满足线性一致性但是满足顺序一致性。首先，因为在全局时钟下，事情发生的顺序应该是{P_2：Write(y,2)，P_1：Write(x,4)，P_2：Read(x,0)，P_1：Read(y,2)}，

因为 P_1：Write(x,4)发生在 P_2：Read(x,0)前，所以不满足线性一致性，即 P_2：Read(x,0)应为 P_2：Read(x,4)。但是，图 5.1(a)满足顺序一致性，在每个进程看来，情况是这样的{Write(y,2),Read(x,0),Write(x,4),Read(y,2)}，顺序一致性不需要保证有上帝视角看到全局时钟，只需要有一个合理的全局顺序就行。图 5.1(b)能够满足线性一致性，因为不通过上帝视角得出来的结果为{Write(y,2),Write(x,4),Read(x,4),Read(y,2)}的确和全局时钟下事情发生的顺序一致。图 5.1(c)连顺序一致性都不能满足，因为经过推导，也不能得到一个自洽的全局的序。

实现线性一致性和顺序一致性往往需要准确的计时设备。Google 曾在其分布式数据库 Spanner 中采用基于原子时钟和 GPS 的 TrueTime 方案，这种方案能够将不同数据中心的时间偏差控制在 10ms 以内。方案简单粗暴而且有效，但存在成本较高的问题。

由于强一致性的系统的实现难度往往比较大，而且很多时候，现实生活中的实际需求并不是严格的强一致性。因此，可以适当地放宽对一致性的要求，从而降低系统实现的难度和复杂性。例如，在一定约束条件下实现最终一致性(Eventual Consistency)，即总会存在某一个时刻(而不是立刻)，让系统达到一致的状态。例如，平时浏览的大部分 Web 系统实现的都是最终一致性，而不是强一致性。相对强一致性，这一类在某些方面弱化的一致性都统称为弱一致性(Weak Consistency)。

最终一致性不保证在任意时刻的任意节点上每一份数据都是相同的。例如，在图 5.2 中虚线范围内，节点 A、B、C 并不能达成一致。但是随着时间的迁移，不同节点上的同一份数据总是在向趋同的方向变化。简单地说，就是在一段时间后，节点间的数据会最终达到一致状态。

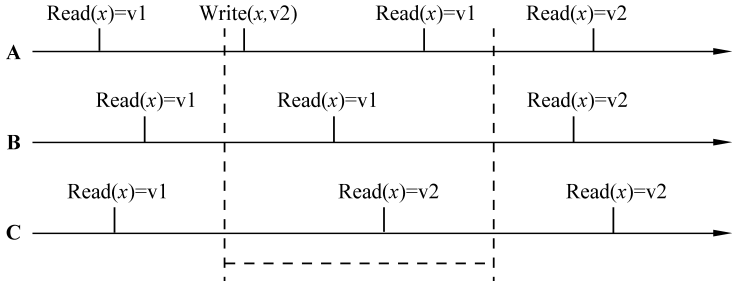


图 5.2 最终一致性并不会在所有时刻达到一致

为了把最终一致性中间的不一致的情况进行规范和分类，最终一致性根据更新数据后各进程访问数据的时间和方式的不同，又可以区分为因果一致性(Casual Consistency)、读你所写(Read-Your-Writes)一致性、会话一致性(Session Consistency)、单调读一致性(Monotonic Read Consistency)和单调写一致性(Monotonic Write Consistency)。

因果一致性适用于进程之间有因果依赖的情况。如图 5.3 所示，进程 A、B 之间存在依赖关系，进程 A、C 之间不存在依赖关系。进程 A 将 x 的值更新为 v2。由于进程 A、B 之间的依赖关系，进程 A 会通过消息 Notify(A,B,x,v2)来通知进程 B。在接到通知后，进程 B 意识到进程 A 把 x 的值设置为 v2。因此，进程 B 后续的操作会对 x 的新值 v2 进行操作，从而进程 A 和 B 保证了数据的因果一致性。另一方面，进程 C 在不一致窗口内

可能看到的依旧是 x 的旧值 v1。

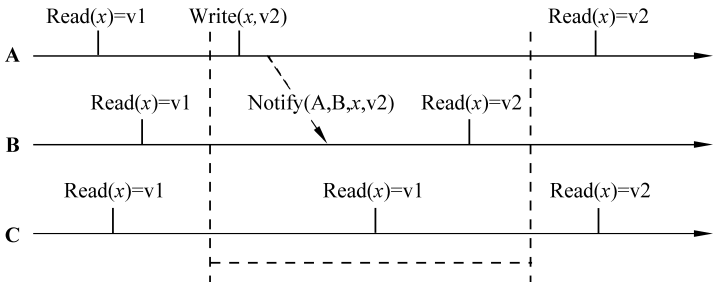


图 5.3　因果一致性

读你所写一致性属于因果一致性范畴中的特例，即进程 A 依赖于进程 A 本身。如图 5.4 所示，进程 A 把数据 x 更新为 v2 后，相当于给自身发出一条通知 Notify(A,A,x,v2)(并无发生)。同理可知，进程 A 后续依据 x 的新数值 v2 进行操作。图 5.4 中的其他进程 B、C 并未受到影响。

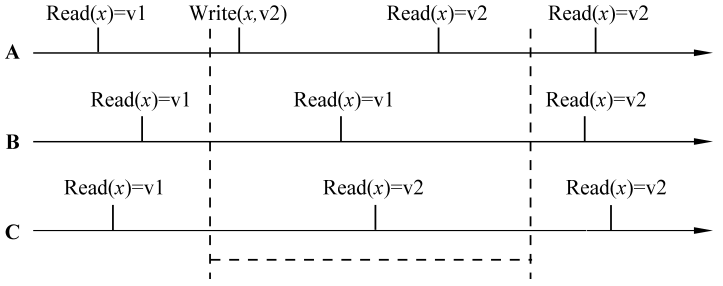


图 5.4　读你所写一致性

会话一致性是读你所写一致性的一种变体。会话一致性，是指读你所写一致性是建立在某个会话中的。对于同一个进程 A，当会话终止了，读你所写一致性也就不需要满足了，即可能读出旧值。如图 5.5 所示，在不一致窗口内，进程 A 在同一个会话内，读出的是 x 的新值 v2。若会话终止，进程 A 仍可能读出 x 的旧值 v1。

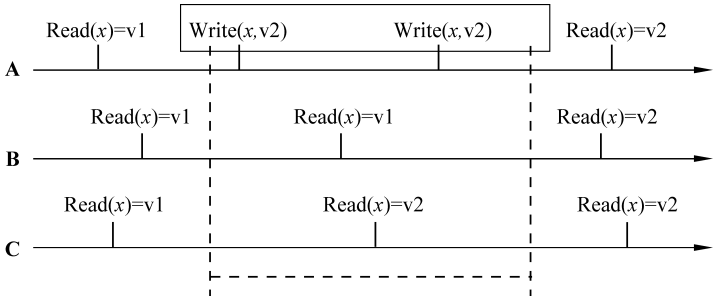


图 5.5　会话一致性

单调读一致性保证的是如果分布式系统中的某个进程(在图 5.6 中为进程 C)读取到

数据 x 的某个版本的值 v2，那么系统所有进程后序不能读出数据 x 的比值 v2 更旧的版本。图 5.6 中进程 A 在进程 C 读出 v2 后，进程 A 也读出 v2。

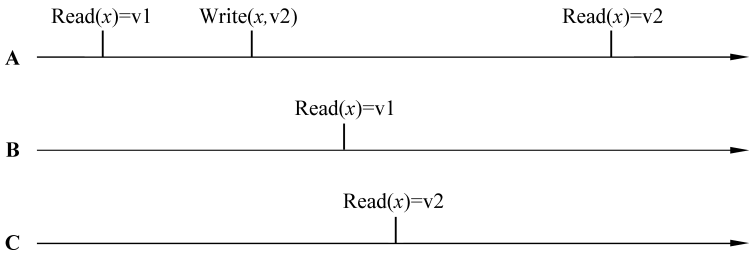


图 5.6　单调读一致性

单调写一致性保证的是同一个进程执行写操作的单调性，即一个进程对数据项 x 的写操作必须在该进程对 x 执行任何后续写操作之前完成。单调写一致性保证客户端的写操作是串行的，方便了程序的编写。

上述不同的一致性可以根据不同的场景要求组合起来使用，例如，单调读和会话一致性可以组合在一起。此外，不同一致性的要求是不同的，根据其要求的规范以及严格程度，可以归纳出不同一致性的关系，如图 5.7 所示。

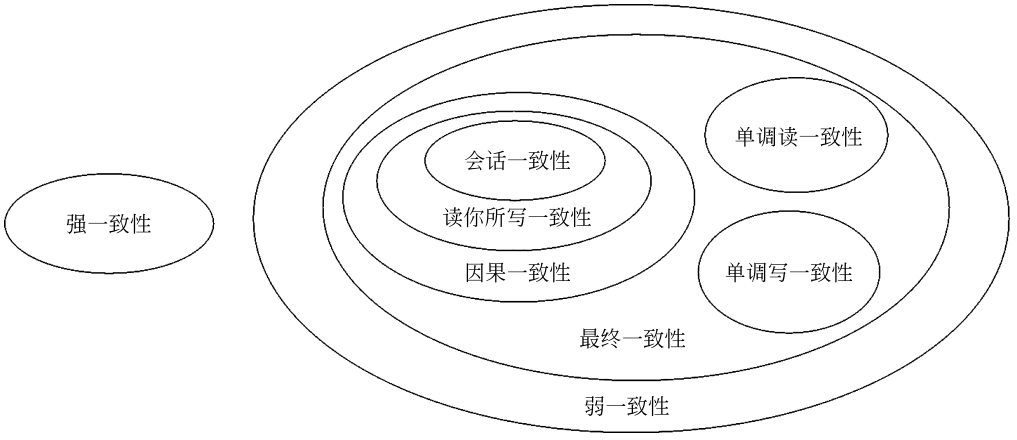


图 5.7　不同一致性的关系

5.2　共识设计的理论限制

数学家都喜欢对问题先确定一个最坏的理论界限。那么，对于共识问题，其最坏界限在哪里呢？很不幸的是，在推广到任意情形时，分布式系统的共识问题是无通用解的。这比较好理解，当分布式系统中多个节点之间的通信网络不可靠的情况下，即大多数节点之间的消息都出现丢失的情况下，很显然，无法确保实现共识。那么，对于一个设计得当，可以大概率保证消息正确送达的网络，是不是就一定能保证达成共识呢？

科学家证明，即使是网络通信可靠的情况下，可扩展的分布式系统的共识问题，其通

用解法的理论下限是没有下限，即仍然可能无解。这个结论称为 FLP 不可能原理。在本节的剩余部分，将对共识问题的理论界限展开描述，以让读者对共识问题有一个基本的认识。

5.2.1　FLP 不可能原理

即使在网络通信可靠的情况下，分布式系统的共识问题也无通用的解法，这个结论称为 FLP 不可能原理。FLP 不可能原理表明了在分布式情景下，无论任何算法，即使只有一个进程中断，对于其他非失败进程，也存在着无法达成一致的可能。FLP 不可能原理的具体表述如下：在网络可靠，但允许节点失效（即便只有一个）的最小化异步模型系统中，不存在一个可以解决一致性问题的确定性算法。提出并证明该定理的论文 *Impossibility of Distributed Consensus with One Faulty Process* 由 Michael J. Fischer 等科学家于 1982 年发表[9]。FLP 不可能原理本质上告诉人们，不要浪费时间去为一个异步分布式系统设计能在任意情形都能实现共识的确定性算法。

为了正确地理解 FLP 不可能原理，首先要弄清楚“异步”的含义。异步与同步都是一种传输模型，为了更好地理解异步与同步的含义，下面对传输模型进行介绍。在分布式系统中，传输模型主要分为以下两种。

（1）同步（Synchrony）。指系统中各个节点的时钟误差存在上界；节点所发出的消息，在一个确定的时间内，肯定会到达目标节点（传输时间有上界，且上界已知）。对于同步系统，可以很容易地判断消息是否丢失。

（2）异步（Asynchrony）。指系统中各个节点可能存在较大的时钟差异；节点所发出的消息，不能确定一定会到达目标节点，可能会丢失（传输时间无上界）。这就无法判断某个消息迟迟没有被目标节点响应是因为出了什么问题（目标节点故障还是传输故障）。在现实生活中，大多数系统都是异步系统。

FLP 不可能原理在原始论文中以图论的形式进行了严格证明。要理解这个原理并不复杂，下面通过一个简单但不严谨的例子来阐述该原理：四个人分别位于全球的四个不同位置对某一个提案进行投票，投票结果为“同意”或者“拒绝”。彼此之间仅能通过电话进行沟通，但是这四个人不可能一直保持在电话旁边，因为他们需要休息。比如某个时候，A 和 D 投票同意，而 B 投票拒绝，C 收到三人的投票，然后 C 睡着了。此时，A、B 和 D 将永远无法在有限时间内获知最终的结果，因为他们无法知道是因为 C 没有应答（由于时差问题，C 可能正在睡觉）还是因为应答的时间过长导致一直无法收到 C 的答复。如果可以重新投票，则类似情形可以在每次取得结果前发生，这将导致共识过程永远无法完成。

FLP 不可能原理实际上说明了对于允许节点失效的情况下，纯粹的异步分布式系统无法保证一致性在有限的时间内完成。即使在所有节点都不是恶意的（非拜占庭错误）前提下，包括 Paxos、Raft 等算法也存在无法达成共识的情况，只是在实践中出现这种情况的概率比较低。

FLP 不可能原理的另一种表述形式为异步的分布式系统不能同时保证活性和安全性。活性反映了来自客户端的请求最终会被处理，即好事总会发生。安全性反映了分布

式系统处理了来自客户端的请求后不存在不一致的状态。FLP 不可能原理表明了人们需要在活性与安全性上进行权衡折中。折中方向主要有弱化活性的传输模型假设或者弱化安全性的传输模型假设两个。

可以通过弱化活性的异步假设为同步假设以实现活性,这个方式的代表就是 PBFT。PBFT 中的安全性在异步网络的环境中都能得到保证。但是,PBFT 的活性不能在异步网络中达成,需要在同步模型中达成。换句话说,如果 PBFT 中的网络变成异步的话,可能永远不能完成客户端请求的处理。

可以通过弱化安全性的异步假设为同步假设以实现安全性。这个方式的代表就是比特币工作量证明,工作量证明的活性在异步网络的环境中都能得到保证。但是,工作量证明的安全性是不能保证的,所以会发生分叉的情况,即账本不一致。同时,工作量证明的安全性是一种概率上的安全性,只是账本随着时间被回滚的概率越来越低。另一方面,工作量证明的安全性是依赖同步模型的,只有在 10min 内完成全网广播才能保证不发生分叉。另外,Casper FFG 也是这种类型,但它实现的是一种确定性而非概率性的安全性。

5.2.2 CAP 原理

FLP 不可能原理告诉人们怎么才能设计出一个同时满足活性和安全性的系统,即对传输模型进行不同的假设以实现不同层次的活性和安全性。然而,当人们不能满足这些假设时,需要在牺牲一定的活性以获得更强的安全性,以及牺牲一定的安全性以获得更强的活性之间进行选择。CAP 原理就是通过描述这种权衡来指导分布式系统的设计。CAP 原理最早是 2000 年 ACM 组织的一个研讨会上提出的猜想,后来由 Lynch 等进行了证明并发表。该原理被认为是分布式系统领域的重要原理之一,对于分布式系统的发展有着极大的指导意义。CAP 原理本质上描述的是分布式系统在应用过程中三个特性的取舍,即分布式计算系统不可能同时确保以下一致性(Consistency)、可用性(Availability)和分区容忍性(Partition)三个特性,设计中往往需要弱化对某个特性的保证。

(1) 一致性。每次读操作都能得到最近写的结果或者返回错误。

(2) 可用性。每次请求都能返回一个非错误结果,但结果不需要是最近写的结果。

(3) 分区容忍性。任意节点间的连接中断或大大延迟,系统仍然能够工作。

所有的分布式系统都需要在网络分区的情况下继续工作。因此,分区容错性是所有分布式系统必须满足的。当相对于可用性更侧重一致性的情况下,如果系统因网络分区不能保证自身的数据是最新的,那么系统会返回错误或者超时。当相对于一致性更侧重可用性的情况下,如果系统因网络分区不能保证自身数据是最新的,那么系统会返回查询分区最新的结果(不能保证是全局最新的)。

注意:CAP 原理经常被人误解为在所有时间上需要在一致性、可用性和分区容错性这三个性质中抛弃某一个性质。如果没有发生网络分区,分布式系统正常运行,即同时满足可用性和一致性。

在区块链领域中,比特币更加侧重可用性。在每个网络分区中,工作量证明仍然能生成区块打包交易。然而,不同的网络分区会出现分叉,即账本不一致。当网络分区消失

后，一致性能够达到，分叉收敛，可用性和一致性都能达到。同时，PBFT 更加侧重一致性。在每个网络分区中，PBFT 若不能得到 2/3 的投票，PBFT 并不能生成区块打包交易。此时，不同的网络分区也不会出现分叉。当网络分区消失后，可用性能够达到，生成区块打包交易，可用性和一致性都能达到。

5.3　区块链共识算法

区块链共识算法本质上是为了解决拜占庭问题。拜占庭问题(Byzantine Problem)又称为拜占庭将军问题(The Byzantine Generals Problem)，讨论的是允许存在少数节点作恶(消息可能被伪造)场景下的一致性达成问题。拜占庭容错(Byzantine Fault Tolerant，BFT)算法讨论的是在拜占庭情况下系统如何达成共识。

5.3.1　拜占庭问题

1. 两将军问题

在拜占庭问题之前，就已经存在两将军问题(Two General Paradox)。两将军问题描述的是两个将军要通过信使来达成进攻还是撤退的约定，但是信使可以被敌军阻拦或者迷路导致消息无法送达(信息丢失或者伪造)。根据 FLP 不可能定理，两将军问题是无通用解的。

2. 拜占庭问题

拜占庭问题是 Leslie Lamport 等科学家于 1982 年提出用来解释一致性问题的一个虚构模型[10]。拜占庭是古代东罗马帝国的首都，由于其地域宽广，守卫边境的多个将军(即分布式系统中的多个节点)需要通过信使来传递消息，以对军事活动(系统中的某个提案)达成一致的决定。但由于将军中可能存在叛徒(系统中某些节点作恶)，这些叛徒将努力向不同的将军传递不同的消息，试图干扰共识的达成并使得某些将军做出错误的决定。拜占庭问题描述的就是在此情况下，如何让忠诚的将军们能达成行动的一致。论文中指出，对于拜占庭问题来说，假如节点总数为 N，叛变将军数为 F，则当 $N \geqslant 3F+1$ 时，问题才有解，由 BFT 算法进行保证。例如 $N=4, F=1$；$N=7, F=2$ 等。下面，通过简单的例子来说明当 $N<3F+1$ 时，拜占庭问题是无解的。

例如，当 $N=3$ 且 $F=1(3<3\times1+1)$时，即三个将军中存在一个叛徒时，那么忠诚的将军无法达成行动的一致。主要有以下两种情况。

(1) 提案者 A(忠诚的将军)不是叛徒，提案者发送一个“进攻”的提案出来，并发送给另外一个忠诚的将军 B 和叛徒 C；然而叛徒 C 可以向将军 B 宣称自己收到的是“撤退”的提案，这时将军 B 收到两个相反的提案，无法判断谁是叛徒，则系统无法达成一致。

(2) 提案者 A 是叛徒，分别发送“进攻”和“撤退”的提案给忠诚的将军 B 和将军 C，将军 B 和将军 C 都收到两个相反的提案，无法判断谁是叛徒，则系统无法达成一致。

Leslie Lamport 等人在论文 *Reaching Agreement in the Presence of Faults* 中证明，

当叛徒不超过 1/3 时，存在有效的拜占庭容错算法。反之，如果叛徒过多，超过 1/3，则无法保证一定能达到一致的结果。

5.3.2 拜占庭容错算法

拜占庭容错算法(Byzantine Fault Tolerance，BFT)是面向拜占庭问题的容错算法，主要用于解决在网络通信可靠但节点可能故障的情况下如何达成共识。拜占庭容错算法最早的讨论出现在 1980 年 Lesile Lamport 等人发表的论文 *Polynomial Algorithms for Byzantine Agreement*[11]，之后出现了大量的改进工作。长期以来，拜占庭问题的解决方案都存在复杂度过高的问题，达到 $O(N^{F+1})$，直到后来实用拜占庭容错算法的提出。

实用拜占庭容错算法(Practical Byzantine Fault Tolerance，PBFT)其实就是给全网消息的顺序进行共识，得到一个全局的序。在恶意节点不高于总数的 1/3 并在一个比较弱的同步假设的情况下，该算法能够同时保证安全性(Safety)和活性(Liveness)。该算法首次将拜占庭容错算法的复杂度从指数级降低到了多项式级 $O(N^2)$。PBFT 应用于联盟链的场景而不应用于公有链的场景有以下三个原因。

(1) PBFT 在网络不稳定的情况下延迟很高。

(2) 基于投票机制，而投票集合是有限的，否则无法满足少数服从多数的原则。

(3) 通信复杂度 $O(N^2)$过高，可拓展性比较低，一般的系统在达到 100 左右的节点个数时，性能下降非常快。

1. PBFT 的基本概念

(1) 客户端(Client)。向主节点发起请求的客户端，在区块链中往往跟主节点合二为一。

(2) 主节点(Primary)。提案发起者，在区块链中即为区块发起者，在收到客户端请求后生成新区块并广播。

(3) 验证节点(Backup)。提案投票者，在区块链中即为区块验证者，在收到区块后进行验证，然后广播验证结果对区块进行投票与共识。

(4) 视图(View)。一个主节点和多个备份形成一个视图，在该视图上对某个提案达成共识，在区块链中即为对某个区块达成共识。这里需要注意的是，不同视图的主节点一般是不一样的，即所有节点轮流做主节点，每个视图都会重新选择一个主节点。

(5) 编号(Sequence Number n)。在每个视图中由主节点指定的一个数字，即提案的编号，在区块链中可以理解为区块高度。

(6) 检查点(Checkpoint)。如果某个编号 n 对应的提案(区块)收到了超过 2/3 的确认，则称为一个检查点。

2. PBFT 的具体流程

PBFT 的核心为三个阶段：预准备阶段(PRE-PREPARE 阶段)、准备阶段(PREPARE 阶段)、提交阶段(COMMIT 阶段)。以图 5.8 为例，介绍 PBFT 共识正常进行的情况。图中的 C 为客户端，0、1、2、3 为共识节点，其中共识节点 0～2 正常执行，而共