Consensus Survey

星云研究院

2019 年 2 月
版本号:0.0.1

目录

1 介绍

区块链技术在近十年来有了长足发展，其中，共识算法是区块链最为核心的技术之一。十年来，各式区块链项目如雨后春笋办冒出，也为我们带来了丰富多彩的共识算法及新的思考。

本文将从技术的角度对现在区块链行业热门的共识算法进行调研。相比于现有的共识算法调研如 [1, 2]，本文具有如下特点。

1. 本文将以章节的形式，每一章节对一个共识算法进行着重介绍。在具体写作方面，本文将不拘泥于对共识算法进行传统意义上的分类，而是突出各共识算法在技术上的贡献以及在新的共识设计上能参考的点。我们认为，每个共识算法都有其独特之处，而传统意义上的分类（如去中心化程度，PoX 之类）并不能很好的将其反映出来。

2. 本文假设读者对最基本的共识算法，如 PoW，PoS，BFT 等有一定了解，不会做详细介绍。且本文主要作用是供内部设计共识参考用，故重点会放在对新共识设计的思考上。

2 背景

关于共识的研究最早可以追溯到 1959 年 [3]。计算机领域的共识主要研究分布式系统中所有节点达成一致性的问题。具体到区块链中的共识算法而言，其目的是决定出块权（及其奖励）的归属，需要满足下面两个基本性质。（各文献的描述存在细微差别）

- 一致性（consistency, safety）所有诚实节点最终（对某个提案）达成一致。
- 活性（liveness）所有诚实节点发起的交易最终都会被记录，

同时，一个好的区块链共识算法有如下基本指标：

- 去中心化：出块权不应集中在某个个体或团体手里。
- 抗女巫攻击：鉴于在区块链上建立新账户是没有成本的，共识算法的模型应该能够遏制用户通过大量建立新账户来提升自己获得出块权的概率。PoW 和 PoS 分别用算力以及才产作为竞争出块的依据来防止女巫攻击。

- 每秒交易次数（TPS）：TPS 反映出区块链系统的吞吐量。目前比特币的 TPS 为 10，以太坊为 15-25。
- 交易确认时间：交易确认时间影响链上交易的效率以及用户体验。目前比特币的确认时间为 1 小时，以太坊为 3 分钟左右。

2.1 思考

现在普遍认为区块链系统中也有不可能三角的存在，即去中心化，安全性和可扩展性不能同时达到。故设计新共识时需要有所取啥。

同时，女巫攻击以及去中心化（反独裁性）也难以同时保证。假设财产为 $a(b)$ 的账户能产生的收益（由共识算法决定的，如出块奖励）为 $f(a)(f(b))$。若能抵抗女巫攻击，理论上需要满足 $f(a+b) \geq f(a)+f(b)$。这样可以推出 $f(n) \geq nf(1)$，意味着大户的绝对统治（n 可以达到上亿级别）。

值得注意的是，上述分析不仅仅适用于 PoS 机制，即使是对 PoW 机制 $f(a)$ 也可以理解为财产为 a 的用户的挖矿效用，因为两者存在正相关关系。所以现在的比特币也可以理解为被矿池所统治的中心化系统，但同时浪费了额外的资源进行挖矿。相比而言，PoS 机制没有挖矿，但同时也是一个受资本操控的系统。

是否存在矿工证明和系统利益一致的共识?

例如，矿工通过提升节点性能来挖矿，并且同时能够提升整个系统的 TPS，但是这样可能造成安全性降低（见第二章，GHOST）。或者通过吸引新用户加入来挖矿，但这样除了 IOTA 这种 DAG 系统之外，节点数目超出系统容量之后反而会降低系统性能。

其他思考:

是挖矿还是看资产?

是否要有委员会/准入门槛?

是否支持智能合约?

是否考虑分片？或者用 DAG?

Part I

对工作量证明性能升级的思考

比特币作为最经典的区块链项目，对其所采用的工作量证明的相关研究也最为丰富。迄今为止，比特币白皮书 [4] 的引用次数已高达五千多次。本文选取共识层面几篇最据参考价值的文献进行调研。

3 GHOST

3.1 介绍

GHOST 的全称是 The Greedy Heaviest-Observed Sub-Tree，由 Yonatan Sompolinsky 和 Aviv Zohar 在 2015 年提出 [5]。其主要思想是对比特币最长链原则的一种修改。[1]

GHOST 实现的主要目标是，在保证安全性的前提下给出了比特币的扩容方案。该共识思想已经成为以太坊升级项目的一部分。

3.2 模型

众所周知，比特币中存在 51% 攻击，即需要假设作恶节点所占的算力比例不超过一半。这个假设在各大共识算法中都普遍存在。但 GHOST 认为，在比特币系统实际运作中，作恶节点实际需要占据的算力比例不需要 50% 那么多即可发起攻击。

GHOST 用 $tree(t)$ 表示在时刻 t 整个区块链（所有区块）的结构。由于分叉的可能性，这个 $tree(t)$ 不一定是一条链，而是“区块树”。

下面是 GHOST 模型的一些关键参数：

- λ_h，表示诚实节点的出块速率，即新区快加入 $tree(t)$ 的速率。值得一提的是，并不是所有的诚实节点出的区块都会最终出现在主链上，因为诚实节点之间也存在竞争，可能出现同时出块的情况进而产生分叉，

[1] 比特币工作量证明可能会出现两个矿工几乎同时发现新的合法区块的情况，当他们同时公布自己新挖出得区块并接在当前主链上，这样就产生了分叉。比特币 Nakamoto 共识约定当前主链存在多个分叉时，将最长（即区块高度最高的）的分叉链最为唯一合法的链。

- $q \cdot \lambda_h, 0 < q < 1$，表示作恶节点的出块速率。值得一提的是，作恶节点比诚实节点更团结——他（们）永远都只会在某一条秘密的链上进行挖矿，不会出现分叉。
- $s(T)$，s 函数用于确定主链成员：输入一个子树 T，输出一个区块，这个区块被确定为主链成员同时为下一个区块的父区块。比特币的最长链原则即让最长子链的起始区块作为 s 的返回值。
- β，表示主链增长的速度。注意只有 s 函数所决定的区块才能被主链记录。
- $TPS = \beta(\lambda, b) \cdot K$，其中 b 为区块大小（KB），K 为平均每 KB 数据包含的交易数目。

在比特币最长链原则下，针对传统的 51% 攻击，GHOST 认为，**需要竞争的是（诚实节点带来的）主链增长速度与作恶节点的秘密链的增长速度，即 β 与 $q \cdot \lambda_h$，而不简单的是用出块速率 λ_h 来对比**。所以，GHOST 把 $\frac{\beta}{\lambda_h}$ 作为所谓的"安全系数"，即当且仅当 $q > \frac{\beta}{\lambda_h}$ 时作恶节点将会成功攻击（传统认为需要 $q > 1$ 才能进行攻击）。

3.2.1 对比特币扩容的思考

比特币的低 TPS 一直是人们诟病的原因。两种简单的提升 TPS 的想法为：

- 增加每个区块的大小。（增大 b, 比特币的区块平均大小为 1MB）
- 减小区块出块的平均间隔时间。（增加 λ_h，目前比特币的平均间隔为 10 分钟）

根据定义，咋看之下这两种方式确实能提升 TPS，但 GHOST **指出这两中简单的方法都存在一系列问题。而这也是 GHOST 这篇论文的主要贡献**。

1. 增加区块大小 b

随着区块容量的变大，当矿工挖到新的区块时需要更长的时间来广播，同时用户也要花更长的时间来同步实时的区块链状态。这意味着新的区块需要花更长的时间被主链接收且被绝大多数节点验证，这样会导致更容易产生孤块，更容易发生分叉，安全性降低，主链增长速度反而会受到影响。有调研表明在维持基本的安全性的情况下比特币区块容量不能超过 8M。

2. 增加出块速率 λ_h

注意到当 λ_h 增加时，直接影响是安全系数 $\frac{\beta}{\lambda_h}$ 降低了。原因是诚实节点和作恶节点的出块速率都会增加，同时也会造成状态同步压力变大，分叉变多，主链增长速率 β 的提升受限。

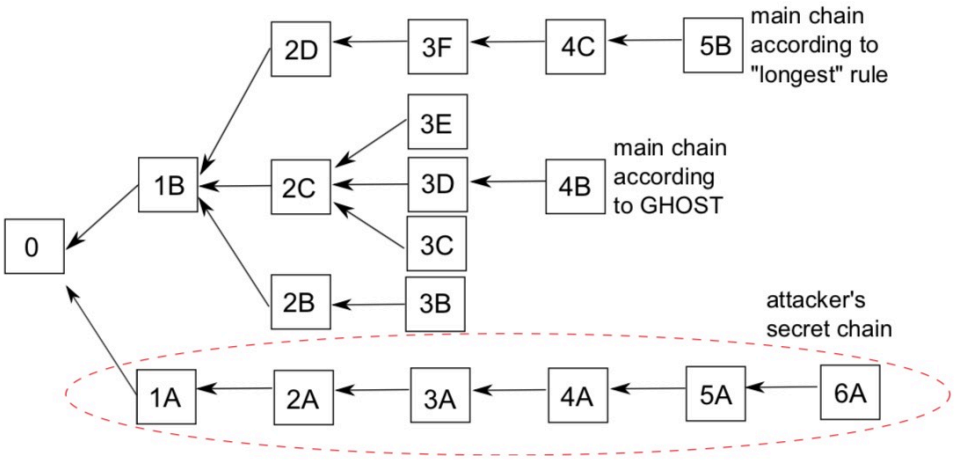

图 1: GHOST 中的最重子树

所以，GHOST 对简单的通过增加区块大小和降低出块时间的比特币扩容进行了否定。

3.3 GHOST 算法

如前所述，GHOST 的核心思想是将比特币的最长链原则改为最重子树原则。

对一个区块 B，定义 $subtree(B)$ 为以 B 为根的子树，$Children(B)$ 为 B 的直接子区块。写下来我们将实现 GHOST 中的 s 函数，用于决定一棵树被选为主链的块。

Algorithm 1: GHOST

Input: Block Tree T
set $B \leftarrow GenesisBlock$;
if $Children(B) = \emptyset$ then
    Return B and exit;
else
    Update $B \rightarrow \arg\max_{C \in Children(B)} |subtree(C)|$;
end
GOTO line 2;

其核心思想如图1（1B，2C，3D 这几个区块相比其兄弟区块拥有更重的子树。）

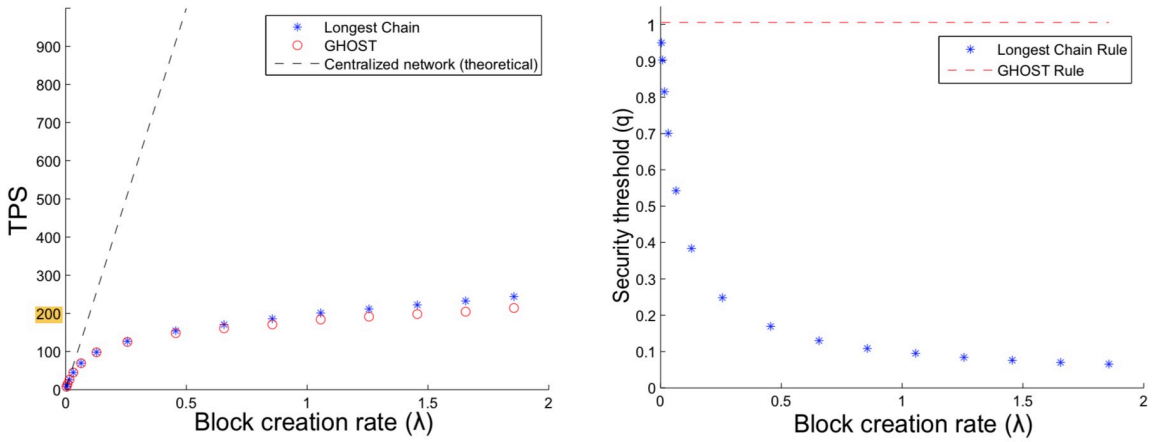

图 2: TPS 与安全系数的变化

3.4 性质分析

相比于比特币的最长链原则，GHOST 的主要性质是安全系数能独立于区块产生的速度，始终为 1（即能抵抗 50% 的攻击）。这个性质由下面两个引理保证：

引理 1. 定义 ψ_B 为区块 B 或者被所有节点接受，或者被所有节点所拒绝的时刻。那么 $Pr(\Psi_B < \infty) = 1$ 且 $E[\Psi_B] < \infty$

即这个引理保证最终一致性。和比特币一样，GHOST 不存在 finality 状态，故这个一致性也是概率上的。这个引理的证明思想是，出现不一致性仅当存在两个重量相同的子树。那么存在一个时刻，在一段时间内只挖出一个区块且在之后进行广播的过程中也没有新的区块被挖出，那么平衡将被打破，最重子树得以确定。

GHOST 的抵抗 50% 以下攻击由下面的引理决定：

引理 2. 如果 $0 < q < 1$，假设区块 B 已经在主链上出现的时间趋于无穷大，那么它被排出主链的概率趋近于 0。

由于这个结论与 λ_h 无关，这就为 GHOST 的扩容提供了保证。下面两张图给出了 GHOST 的 TPS 及安全系数随 λ_h 的变化。

所以，当 λ_h 达到一定数值后 TPS 可以达到 200 左右，若再继续增大整个系统的同步会受影响，安全性降低，故对主链增长及 TPS 的提升很小。

文章最后还有一些关于主链增长速度的定理。这里不做详细介绍。

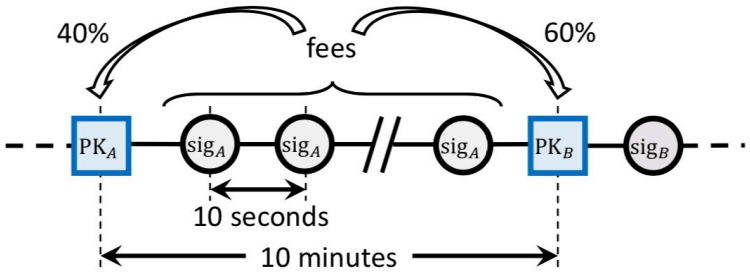

图 3: Bitcoint-NG 区块结构。其中 40% 和 60% 分别为一个子区块的交易费的分配方式，即 leader 获得 40%，挖出下一主块的矿工获得 60%

3.5 总结

GHOST 给我们最大的其实是不能简单地靠增加出块速率和区块大小来增加 TPS。设计共识算法是可以不局限于最长链原则。

4 Bitcoin-NG

4.1 介绍

Bitcoin-NG 是另一个针对比特币的扩容限制（scalability limits）提出思考的论文，发表在 NSDI16 上 [6]。其作者之一 Ittay Eyal 专攻区块链与博弈论相结合的领域，曾提出过私自挖矿（selfish mining [7]）等著名进攻方式。该论文提出基于比特币的扩展协议旨在满足下面的目标：

- Bitcoin-NG 的延迟只受限于网络传输的延迟。
- Bitcoin-NG 的带宽只受限于个人节点的处理容量。

4.2 模型

其主要思想是，允许矿工在挖出一个（主）区块的基础上，按固定速率不断出接在主块之后的子区块（microblocks），直到下一个主区块被挖出。这段时间内这个矿工叫做 leader，而只有包含 leader 有效签名的子区块才会被认可。子区块无需工作量证明，但产出子区块的速率被事先确定。

图3展示了 Bitcoin-NG 的区块结构以及交易费的分配方式。

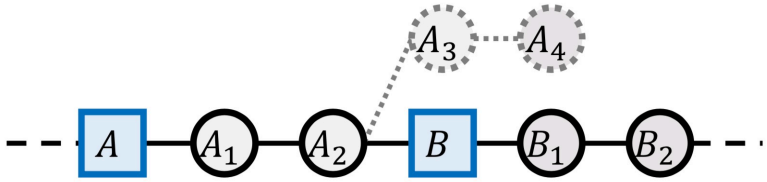

图 4: Bitcoint-NG 分叉情形：leader A 一直在出子区块，但是矿工 B 已经在子区块 A_2 后面挖出了一个主块。

值得注意的是，因为每个 leader 都会尽全力出子区块，所以，当下一个主区块被挖出的时候几乎会必定产生分叉，如图4。所以，文章指出当用户收到一个子区块的时候，在确认该区块在主链之前应该等一段时间。

同时，因为每个 leader 拥有自块的任意出块权，他可以选择恶意分叉他出的子区块以实现双花攻击（double spending）。Bitcoin-NG 的解决方式是，允许任何用户发送一个有毒交易（poison transaction），这个有毒交易是对分叉行为的一种举报，包含被裁减的分叉的第一个区块的内容，作为“欺骗证明（proof of fraud）”。每个 leader 需要等待一定的时间（maturity window）才能获得发区块的收益，在这期间内一旦被举报作弊 leader 将失去收益作为惩罚。同时，发出有毒交易当前 leader 能获得作弊者赔偿的 5%。

4.3 安全性分析

Bitcoin-NG 假设作恶节点的比例不超过 1/4。这是在考虑到了私自挖矿（selfish mining [7]）的情况下。鉴于大量其他共识的论文并没有把私自挖矿考虑进去，这里不做详细介绍。

文章接下来从博弈论角度分析 leader 的进攻方式。假设每笔交易费有 r_{leader}（待定）的比例归 leader 所有，剩下 $1-r_{leader}$ 归挖出下一个区块的矿工所有。假设 leader 拥有全网 α 比例的算力：

- 当 leader 创建包含某笔交易的子区块时，他不公布这个区块，而是在这个区块之后继续挖矿，期待自己能挖到后继的子区块，这样能获得该笔交易 100% 的交易费。若失败，该笔交易被部署在其他矿工挖出的子区块上，则该 leader 回归正常，继续在新的子区块上挖矿。这种行为不会提升该 leader 的收益当且仅当

$$\overbrace{\alpha \times 100\%}^{Win} + \overbrace{(1-\alpha) \times \alpha \times (100\% - r_{leader})}^{Lose\ but\ mine\ after\ txn} < r_{leader}$$

  当 $\alpha < 1/4$ 时，解得 $r_{leader} > 37\%$

- 对于某笔交易，leader 选择裁减掉这笔交易所在的区块，直接在其之前的区块上挖矿。若 leader 成功挖出下一个主区块，则他可以将该特定交易部署到自己接下来出的子区块上，并继续挖矿争取能挖到再下一个主区块以获得该笔交易的剩余交易费。相比于直接在这笔交易子区块挖矿，leader 的收益不会提升当且仅当

$$\overbrace{r_{leader}}^{Place\ in\ microblock} + \overbrace{\alpha \times (100\% - r_{leader})}^{Lose\ but\ mine\ after\ txn} < 100\% - r_{leader}$$

  当 $\alpha < 1/4$ 时，解得 $r_{leader} < 43\%$

故 $r_{leader} = 40\%$ 是一个可行的选择。

作为一个博弈论专家，作者接下来考虑的是各种可能的进攻方式以及各类指标（比通常的共识算法考虑的更多）并与比特币进行比较。这里列出一些重点供参考。

1. 关于挖矿难度：比特币的挖矿难度是动态调整的，故矿工有动机在难度较高的时候停止挖矿或转向其他公链挖矿，等到难度降低之后再回归。而在 Bitcoin-NG 中，由于子区块的出块速率是固定的，故只有主区块会受到这种行为的影响。故相对而言影响较小。

2. 关于分叉。Bitcoin-NG 承认会比比特币拥有更多的分叉，如图4所示。但是，在 Bitcoin-NG 中分叉消失的更快，因为一旦一个主区块被挖出则 leader 权改变，子区块分叉消失。但是对于主区块的分叉并没有解释很清楚。（见原文 5.2）

3. 文章提出了一些衡量区块链性能的指标，包括共识延迟，公平性，矿力利用率，修剪（分叉被确认忽略）时间，胜利（主链确认）时间等。文章的实验是基于不同的区块大小和出块评率绘制上述指标的曲线，并与比特币相比较。有兴趣的读者可以参阅原文查看结果。

4.4 总结

Bitcoin-NG 这种出小块的思想具有一定的借鉴意义，类似于一种锦上添花的功能，但和 GHOST 一样，对共识原型的贡献不大。同时，文章并没有很显示的证明本章节开头提出的关于比特币扩容的两点。另外，文章提出的各项指标实际被后续工作引用的不多，参考价值有限。

Part II

少量节点的强一致性保证：拜占庭容错

5 BFT 背景：拜占庭将军问题

BFT（Byzantine Fault Tolerance）的全称是拜占庭容错，是一个能实现状态机复制且能够容忍拜占庭节点的算法。这里的拜占庭节点的就是所谓的恶意节点，其行为可以是任意的（发送错误信息，任意长时间不响应信息，不按照协议运作等等，且各个拜占庭节点之间可以合谋）。

介绍 BFT 要从广为人知的拜占庭三将军问题（BGP，Byzantine General Problem）说起。这两者容易混淆，但实际 BFT 指的是一系列容错问题，而 BGP 是 BFT 的一个特例。

BGP 问题由 Lamport 在 1982 年提出 [9]。所谓三将军 BGP 问题指的是，有一个指挥官和两个将军，他们需要对进攻或者撤退达成统一的决定。他们之间有一个叛徒，这个叛徒就是我们之前提到的拜占庭节点，可以任意行为（如撒谎，不回复等）。将军们（包括指挥官）之间可以任意轮进行点对点交流。问题要求，如果指挥官是诚实的（不是叛徒），那么所有诚实将军的决定要和指挥官的决定一致。如果指挥官是叛徒，那么所有的将军要做出一致的决定。

常规的 BGP 问题分为两个部分，分别针对**同步网络模型**和**异步网络模型**。

同步网络：所有节点的消息传输的延迟不会超过某个定值 t（这个 t 是有限且已知的）。

异步网络：所有节点的消息传输延迟可以是任意的（对于诚实节点的只能保证消息一定可达，但是延迟上限未知，且随时间是可变的）。

同步网络是一种理想的网络环境，而我们熟知的 BGP 在同步网络环境的描述如下:

同步网络环境下三将军 BGP 问题不可解，四将军或以上 BGP 问题可解

这个问题的证明是如果指挥官是恶意的，那么他可以给两个将军发出不同的指令。如果指挥官不是叛徒，那么两个将军交流时其中一个将军可以谎报指挥官的指令。具体细节这里不介绍，可以参阅知乎专栏 [10]。值得注意的是这里的不可能与后面提到的 FLP 没有关系，因为是同步网络环境，仅由数学上的推导即可得出。

这个问题（同步 BGP）在四将军（包括指挥官）情况下是有解的。同时可以推

广为，假设拜占庭节点（将军）的数目为 f，当且仅当将军总数大于等于 $3f+1$ 时 BGP 问题是有解的。

当然对于同步三将军 BGP，Lamport 提出的解决方案就是加入签名 [11]，加入了签名之后，即便全世界只有两个将军是诚实的而其他将军全是叛徒，这两个将军也能达成共识。这其实就是前段时间流传的 V 神发现了 99% 共识算法。

异步网络环境下 BGP 问题无解

如前所述，异步网络环境下消息传输延迟可以使任意长且可变。这种情况下只要存在一个叛徒 BGP 问题就无解。这是由分布式系统的经典结论，FLP 不可能定理得出 [12]。主要原因是当一个节点长时间未接收到消息时，他无法判定对方是恶意节点故意不发还是对方是诚实的但因延迟消息还未传过来。书 [13] 第三章有对 FLP 问题的详细证明。

这里注意到拜占庭节点“不说话”比“说假话”在异步 BGP 问题中更具杀伤力，因为有了签名的存在，任何节点能伪造的信息有限（容易被查出），而“不说话”让其他节点无法分辨是恶意节点还是消息没达到。退而求其次的，人们开始在给定某些假设的情况下研究 BGP 问题：假设指挥官一定会发信息（可以说假话，但一定会到达），且可以引入签名，这种情况下三将军 BGP 能不能解决。这种假设在 [10] 叫“弱终止假设”（或者可以理解为半异步假设。这个假设仅仅针对 BGP 问题，在实际中并不常见）。

弱终止假设环境下三将军 BGP 问题无解，四将军或以上问题有解

同样，该结论可推广为当且仅当 $3f+1$ 或以上将军可解。

BGP 的结论总结如下：

环境	同步（无签名）	同步（带签名）	异步	弱终止
有解所需最少将军数	$3f+1$	2	$f>0$ 必无解	$3f+1$

6 实用拜占庭容错协议：PBFT

6.1 背景

BGP 问题是 BFT 问题的一个特例：它假设所有节点对指挥官的身份形成了共识。实际的 BFT 问题里不一定有指挥官的存在，但拜占庭节点仍然存在。以下用 f 表示拜占庭节点的数量，用 n 表示总结点数量。解决问题的思路仍是沿用 BGP 问题的解法。

上一章我们提到了同步模型与异步模型的概念。这里指出，当我们研究区块链的共识算法的时候，鉴于区块链的大规模和去中心化，对网络条件的同步性假设越少越好。对于状态机复制问题，目前各文献对同步模型的研究已经不多。

然而异步环境共识不可避免的受限于 FLP 不可能定理：

定理 1. 当 $f > 0$，不存在一个确定的算法总能在异步网络环境下达成共识。

所以，各大著名的异步共识算法实际都做了各式各样的假设。如 Ben-Or 算法 [14] 引入随机性，即一定情况下需要节点各自通过掷硬币来选择决定，同时期望若干轮后所有诚实节点能掷出同样的结果。该算法仅能容忍 $f < 1/10n$ 的拜占庭节点，同时达到共识的期望轮数是指数级别。类似共享硬币 [15] 的思路能够提升时间复杂度，但容错率仍然很低。

在正式介绍 PBFT 之前，这里简单介绍状态机复制问题，指的是一系列节点（初始状态相同）以相同的顺序执行一系列指令的问题。这个要求实际上比共识更强，因为像现在如比特币的共识是概率性的，即很难被扭转，但并不存在所谓的 Finality 状态，即完全一致性。而状态机复制需要保证完全的一致性。另外，如果一个算法能实现状态机复制（可能各文献说法会有不同，但这里我们默认状态机复制已经满足一致性），则可由该算法实现共识问题。

为什么要用 PBFT?

最初，Paxos[16] 算法的提出解决了同步网络下没有作恶节点（但可能失效）的状态机复制问题。然而，在区块链被提出之前，PBFT 等一系列实现异步状态机复制的算法并没有受到重视。其原因是在传统的中心化系统中，对指令时序的一致性要求并没有那么强烈，往往会根据具体业务实现不同层面的一致性和安全性指标——一般而言，双机备份就足以满足大部分业务需求，同时大部分情况下少量指令顺序的改变并不会对最终结果产生太大的影响。

区块链的诞生让尘封已久的 BFT 重建天日：由于区块链的大规模以及去中心化（意味着难回滚，不可篡改，冲突难以调和，需要备份的节点多）特性，人们需求一个状态机复制的强一致性保证。而其中最著名的算法就是 PBFT，其全称为 Practical Byzantine Fault Tolerance [17]，发表在 1999 年 OSDI 上。

6.2 模型介绍

传统的状态机复制算法或者依赖于同步网络环境的假设，或者在实际运行时间太慢。PBFT 旨在提出能实现 1/3 异步拜占庭容错的确定性协议，用于实现服务器副本按顺序执行一系列操作的问题。论文的假设是，在任何时刻 t 消息传输延迟 $Delay(t)$

不会无限制增长。这是一个非常弱的假设，在实际中通常是满足的。而这个假设的存在导致 PBFT 算法和 FLP 定理并不矛盾。

定义 $\mathcal{R} = \{0, 1, ..., |\mathcal{R}|\}$，其中 $|\mathcal{R}| = 3f + 1$。PBFT 将所有服务器分为一个主节点和其他副本节点。每个节点在他的视角看到的信息称为一个 view，每个节点的 view 允许不同。一般而言，主节点由 $p = v \ mod \ R$ 决定，其中 v 表示 view 的轮数，在算法里会不断更新。

大致而言，PBFT 算法执行包涵下面 4 个步骤:

- 客户端向主节点发送一个操作请求
- 主节点向所有副本节点广播这个请求
- 所有副本节点执行这个请求，并且给客户端发送一个带执行结果的回复（reply）
- 当客户端收到至少 $f + 1$ 个相同的回复时，即得到该操作请求的执行结果。

客户端等待 f+1 个从不同副本节点得到的同样响应，同时需要保证签名正确，并且具有同样的时间戳 t 和执行结果 r。这样客户端才能把 r 作为正确的执行结果，因为失效的副本节点不超过 f 个，所以 f+1 个 replicas 的一致响应必定能够保证结果是正确有效的。

如果客户端没有在有限时间内收到回复，请求将向所有副本节点进行广播。如果请求已经在副本节点处理过了，副本节点就向客户端重发一遍执行结果。如果请求没有在副本节点处理过，该副本节点将把请求转发给主节点。如果主节点没有将该请求进行广播，那么就有认为主节点失效，如果有足够多的副本节点认为主节点失效，则会触发一次 view change，即主节点轮换。

对于一个客户端而言，在他的视角内仅仅的是简单的发送一个请求以及接受多个回复的过程。所以 PBFT 的核心部分在于服务器节点之间的交互，被分为三个部分:pre-prepare, prepare, commit。见 图5。

- pre-prepare: 主节点分配一个序列号 n 给收到的请求（m），然后向所有子节点广播 pre-prepare 消息 $+m$，同时将请求消息 m 写入消息日志。消息的格式为 $\left\langle \langle PRE - PREPARE, v, n, d \rangle_{\sigma_p}, m \right\rangle$，这里 v 是视图编号，m 是客户端发送的请求消息，d 是请求消息 m 的摘要（摘要在文章中的意思即哈希值 $D(m)$，密码学上难以做逆运算），σ_p 为主节点 p 的签名，n 是消息的序号，必须在 h 和 H 之间（称为水线（watermark）限制。这样可以防止拜占庭节点使用很大的 n 消耗序号空间）。注意为减少传输开销，此处客户端的原始请求并没有包括在 pre-prepare 消息内。副本节点只会接受格式，签名，view，水线限制均满足的 pre-prepare 消息。

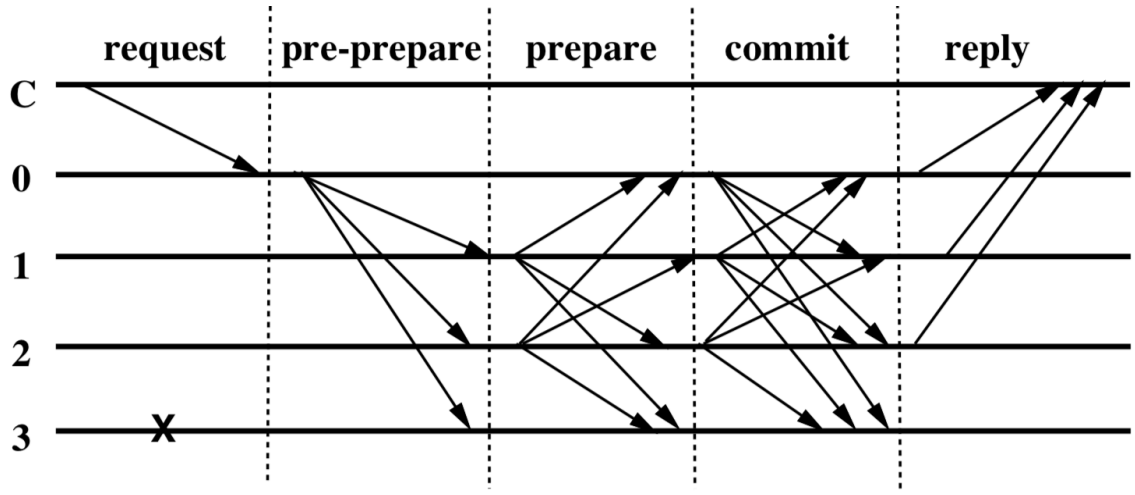

图 5: PBFT 流程

- prepare: 一旦副本节点接受了 $\langle\langle PRE-PREPARE, v, n, d\rangle, m\rangle$，则进入 prepare 阶段，同时。该节点向所有副本节点发送准备消息 $\langle PREPARE, v, n, d, i\rangle_{\sigma_i}$，并且将两者写入自己的消息日志。定义 $prepared(m, v, n, i)$ 为真当且仅当副本节点 i 已将下列消息写入日志：消息 m，针对 m 且 view 为 v 且序号为 n 的 pre-prepare 消息，$2f$ 个从不同副本节点发来的与 pre-prepare 消息匹配的 prepare 消息。预准备阶段和准备阶段确保所有正常节点对同一个视图中的请求序号达成一致。预准备阶段和准备阶段确保所有正常节点对同一个视图中的请求序号达成一致，即 $prepared(m, v, n, i)$ 和 $prepared(m', v, n, j)$ 不能同时为真（若 $m \neq m'$）。

- 当 $prepared(m, v, n, i)$ 为真的时候，副本节点 i 将 $\langle COMMIT, v, n, D(m), i\rangle$ 向其他 replicas 广播，于是就进入了确认阶段。此阶段定义了两个函数，$committed(m, v, n)$ 为真的条件为：存在 $f+1$ 个正常副本节点集合中使得其中所有副本节点 i 的 $prepared(m, v, n, i)$ 为真。$committed-local(m, v, n, i)$ 为真的条件为：$prepared(m, v, n, i)$ 为真，并且 i 已经接受了 $2f+1$ 个 commits（包括自身在内）与 pre-prepare 消息一致。commit 与 pre-prepare 消息一致的条件是具有相同的视图编号、消息序号和消息摘要。对某个正常节点 i 来说，如果 $committed-local(m, v, n, i)$ 为真则 $committed(m, v, n)$ 也为真。这个不变式和视图变更协议保证了所有正常节点对本地确认的请求的序号达成一致，即使这些请求在每个节点的确认处于不同的视图。更进一步地讲，这个不变式保证了任何正常节点的本地确认最终会确认至少 $f+1$ 个的正常副本。

视图变更协议：所谓视图变更即将 v 加 1，进而更换主节点。视图变更协议在主节点失效的时候仍然保证系统的活性。视图变更可以由超时触发，以防止副本节点无期限地等待请求的执行。

如果仅仅是这样，这个拜占庭容错在异步系统中是无效的。因为这个算法会在延迟超过阈值 t 的时候失效，而异步系统的定义就是找不到这样一个阈值 t。PBFT 采用的方法是：在 $p = 1$ 的时候，把阈值设为 t_1，然后如果超时，则增加这个阈值，变成 $t_2 > t_1$。这样保证无论这个系统的延迟有多大，**只要延迟不会无限增长**，PBFT 都能保证最终达成共识。

论文最后证明 PBFT 的安全性（safety）与活性（liveness），并进行了实验测试性能。这里不详细介绍，有兴趣的读者可以参阅全文。

6.3 思考

PBFT 的好处是能够实现异步一致性，并且已经被大量区块链主要是联盟链项目所采用。其缺点在于共识的达成用到了长达三轮的消息传播，相对较长。在节点数量较少的情况下仍不失为一种最优的选择。

7 投机 BFT：Ouroboros-BFT 与 Zyzzyva

对 BFT 的高通信复杂度优化的思考引出了一系列所谓投机 BFT 算法，即，在更强的假设前提下（网络环境更好或拜占庭节点更少）能有更好的性能（performance）。

7.1 Zyzzyva

Zyzzyva[18] 由 Lorenzo 等人在 2007 年提出，发表在 SOSP 上并被评为 Best Paper。这个命名据说是选取的字典表里的最后一个单词，意味着 Zyzzyva 是这一系列算法的终结（然而之后还是出现了更好的算法）。

Zyzzyva 的协议分为检查点（checkpoint）协议，视图转换（view change）和一致性（agreement）协议。这里我们介绍后两部分。

Zyzzyva 其大致结构和定义和 PBFT 类似，（f 个拜占庭节点，一共 $3f + 1$ 个节点，异步网络模型）实现的目标是能有更快的消息复杂度，其核心思想在于在请求的确定被完全确定之前就开始执行。

7.1.1 流程

- 1. 客户端发送请求给主节点

- 2. 主节点接受到请求，给其设置编号，并将请求广播到所有副本节点。
- 3. 副本节点接收到有序（ordered）请求，并投机的执行它们，并且给客户点发送回复。
- 4. 客户端开始接受副本节点发出的回复，根据一定时间内收到的回复数量可以分为下面三种情形：
  - 4a. 若客户端收到 $3f+1$ 个回复（和他的请求匹配的），则认为请求被成功执行，完成请求（complete the request）
  - 4b. 若客户端收到的回复个数在 $2f+1$ 和 $3f$ 之间，则其生成一个 commit 信息（包含发送这 $2f+1$ 个回复的节点的 ID，签名，请求内容等），并广播给所有的副本节点
    - 4b.1 当副本节点接收到一个合法的 commit 信息时，生成一个 local-commit 信息并发送给客户端（若收到的 commit 信息包含的记录和本地记录不一致，则发起 view change（更换主节点））
    - 4b.2 当客户端收到 $2f+1$ 个合法的 local-commit 信息时，完成请求。系统能保证即使存在 view change，所有诚实的副本节点都会执行请求。（若客户端在限定时间内没有收到 $2f+1$ 个 local-commit 信息则跳入 4c 步骤）
  - 4c. 若客户端收到的回复个数少于 $2f+1$ 个，则客户端将请求重新发送给所有副本节点并抄送给主节点（以获得序号）。
    - 4c.1 当副本节点收到客户端的请求信息时，若该请求拥有最高的时间戳（timestamp），则副本节点再发送一个 confirm-req 信息给主节点 p 并开始计时。若在限定时间内收到主节点发送的有序请求（order-req），则如前所述正常执行该请求。若在限定时间内没有收到，则发起 view change，同时将 confirm-req 广播到所有副本节点。[2]副本节点收到 confirm-req 时向发送方发送从主节点发来的有序请求。
    - 4c.2 当主节点收到 confirm-req 信息时，主节点如第二步所述发送有序请求。
  - 4d 当客户端发现请求不一致时，发送 proof-of-mistake（POM）给所有副本节点并开始 view change。（文章提到发起 view change 并不会影响请求被执行）

Zyzzyva 的视图转换协议运作当且仅当下面两种情况发生

[2]注意到这里出现了 n^2 级别消息传输复杂度。

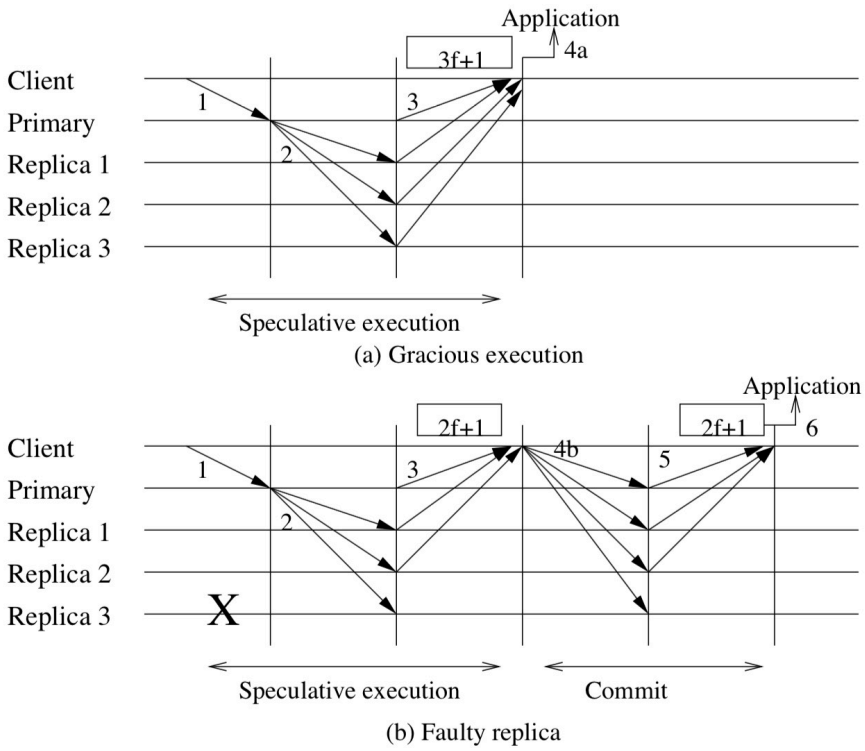

图 6: Zyzzyva 共识算法流程

- 当主节点查出有拜占庭行为
- 当 $f+1$ 个节点发起 view change（文中的方式为发送 i-hate-the-primary 信息）

图6演示了 Zyzzyva 的共识流程。

文章之后给出了安全性及一致性的证明，以及用实验对比 PBFT 的性能。这里不详细介绍。

7.1.2 总结

Zyzzyva 和 PBFT 相比，运气好的时候执行的更快，但是当主节点更换频繁时效果可能更差。

值得一提的是，Zyzzyva 团队之后又提出了致力于在更多的拜占庭故障发生时也能保持高性能的算法，发表在 2009 年 NSDI 上 [19]，称作 Aardvark，据说是字典表的第一个单词（表达他们已经经历了一个轮回）。

7.2 Ouroboros-BFT

Ouroboros-BFT 由 Ouroboros（Cardano 项目，代币名称: ADA）团队于 2018 年 9 月提出 [20]。Ouroboros-BFT 和他们本身的 Ouroboros 共识（会在下个部分介绍）联系不大，仅仅是提出一种投机 BFT 协议。

文章解决的是区块链的共识问题。文章的主要贡献是提出一种交易确认时间极快（通常为 $O(t)$，t 为拜占庭节点数目），传播复杂度较小（最优情况为 $O(n)$，最坏情况为 $O(nt)$），容错为 $t \leq n/3$ 的共识协议。

假设

Ouroboros-BFT 与传统 BFT 算法最大的不同是他对网络环境做了很强的假设，主要在以下两个方面:

- 文章所有的结论都是在同步网络模型下进行分析。文章假设任何异步网络环境都是暂时的，一旦网络环境回归同步，则共识协议会收敛回安全状态。
- 文章假设存在一个全局同步的时钟，并在文章的最后给出了如何模拟这个时钟的方法。

算法

Ouroboros-BFT 将时间看做若干个时间段 $sl_1, sl_2, \cdots$，并假设该时间段发出的消息能在下个时间段被所有的服务器接受。同时，每个服务器维护一个区块链 $B_0, B_1, ...$，其中 B_i 包含的信息为前一个区块哈希值，当前的时间段，打包的所有交易，对时间段的签名，

具体算法如下:

1. 当服务器收到一笔合法的交易 tx 时，将其放到交易池里面，其在交易池中“存活”u 个时间段。服务器可以选择性的给客户端回复一个收据用以确认交易。
2. 当服务器意识到一个更长的区块链的存在时，若验证了其合法性（主要验证区块提出者编号等于时间戳 $mod\ n$），用之替换其本地的区块链。
3. 当服务器发现轮到自己出块时（同上，编号同余于时间戳），打包所有交池中交易，并且按照格式进行签名同时广播区块。
4. 当服务器收到询问请求时，回复已被最终确认（finalize）的区块
5. 当客户端收到 r 个针对某交易的收据时，则该交易被最终确认。

文章接下来分析协议的活性，一致性保证等等。结论主要在以下两个方面:

1. Ouroboros-BFT 能保证一致性和参数为 $5t+2$ 的活性，即任何交易都会在 $5t+2$ 个时间段内被打包

2. Ouroboros-BFT 能保证参数为 $5t+2+n-r$ 的确认时间，即任何交易都会在 $5t+2+n-r$ 个时间段内被最终确认。

文章之后还研究了一种 covert 设定 [21]，具体读者可参阅原文这里不做详细介绍。

总结 Ouroboros-BFT 因为强假设太多，实际可参考价值并不大，更多的是提供了一种基于全局时钟的处理方案。根据分布式经典论文 [22] 中的介绍，实际中能够确定的只是行为间的偏序关系（Partial order），往往会存在多个符合给定偏序关系的全局顺序。通常的假设是每个进程有一个本地的时钟，但可以在信息交换的过程中不断维护和更新本地时钟。故全局时钟的存在是一种理想的环境。当然，全局时钟的设定目前也被一些其他文献所接受，如 google 的 [23, 24]。而在区块链中，通常以区块高度或类似的，轮数（round）这种全局变量来实现全局时钟的效用。

文章还提到了 SBFT[25] (2018 年, cite 9 次)，另外一种投机 BFT 算法。

7.3 总结与思考

投机 BFT 属于基于 PBFT 的扩展，但我们认为实际操作中还是 PBFT 更具参考价值。但是鉴于大部分公链难以承受 PBFT 的高复杂度，需要针对具体环境进行优化。投机 BFT 给出了优化的参考方案。

事实上，本部分提到的所有文献除了 Ouroboros-BFT，都不是关于区块链的。他们解决的是更为本质的状态机复制问题，而大量区块链项目仍然需要对网络环境作一定量的假设，这在下部分介绍中可以看到。作为区块链共识协议的设计者，需要了解状态机复制的工作原理和基本方法，将其作为一种最高要求的实现，但同时也不需要拘泥于此，适当的对网络环境做出假设，以牺牲一部分理论高度来换取应用于实际的性能是更为可行的方案。

Part III

基于随机数的权益证明机制

如果说工作量证明（Proof of Work）存在一个最为可能的替代品的话，那无疑是广为人知的权益证明（PoS）。对 PoS 的基本概念这里不多做介绍。值得一提的

是，目前有个很大的误区是以太坊用的就是 PoS 机制，这个说法并不准确。以太坊现在的 Casper [26] 版本在出块方式上仍用的 PoW，即挖矿机制。只有每隔 50 个区块需要通过委员会抵押投票的方式来确定一个检查点（checkpoint）以实现最终确认性（finality）。（成为委员会成员需要调用特殊的智能合约并抵押至少 1500ETH，一个检查点需要有抵押总财富的 2/3 以上才能被最终确认）

PoS 并不是想象中的所谓谁钱多给谁，因为区块链的出块权需要随机性，所以，所谓的 PoS 目前解决的并不是由谁出块的问题，而是解决出完块后投票确认的问题。在一系列 PoS 相关的文献里，出块权可以不做限定，而投票过程其实是另外一种达成共识的过程，可以与也可以不与所持财富（Stake）挂钩。本部分介绍的三大类似机制，Dfinity，Algorand 和 Ouroboros，都符合上述要求。这类文献为我们设计基于投票的共识机制提供了参考。

8 Dfinity

Dfinity 是 2018 年在 arxiv 上的一篇论文，但作为代表引起了广泛关注。其本质做的是一个基于认证的联盟链，一切共识协议均根据可验证随机函数（Verifiable random function）实现，具有较快的传输复杂度（基本线性）与同步高安全性。

8.1 四层结构

首先 Dfinity 拥有委员会的概念。委员会成员从所有经过认证的客户端中随机产生，并每一轮都会更换。

Dfinity 的共识机制共有四层，认证（identity）层，随机种子（random beacon）层，区块链层和公证（Notary）层。下面简要介绍每一层的功能及实现。

1. **认证层**

   认证层的目的是实现客户端（矿工）的申请与认证。所有申请进入认证层的客户端都需要提供包括资产抵押在内的认证。而一旦被发现有作恶行为，客户端会受到除失去出块奖励外，扣除抵押资产的惩罚。认证层支持任何提供资产抵押的客户进入。

2. **随机种子层**

   该层通过运行 VRF，由一个委员会运作，用以实现在每一轮产生随机种子 ξ_r，其中 r 为当前轮数，用以决定所有客户端的排名。该随机种子以门限签名的算法实现，具有不可预测性以及可验证性（这样防止了委员会成员消极怠工或者

合谋分叉的可能）。同时该随机数实现是非交互性的，不需要运行拜占庭协议，时间开销较小。

3. 区块链层

   同传统的区块链一样，该层用于客户端打包交易并上链。在 Dfinity 中允许任何客户端提交区块，但区块的权重由提出者的排名决定，而提出者的排名又是根据上一层的随机种子随机被随机分配。一般而言，Dfinity 协议默认客户端将区块接在总权重最高的链上。

4. 公证层

   该层包括实现区块的公证化（Notarization）和最终确认。该层同样由委员会运作。一个公证本质上是对一个区块的认证签名的集合（一般认为需要委员会大多数成员提供签名才算被公证）。委员会只会对当前轮收到的排名最高的区块进行公证签名。注意到这里的公证化并不意味着最终确认，因为一轮可能有多个区块被公证进而产生分叉。但只要下一轮的区块提出者是诚实的，他就只会往某一个分叉上接，这样分叉在下一轮即会消失。

8.2 模型

Dfinity 对容错的基本假设为 $|U| > \beta f(U)$。其中 U 为全节点集合，$f(U)$ 为拜占庭节点的数目，$\beta > 2$，即，至少满足 1/2 容错假设。同时，委员会成员的数目为定值 n。Dfinity 文章证明对于合适的 n, β，能以高概率满足，对于每一个委员会 G 都有 $n > f(G)$，即，对于每个委员会内部都满足拜占庭容错条件。一个例子为 $\beta = 3, n = 405$ 时，一个委员会不满足 $n > f(G)$ 的概率为 2^{-64}。

Dfinity 的网络环境假设为存在一个已知的描述网络延迟的随机变量 Y，所谓半同步（semi-synchronous）。我们认为其本质就是同步网络假设。

Dfinity 的一个区块 $B = (p, r, d, z, o)$，分别对应前一个区块的哈希值，当前轮数，前一个区块的公证，所存数据（包括打包的交易）和该区块创建者。一个区块链 C 为上述区块的集合。

根据第 r 轮的随机种子 ξ_r，可以随机生成每个客户端的排名（通过 Psudo-random permutation [27],Algorithm 3.4.2P）$\pi_r(i)$。一个区块的排名等于其创建者的排名 $rk\ B = \pi_r(o(B))$。区块的权重为其排名的一个递减函数，如 2^{-x}。一条区块链 C 的权重为其上所有区块权重之和。图7是计算区块链权重的一个例子。

区块提出矿工将区块接到权重最高的链上。注意，公证层被公证的区块仍是当前轮排名最高的区块。同时将新区块进行广播以请求公证。

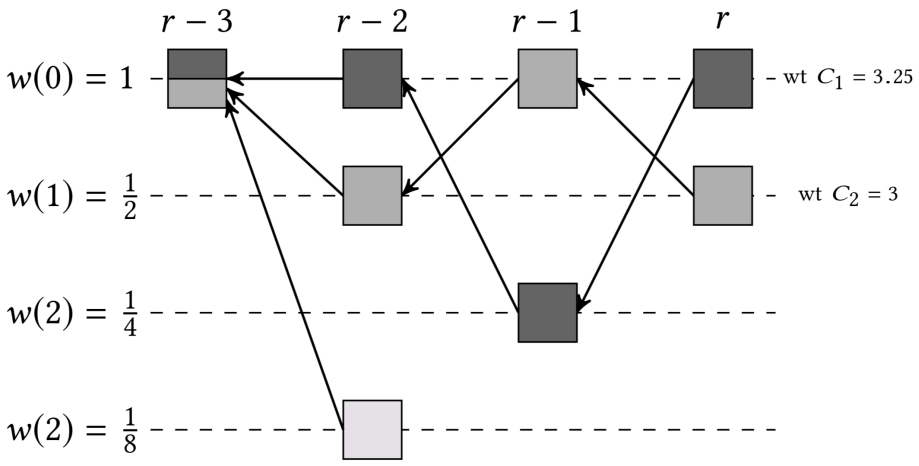

图 7: Dfinity 区块链权重举例。四条横轴虚线对应四个矿工且权重分别为 $1, 1/2, 1/4, 1/8$。图中区块不同深浅对应与该区块属于哪一条区块链（共 3 条，C_1 颜色最深，C_3 颜色最浅）。通过计算表明链 C_1 拥有最高的总权重

区块公证首先 Dfinity 的公证是每轮进行的。如果区块被揭露得过晚则永远不会被公证。这样有效的防止了私自挖矿（selfish mining）进攻。

一个区块被公证了当且仅当它收到了大部分公证者的签名。对每个公证者（委员会成员，也是客户端或副本节点 replica），具体的公证算法用文字描述如下：

- 设当前轮数为 r。等待 Blocktime 时间
- 如果没有收到任何被公证的区块，则选择第 r 轮接收到的所有区块中排名最高的一个，对其进行签名并广播。
- 如果收到了被公证的区块，则 $r = r + 1$

文章把当前轮只有一个区块被公证的情况称作常规操作（normal operation），然后针对常规操作分析了确认时间及安全性等，但这本身是一个比较强的假设。

最终确认之所以公证化不等于最终确认，是因为因传输延迟等影响存在多个区块被公证的情况。这是还需要一个最终确认（finalize）算法。

该算法描述如下：在第 r 轮，节点等待 T 时刻以接受第 r 轮被公证的区块并放入集合 N_r，随后执行 $Finalize(r-1)$，即，把 N_{r-1}（即，第 $r-1$ 轮被公正的区块集）的最长公共前缀标为最终确认。图8指出了最终确认的过程。

同时需要基于假设：在 $Finalize(h)$ 被执行时，N_h 包含第 h 轮所有被公证的区块。

关于最终确认的主要定理为，在常规操作下任何交易都会在接收到两个确认信息后（被公证区块 B_r 打包，同时区块 B_{r+1} 接在上面）加上网络延迟时间的两倍内

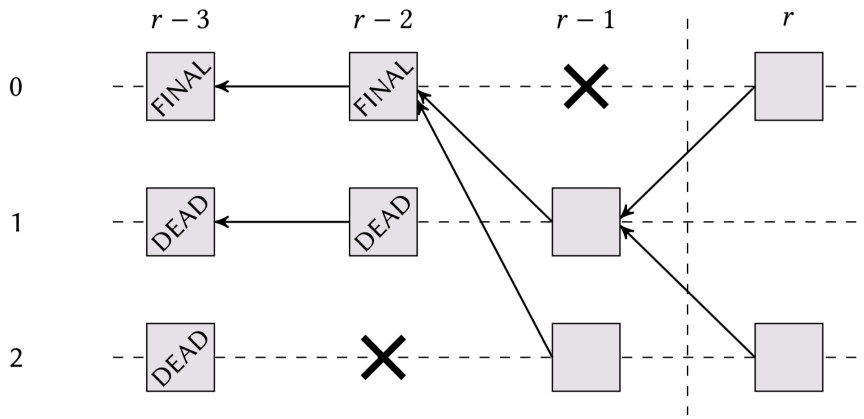

图 8: Dfinity 区块最终确认。第 r 轮加 T 时间执行第 $r-1$ 轮的最终确认。

被最终确认。

8.3 门限签名

关于具体原理涉及到密码学的诸多知识，这里不多做介绍。这里说明的是，t, n 门限签名实现的主要功能为，一旦获得了 n 个人中 t 个人的签名 $\sigma_{i_1}, ..., \sigma_{i_t}$，则可以算出所需的组签名 σ。该组签名可以被任何人验证（组公钥为公开信息），且所有人的私钥仍没有被透露。其工作原理大致是用的是 $t-1$ 次多项式函数可以被 t 个点唯一确定。

Dfinity 随机种子层的工作原理就是用的门限签名。在第 r 轮开始是，每个委员会成员计算签名

$$\sigma_{r,i} = Sign(r||\xi_{r-1}, sk_{G,i})$$

其中 $sk_{G,i}$ 为成员 i 在委员会 G 中的私钥。

随后，根据 t 个签名 $\sigma_{r,i}$ 计算出组签名 σ_r，将其哈希值作为第 r 轮的输出 ξ_r。

该签名方案简单实用，事实上被绝大部分区块链项目所使用。

Dfinity 最后还提到了朝代（epoch）的概念，朝代更迭是开展对客户端/委员会候补集合的申请与注销工作。

8.4 总结与思考

关于 VRF 的使用，不可预测性和可验证性都是必须要满足的条件。其中不可预测性就意味着随机种子不能是固定值。所以，随机种子的选取非常重要，在没有全局时钟的情况下，由前一个区块的数据及当前区块高度共同决定是可行的方案。Dfinity

对 VRF 的应用和投票思想都很清晰严谨，具有很高的借鉴意义。事实上，TAS chain 白皮书[3]大部分就是借鉴 Dfinity。不足之处在于假设过多，真正的安全性和性能存疑，而且准入门槛对于 permissionless 的公链也是个难以解决的问题。

9 Algorand

Algorand 由 Micali 等人提出，发表在 2017 年的 SOSP（OSDI 的奇数年）上 [28]。Algorand 提出支持大量节点，基于用户所持财富投票以及 VRF 的区块链共识算法。

9.1 模型与假设

Algorand 中所有用户根据其拥有的财富比例对应一个权重（weight），同时假设至少 2/3 的财富被诚实用户所拥有。

Algorand 核心叫 BA★ 算法，用于实现根据用户权重投票达成共识：可发起投票的委员会成员根据用户权重随机选取。由于权重与财富挂钩自动防止了女巫攻击。同时，为了防止委员会被定点攻击（包括贿赂，进攻 IP 地址，伪造身份等等），BA★ 能保证任何委员会成员只能发出一次信息（指发起新信息源；之后每次广播会重新生成委员会成员，且只能对现有投票结果进行再次投票），一旦信息发出后该成员和 BA★ 没有关系。另外，文章指出 BA★ 能对大量节点实现共识要求，而不像传统的拜占庭容错算法，对参与成员的总数有一定限制。

网络环境假设

Algorand 分了两种网络环境：

强同步假设 要求大部分 (95%) 诚实用户信息都能被送达，在此假设下 Algorand 能满足活性，即，任何诚实交易都会被上链，且确认时间为一分钟左右。

弱同步假设 允许系统在一段时间内处于异步状态，但在之后一定会存在一段较长的同步网络状态时间用以恢复安全性。在此假设下，Algorand 能满足安全性，即所有诚实节点能以高概率达成一致。

这正是所谓“同步活性，异步一致性”的概念。值得一提的是，增加等待出块的时间能够放宽同步性假设，即可以允许更长的异步时间。但代价是用户可以有足够的时间将钱转到另外一个账户进行抵押投票，引发所谓的无利害关系问题。Algorand 提出的解决方法为将用户权重定义为账户当前财产和抵押到区块财产（balance from the look-back block，待定）的最小值。

[3]https://www.taschain.io/static/pdf/TASChainTechnologyWhitePaper.pdf

另外，Algorand 还假设（弱）同步时钟的存在，即，每个诚实节点的本地时钟大致相近。文章指出可以用 NTP(Netword Time Protocal) 实现。

9.2 算法介绍

Algorand 给每个用户一个权重 w_i，该权重同用户财富正相关。定义 $W = \sum_i w_i$，用户权重总和。同时每个用户拥有一对公私钥 (pk_i, sk_i)。

同时，Algorand 还用到了传统的 VRF，即，可验证随机函数。在 Algorand 里，$VRF_{sk}(x)$ 实际上是一个返回值足够随机[4]的对 x 的签名，其验证可以通过 pk 得到。该函数的实现在 VRF 原始论文里即有介绍 [29]，其具体实现为对 RSA 加密稍作调整，使得签名的值域足够随机。

BA★ 核心算法主要包含以下几部分：

- $Sortition()$

  该函数的目标是根据用户权重决定用户能投的票数。假设用户权重为 w，随机种子为 $seed$，τ 为期望委员会总票数，$role$ 包含当前轮数 $round$，当前投票阶段 $step$ 等信息。定义 $p = \frac{\tau}{W}$。该函数简略流程如下

  - $hash = VRF_{SK}(seed||role)$，用于生成绑定该用户的可验证随机数。
  - 返回唯一的 j 使得

$$\frac{hash}{2^{hashlen}} \in [\sum_{k=0}^{j} B(k; w, p), \sum_{k=0}^{j+1} B(k; w, p)),$$

    即为该用户的票数

  其中，$B(k; w, p)$ 为，在 w 个人中每个人以 p 的概率被选取，最终恰有 p 个人被选出的概率。其表达的意思为，一个权重为 w 用户可以看做 w 个票数为 1 的子用户。然后，委员会共需要选出 τ 个子用户用于投票，故每个子用户备选出的概率为 p。而 $\frac{hash}{2^{hashlen}}$ 相当于一个 0 至 1 的随机数，故上模拟了此次选取恰有 j 个子用户被选取的过程，即该用户拥有 j 票。

- $VerifySort()$

  该函数用于验证一个用户的投票权利合法。通过验证 $hash$（由 $Sortition()$ 给出）符合用户公钥，然后返回该用户的票数。

[4]准确来说，是指任何一个 adversary 都无法在多项式时间内以大于（1/2+ 不可忽略误差）的准确率区分出某个给定结果来源于消息的签名还是一个纯随机数。

- **随机种子的选取**

  第 r 轮的随机种子由第 $r-1$ 轮出块者的 u 签名决定 $seed_r = VRF_{sk_u}(seed_{r-1}||r)$。若当前轮出的是空快，则根据哈希值决定 $seed_r = H(seed_{r-1}||r)$。

- **优先级的决定**

  对于用户 i 对应的 j 个子用户，以"哈希 [区块哈希值 || 子用户编号]"作为子用户优先级，取优先级最大的子用户作为区块（用户 i 提出的）的优先级。Algorand 用户会丢弃所有低于当前接受到的最高优先级的信息。

- $CommiteeVote()$

  每个用户执行该函数时完成两个过程。

  1. 确定该用户是否有投票权。通过调用 $Sortition()$ 实现。如果返回的票数不为 0 则拥有投票权。

  2. 如果拥有投票权，通过 Gossip 协议广播投票目标及可验证的票数。投票目标为传到 $commitvote()$ 的变量。其中 Gossip 是一个文件转发协议。注意到 BA★ 过程只有第一次调用可能提出自己的区块，之后均只能广播指定的投票结果。

- $CountVote()$

  运行该函数时，用户会用一段时间接收投票信息。每收到一条投票信息 m，进行信息处理（文章中为 $ProcessMsg()$ 函数）包括通过执行 $VerifySort$ 来验证投票合法性，获取投票目标区块（value）及票数（votes）。同时，将投票目标 value 的投票计数器进行累加 $counts[value] += votes$。

  用户持续接受信息并作上述处理，直到下面两种情况之一发生则终止:

  - 若存在 $count[value] > T \cdot \tau$，则返回 $value$。其中 $\tau > 1$ 为算法相关参数，$T > 2/3$ 为门限比例。
  - 若一段时间内（定值）没有收到新的信息，则返回 TIMEOUT。

  该算法保证如果某个诚实用户返回某个区块，则所有诚实用户或者返回相同区块，或者返回 TIMEOUT。

- $Reduction()$

  该算法的目标为将需要达成共识的多个区块转化为对出某一特定区块或出空块二选一达成共识。

  该函数有两次投票与统计过程，

- 第一次，用户针对传入 $Reduction()$ 的参数区块（首次调用时即用户自己的提出的区块）调用 $Commiteevote()$ 函数，即，发起对原始区块的投票。随后调用 $Countvote()$ 进行票数统计，返回结果为 $hblock_1$。
- 如果 $hblock_1$ 不为 TIMEOUT（即，存在某一个区块的票数大于阈值 $T \cdot \tau$），则针对该区块再发起一轮 $Commiteevote()$，否则，针对空块再发起一轮 $Commiteevote()$。随后再次调用 $Countvote()$ 进行统计，返回结果为 $hblock_2$。如果 $hblock_2$ 不为 TIMEOUT，将其作为 $Reduction()$ 的返回值，否则，将空区块作为 $Reduction()$ 的返回值。

由于阈值的存在，该算法保证若 $Reduction()$ 返回的不是空块则所有用户返回的值相同。

- $BinaryBA\bigstar$

  该算法用于在某个特定区块和空块之间达成共识。持续 $Maxstep$ 轮（若期间没有达成共识则认为网络状况出现问题）。每一轮执行大致描述如下（我们用发起投票和统计票数来表示 $Comiteevote$ 和 $countvote$：

  第一步，先对传入的给定参数区块发起投票并统计票数，若对参数区块达成共识则返回该区块。同时，如果在第一轮就对该区块达成了共识则将该区块标记为 $Final$ 状态（其余时候则为 $Tentative$ 状态）。令 r 为统计票数的返回值（没达成共识且没超时只能是空块）。如果返回 TIMEOUT 则令 $r = block\_hash$（传入的参数区块）。

  第二步，针对 r 再发起一轮投票和统计票数，将 r 更新为返回值。若对空区块达成共识则返回空区块。若超时（返回 TIMEOUT）令 r 等于空区块。

  第三部，针对 r 再发起一轮投票和统计票数，将 r 更新为返回值。若返回 TIMEOUT，则通过一个掷硬币函数来决定 r 被赋值为空块还是 $block\_hash$，同时进入下一轮。

- $BA\bigstar$

  每个用户执行的共识的主体算法 $BA\bigstar$ 为，先对传入的参数区块（可以认为是每个用户自己提出的）调用 $Ruduction$，再对其返回值执行 $BinaryBA\bigstar$。最后的结果即为共识算法的输出。

9.3 总结与思考

Algorand 的主要优势在于把财富作为影响随机数的因素考虑了进去，并且实现了最终一致性。

$BA\bigstar$ 算法仍存在多轮的信息传播，不能保证速度。但其优势在于区块提出者一旦作为委员会成员第一次发起了区块信息后，之后虽有投票但完全可以被其他用户所代替，防止了被定点攻击的可能性。本质上 Algorand 并没有实现委员会成员的匿名性，而这是我们认为防止类似攻击最为稳妥的算法。

10 Ouroboros

TBA

参考文献

[1] W. Wang, D. T. Hoang, P. Hu, Z. Xiong, D. Niyato, P. Wang, Y. Wen, and D. I. Kim, "A survey on consensus mechanisms and mining strategy management in blockchain networks," IEEE Access, 2019.

[2] 袁勇, 倪晓春, 曾帅, and 王飞跃, "区块链共识算法的发展现状与展望," 自动化学报, vol. 44, no. 11, pp. 2011–2022, 2018.

[3] E. Eisenberg and D. Gale, "Consensus of subjective probabilities: The pari-mutuel method," The Annals of Mathematical Statistics, vol. 30, no. 1, pp. 165–168, 1959.

[4] S. Nakamoto et al., "Bitcoin: A peer-to-peer electronic cash system," 2008.

[5] Y. Sompolinsky and A. Zohar, "Secure high-rate transaction processing in bitcoin," in International Conference on Financial Cryptography and Data Security, pp. 507–527, Springer, 2015.

[6] I. Eyal, A. E. Gencer, E. G. Sirer, and R. Van Renesse, "Bitcoin-ng: A scalable blockchain protocol," in 13th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 16), pp. 45–59, 2016.

[7] I. Eyal and E. G. Sirer, "Majority is not enough: Bitcoin mining is vulnerable," Communications of the ACM, vol. 61, no. 7, pp. 95–102, 2018.

[8] A. Poelstra, "Mimblewimble," 2016.

[9] L. Lamport, R. Shostak, and M. Pease, "The byzantine generals problem," ACM Transactions on Programming Languages and Systems (TOPLAS), vol. 4, no. 3, pp. 382–401, 1982.

[10] maxdeath, "Bft." `https://zhuanlan.zhihu.com/p/41329283`, 2005.

[11] L. Lamport, "Constructing digital signatures from a one-way function," tech. rep., Technical Report CSL-98, SRI International Palo Alto, 1979.

[12] M. J. Fischer, N. A. Lynch, and M. S. Paterson, "Impossibility of distributed consensus with one faulty process.," tech. rep., MASSACHUSETTS INST OF TECH CAMBRIDGE LAB FOR COMPUTER SCIENCE, 1982.

[13] R. Wattenhofer, The science of the blockchain. CreateSpace Independent Publishing Platform, 2016.

[14] M. Ben-Or, "Another advantage of free choice (extended abstract): Completely asynchronous agreement protocols," in Proceedings of the second annual ACM symposium on Principles of distributed computing, pp. 27–30, ACM, 1983.

[15] G. Bracha, "Asynchronous byzantine agreement protocols," Information and Computation, vol. 75, no. 2, pp. 130–143, 1987.

[16] L. Lamport et al., "Paxos made simple," ACM Sigact News, vol. 32, no. 4, pp. 18–25, 2001.

[17] M. Castro, B. Liskov, et al., "Practical byzantine fault tolerance," in OSDI, vol. 99, pp. 173–186, 1999.

[18] R. Kotla, L. Alvisi, M. Dahlin, A. Clement, and E. Wong, "Zyzzyva: speculative byzantine fault tolerance," in ACM SIGOPS Operating Systems Review, vol. 41, pp. 45–58, ACM, 2007.

[19] A. Clement, E. L. Wong, L. Alvisi, M. Dahlin, and M. Marchetti, "Making byzantine fault tolerant systems tolerate byzantine faults.," in NSDI, vol. 9, pp. 153–168, 2009.

[20] A. Kiayias and A. Russell, "Ouroboros-bft: A simple byzantine fault tolerant consensus protocol," 2018.

[21] Y. Aumann and Y. Lindell, "Security against covert adversaries: Efficient protocols for realistic adversaries," in Theory of Cryptography Conference, pp. 137–156, Springer, 2007.

[22] L. Lamport, "Time, clocks, and the ordering of events in a distributed system," Communications of the ACM, vol. 21, no. 7, pp. 558–565, 1978.

[23] M. Burrows, "The chubby lock service for loosely-coupled distributed systems," in Proceedings of the 7th symposium on Operating systems design and implementation, pp. 335–350, USENIX Association, 2006.

[24] J. C. Corbett, J. Dean, M. Epstein, A. Fikes, C. Frost, J. J. Furman, S. Ghemawat, A. Gubarev, C. Heiser, P. Hochschild, et al., "Spanner: Google's globally distributed database," ACM Transactions on Computer Systems (TOCS), vol. 31, no. 3, p. 8, 2013.

[25] G. Golan Gueta, I. Abraham, S. Grossman, D. Malkhi, B. Pinkas, M. K. Reiter, D.-A. Seredinschi, O. Tamir, and A. Tomescu, "Sbft: a scalable decentralized trust infrastructure for blockchains," arXiv preprint arXiv:1804.01626, 2018.

[26] V. Buterin and V. Griffith, "Casper the friendly finality gadget," arXiv preprint arXiv:1710.09437, 2017.

[27] D. E. Knuth, The art of computer programming, vol. 3. Pearson Education, 1997.

[28] Y. Gilad, R. Hemo, S. Micali, G. Vlachos, and N. Zeldovich, "Algorand: Scaling byzantine agreements for cryptocurrencies," in Proceedings of the 26th Symposium on Operating Systems Principles, pp. 51–68, ACM, 2017.

[29] S. Micali, M. Rabin, and S. Vadhan, "Verifiable random functions," in 40th Annual Symposium on Foundations of Computer Science (Cat. No. 99CB37039), pp. 120–130, IEEE, 1999.