Merkle RO 黄皮书

计算、存储、传输全维度分片实现区块链完全扩展性

2018 年 6 月

版本 1.0

区块链技术带来的“去中心化”理念正在被用于越来越多的场景。作为区块链技术 的起源，比特币已经证实了去中心化对于数字资产的非凡意义；更进一步的，以太坊 证实了去中心化对于分布式应用的重要；越来越多的区块链项目正在探索去中心化 这一理念在更多场景及应用下的价值。

不难发现，“去中心化”理念的背后，是区块链系统中的开放性(Openness)与匿名性并存。

一个区块链价值衡量标准需要具备三个特点：

•真实性 一个好的价值衡量标准应该能够准确反映出区块链经济系统的特征，这 样才能在相应的领域具有足够的公信力；

•公平性 价值衡量标准为相应的激励提供了依据，因此，这一依据必须足够公平，才能防止作弊或操纵带来的“劣币驱逐良币”现象；

•多样性 需要使用数据及数字资产价值的场景可能是多种多样的，其使用方式及 对应的激励方式不尽相同，因此相应的价值衡量标准既不能脱离应用场景，亦 要满足前述的真实性及公平性。

区块链的扩展性是阻碍区块链技术大规模应用的主要瓶颈，而分片在并行化执行上的出色表现使得它成为解决这一问题的最佳技术方向。然而，目前的区块链分片方案尚未实现完全可扩展性。Merkle RO 将在本文中阐述全球首个能够实现完全可扩展性的快速、高效、全维度的区块链分片架构，这也是全球首个在计算、存储和传输三个维度都实现了完全分片的区块链技术。

Merkle RO 设计了一个精妙的基于 Merkle Root 的分布式存储和传输方案，使用极其精简的数据就能对大数据集进行高效和安全地校验、增量修改和跨片通讯，减轻了矿工的存储负担和传输负载，大幅度降低了账本规模和网络传输量，从而实现了存储、传输、计算的分片。

Merkle RO 使用 PoS 机制抵御 Sybil 攻击，使用可验证随机函数(VRF)将区块链网络划分为若干“分片”并将矿工动态分配给分片，分片内部通过拜占庭共识族处理 Transaction 事务；网络可随负载变化进行分片分裂增长。这个全维度分片方案，既使得区块链分片的并行化处理成为现实，实现了网络性能的线性增长和无限拓展，又降低了矿工参与门槛，维护了区块链去中心化的原始价值主张。

在 2018 年 10 月进行的实验室测试中，Merkle RO 在由 64 个分片、12800 个节点组成的网络中达到了 30784 TPS，单分片处理能力超过 500 IPS;随着节点数量和分片的增加，全网性能表现出优异的线性增长；采用不同配置的计算设备进行测试，全网性能结果所差无几，也表明 Merkle RO 可有效降低矿工门槛，个人电脑等普通计算设备也能够自由加入网络，维护了区块链系统的开放性和公平性。

Merkle RO 的全维度分片将极大提升了区块链的容量和扩展性，达到足以支持实体经济的产业级性能要求，同时维护了区块链系统的去中心化、不可篡改和安全性等所有核心价值。

1.介绍

自 2008 年中本聪（Satoshi Nakamoto)发表比特币白皮书[1]以来，区块链技术席卷全球。最受关注的比特币和以太坊[2]，已经被用于跨境结算、供应链管理、娱乐和投资等诸多领域。
然而，区块链技术目前面临着严重的瓶颈，每秒处理事务（TPS)能力低下，运行全节点负担沉重，限制了其在现实世界的发展和应用。比特币吞吐量只有每秒 4-5 笔 Transaction（tps)，以太坊则是每秒 10 笔 Transaction，Transaction 积压和网络拥堵频繁发生。而在现实世界中，占据领先地位的 VISA 支付系统，每秒可以轻松处理 2000 笔 Transaction，最高可处理 4.5 万笔 Transaction。目前区块链系统的吞吐能力远远无法满足实体经济所需要的百万量级的 Transaction 需求，所有区块链从业者都很清楚，是否能够实现突破当前技术的速度、容量和可扩展性，将决定区块链未来能否在现代社会和商业中广泛应用。
Merkle RO 相信，在未来十年中，区块链将被广泛应用于现在可想象的任何类型的 Transaction 中。为了实现这种可扩展性，区块链系统需要克服众多可扩展性的挑战。这将是一场技术创新之战，Merkle RO 将在最前沿且最中心的位置参与其中。

1.1 区块链扩容方案概述

目前学术界和产业界已经提出了若干提升区块链可扩展性的方案，本文大致总结为三种：超级节点，链外扩展，链内扩展。

- 超级节点可看做是一种部分中心化的解决方案，以 EOS 为代表，通过由少量高性能节点处理区块链系统中的大部分共识流程来提高性能。以 EOS [3]， IOST [4]和 Quarkchain 的根链系统[5]为代表。传统区块链系统中，网络的处理能力受到单个节点处理能力的限制，而超级节点方案将共识权利完全交给了具备超高性能的少数节点，从而提升了 Transaction 速度和总吞吐量。但这种方案却淘汰了普通用户，更重要的是它创建的半中心化系统也失去了区块链“去中心化”的原始主张和核心价值。
- 链外扩展主要包含侧链和状态通道两种伸缩方案，都是将大多数事务在主链以外进行处理，从而绕过主链系统的性能瓶颈。侧链方案的代表有 Cosmos 和 Aelf，通过侧链与主链链接的方式处理 Transaction，吞吐量可增加数个数量级；但这种方式也存在更多安全风险和性能隐患，例如跨链之间的信任问题、主链汇总负担过重依然会存在瓶颈等。状态通道的代表如 Plasma 和闪电网络，提供了基于主网之外的用户之间进行 Transaction 的专属事务通道，从而实现扩展性。此类方案的共同挑战在于如何构建足够安全的链上和链下信任机制，重新引入了区块链系统在创建之初想要避免的中心化失败问题。
- 链上扩展目前以分片技术（Sharding)和有向无环图（DAG)为主，希望通过改变网络中节点的结构来实现区块链的可扩展性。DAG 的代表方案包括 Iota、Hash Graph、Vite、Conflux 等，基于有向无环图的数据结构允许异步产生区块。DAG 方案能够在实验室环境中运转迅速，但阻碍它现实应用的主要挑战在于此类方案的 Transaction 广播量极大，在现实环境中会导致网络风暴问题。分片技术是分布式数据库中常用的扩展性方案，能够显著提高吞吐量，在现实环境中也能够稳定运行，例如 ZUliqa 在其测试网上稳定运行超过 2000 TPS[14]。然而，目前已知的分片方案大多数只将矿工进行了分片，单个节点仍然需要承担网络的全部存储和传输负担，

这并没有从根本上突破区块链系统的瓶颈。

1.2 Merkle RO 设计原则

Merkle RO 要解决的问题非常明确：“区块链系统需要实现哪些技术特性，才能够真正达到产业级的可扩展性来服务实体经济”。以下几个原则是显而易见的：

首先，一个理想的可扩展的区块链架构，应该以“去中心化”为核心，因为去中心化是我们创造并开始使用区块链的最重要原因。区块链的设计目标是通过共识算法提供去中心化的计算平台，好处是避免了中心节点的故障和控制，成本是需要一个达成共识的一致性算法来确定结果。我们认为，后退成一个半中心化的系统，不是在朝着一个足以颠覆全球商业结构的强大系统的方向前进。其次，理想的可扩展架构应该包含对区块链底层协议的升级改进。链外方法并不理想，他们更像是补丁，能够有所改进但往往又会引入新的安全风险。因此，Merkle RO 选择了目前为止最可靠的扩容方案一分片，自上而下地改进了区块链的底层协议和整体架构。

分片就是将区块链的共识过程并行化。和当今大规模数据处理往往有多台服务器并行执行的方式类似，并行化是区块链能够处理大型商业应用的计算需求的关键。分片是将区块链系统扩展到产业级容量，同时保持区块链系统的开放性和去中心化属性，唯一可行的链上和底层解决方案。正是因为如此，包括以太坊在内的最知名的公链们，都在致力于在原有系统上进行分片改造。然而在最初的非分片系统上设计分片挑战重重，Merkle RO 的优势也包括，我们从始至终都是一个完整且彻底的分片系统。

Merkle RO 的设计方案不同于现在几乎所有的分片方案，并在它们的基础上有所改进。我们的设计核心是基于一个非常重要的理解：计算机和计算网络的功能不仅仅是完成计算，他们还有大量的工作是在传输和存储。因此，Merkle RO 的核心就是全维度分片：不仅可以实现计算分片，而且在数据的存储和传输层面也可实现分片。Merkle RO 是首个为全维度而设计的区通过达成共识产生新的账本来获取出块奖励。在块链分片系统，完全分片的底层架构也使得它真正可以稳健地处理产业级需求。

对 Merkle RO 的核心技术优势总结如下：

1.仅实现 Transaction 或计算分片，不足以解决区块链系统的可扩展性问题，Merkle RO 是第一个提供三维分片机制的公链，不仅将分片用于 Transaction 和计算，还用于存储和传输。

2.公平性、可靠性、安全性对区块链系统都至关重要，Merkle RO 采用了基于可验证随机函数（VRF)的公平分片技术，确保每个分片的可靠性和安全性。

3.跨片通信是分片系统面临的重要难题之一，Merkle RO 设计了一个简洁的 UTXO 账本结构和基于 Merkle Root 的分布式数据结构，能够实现在分片数据环境下，进行安全的异步跨片通信和可信的第三方数据存储和验证。

2.Merkle RO 协议概览

首先，本文将对 Merkle RO 的计算、存储和传输分片进行一个概述，具体细节将在下一章开始详述。在这篇文章中我们会广泛使用图表，为了便于读者理解，在这里我们先给出图表中使用的图例：我们在图表中使用如下图例：

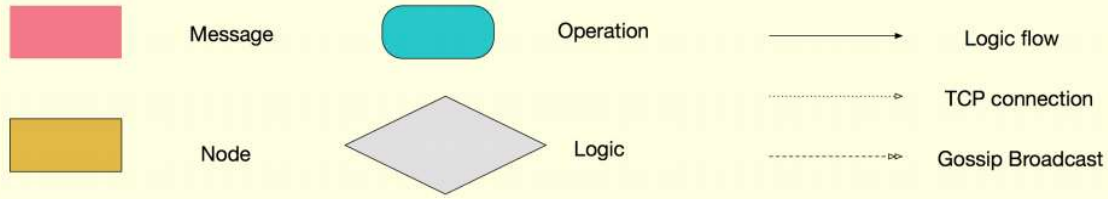


图 1:图 2 和图 3 的图例

2.1 基本概念定义

节点

节点是连接到网络的不同类型的机器。Merkle RO 有三种类型的节点：轻节点、矿工节点和存储节点。轻节点，或称为客户端，是只提交新的 Transaction 而不做任何处理的节点，等同于拥有账户并进行 Transaction 的用户。矿工节点是运行共识算法和更新账本的节点，它们以分片为单位聚集。每隔几分钟分片就会重新随机选取属于它的矿工节点。矿工节点是系统中的记账员，并

Merkle RO 网络中，成为矿工节点所需的硬件门槛较低，普通 PC 也能够成为矿工节点参与共识。存储节点是负责存储和服务 Transaction 的特定节点，每个分片会配有一定量的存储节点。它们的功能类似于网络中的基础设施，给矿工提供服务，使矿工能够更快、更高效地执行 Transaction 共识。以上三种类型的节点组成了 Merkle RO 网络。

可验证随机函数

Merkle RO 使用可验证随机函数（Verifiable Random Functions，VRF)将矿工节点分配到不同的分片，并进一步从单个分片中选出一小部分矿工来完成记账工作。VRF 是一个用于生成可验证随机数的随机数生成器，一个 VRF 函数能够在无信任网络中作为节点的随机数生成器，它在产生随机数的同时还能够生成一个证明，其他节点只要获取了这个证明，就可以验证某节点产生的随机数是否是合法的，从而保证节点无法擅自操纵随机数生成。VRF 还有另外两个特点，一是不可预测性，即它产生的伪随机数在被生成之前是无法被预测的；二是时间无关性，即在不同时间产生的随机数与当前时间无关。Merkle RO 使用 VRF 将矿工随机分配到不同的分片，同时要求其他矿工在继续执行 Transaction 之前验证节点分配的合法性。

节点设置所有节点都通过一个异步网络连接，这使得几乎所有的矿工都可以在一个较短的时间内完成通信。每个节点都有一个加密的公钥和私钥对，私钥只由节点掌握，公钥是公开的。节点可以使用自己的私钥对某个信息进行签名，而其他所有节点都可以使用它的公钥验证并确定该信息的内容和来源的真实性。

2.2 动态的重分片机制

Merkle RO 使用动态分片机制，分片的矿工节点每隔几分钟就会动态改变一次。当某个分片进行矿工的重分配时，新的矿工节点可以在此时加入它，一个矿工节点还可以同时加入多个分片。整个动态重分片的过程如图 2 所示：

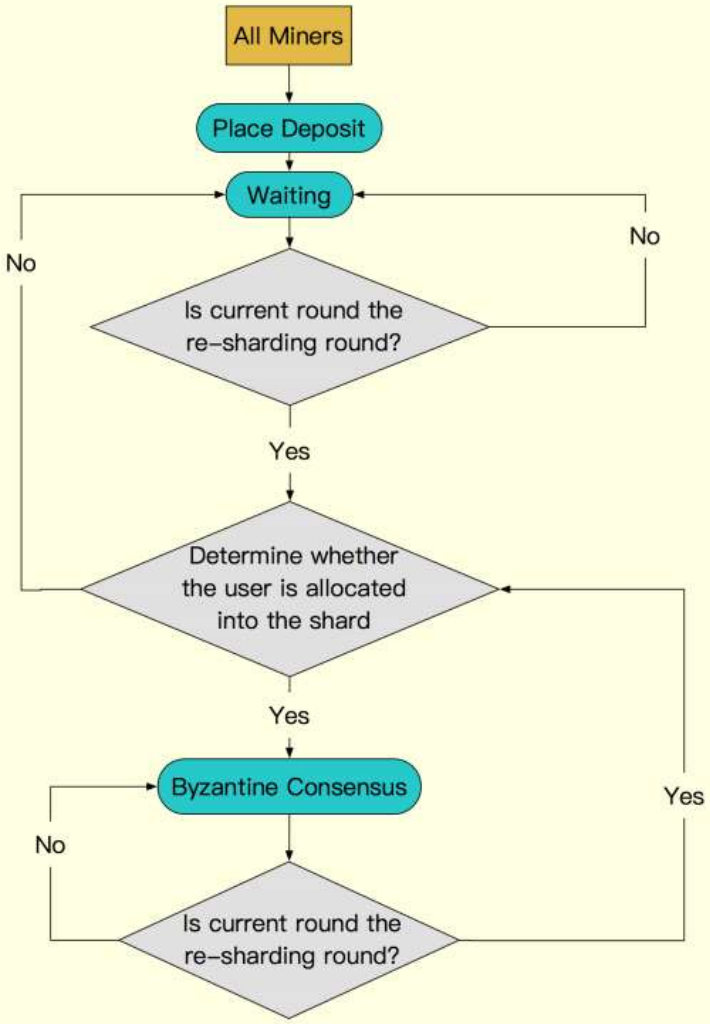


图 2:矿工的动态分片流程图

Merkle RO 使用 PoS 准入机制，矿工必须证明其拥有一定量的代币（股权）的所有权才能参与共识。有关为何我们使用 PoS 而不是 PoW，请参阅附录 1.在 Merkle RO 中，我们要求每个矿工在加入分片之前需要锁定一定数量的押金来强制执行 PoS 机制。

质押押金后，矿工需要等待约几分钟的时间，直到有一个分片开始进行重分片时，将该矿工节点分配到自己下一阶段的矿工集合中。押金越高，被某个分片选中的概率就越高。Merkle RO 以异步方式运行分片，因此矿工可以通过增加押金来提高自己被多个分片选中的概率，从而可以在多个分片中工作并赚取奖励。当某个分片进行重分片时，矿工会使用 VRF 用自己的私钥和一个种子产生伪随机数，从而判断自己是否被选入了该分片。如果没有被分片选中，矿工需要继续等待，直到有下一个分片开始进行重分片。

当某个分片过载时，Merkle RO 还会将原有分片进行再分裂，分片分裂（Shard Splitting），以增加分片数，满足 Transaction 性能的需求。触发分片分裂时，超负载的原分片将被分为两个分片，属于原分片的轻节点将片和旧分片中选择一个继续服务，最后再通过重分片将矿工分配给新分片。

2.3Transaction 确认、共识和存储过程

在 Merkle RO 中，存储节点负责存储所有 Transaction 信息。Merkle RO 使用的是比特币所使用的 UTXO 模型，即每个 Transaction 的输入（Input)都必须是之前被确认的某个或某几个 Transaction 的输出（Out)。网络会追踪每一个输出并记录其状态（即该 Transaction 输出是否已经被花费掉），以避免双重花费。Merkle RO 将每一个 Out 及其状态存储在一个基于默克尔树（Merkle Tree)的数据结构中。

轻节点生成并签名一个新 Transaction 后，会将这个 Transaction 提交给其所属分片的存储节点，存储节点将 Transaction 广播给分片中的所有矿工。在动态重分片机制下，这些矿工会不停切换。（存储节点将在该 Transaction 被写入区块后获得奖励。）

在分片内部，节点们通过允许一个快速的拜占庭共识算法来达成共识。达成共识后，区块被广播到该分片中的存储节点，并由存储节点将区块保存在自己的硬盘中。相应的区块头被广播给全网的所有矿工。能够明显提高空间存储效率的环节在于，在该分片之外的矿工，仅仅需要保留区块头信息，而不需要存储整个区块的具体 Transaction 的全部信息。至此，该笔 Transaction 完成被确认并被写入账本中。整个过程如图 3 所示：

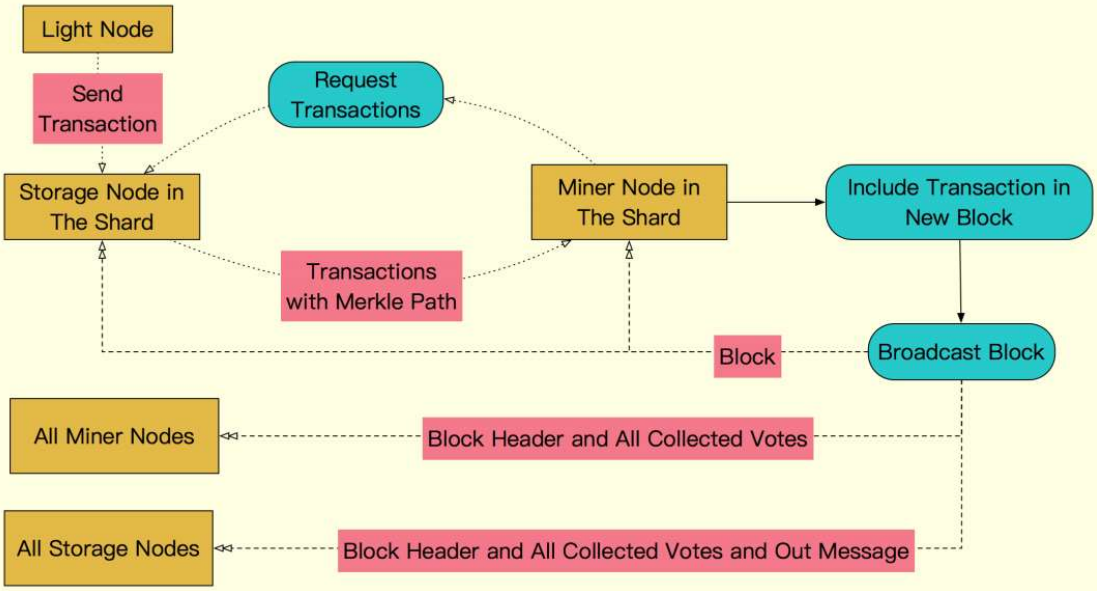


图 3 区块产生的流程图示意

依据其地址分配给两个新分片，存储节点可以在新分

2.4 详细术语定义

现在我们详细介绍 Merkle RO 系统中所使用的术语，便于读者理解和阅读这篇文章。

2.4.1Transaction 术语

- **TransactionTx**：指将一定量代币从一个账户转移到另一个账户的消息。每个 Transaction

都包括一组输入、一组输出和用来代表这笔 Transaction 的 Transaction 类型的 id。

- **输入 Input**:输入是先前某笔 Transaction 的输出，包含了对之前某一笔 Transaction 的输出的引用，和一个用来验证所有者的脚本签名。
- **输出 Output**：输出是某个 Transaction 产生的数据，包含了尚未被花费的代币的数量，以及一个与接受者地址对应的脚本，只有能够提供对应私钥的用户才能解锁并使用该输出。
- **输出状态 Output State**：是一个二进制（0-1)标志，用于确定输出是否已经被用作某笔 Transaction 的输入。
- **Transaction 类型 Transaction Type**：Transaction 类型是一个标记，包含三种类型
  - **N**，正常 Transaction：从一个账户到另一个账户的普通转账 Transaction。
  - **D**，押金 Transaction：用于提交/质押押金的 Transaction，付款人和收款人必须是同一个账户，押金 Transaction 中的代币只能用于取款 Transaction。
  - **W**，取款 Transaction：提取押金并将其解锁的 Transaction。
- **输出确认消息 Output Message**：是指从一个分片的存储节点提交给另一个分片的新成交的转账 Out 消息，它允许接收分片的存储节点重建一个仅仅包括了从第一个分片转往第二个分片的转账 Out 信息的 Block Merkle Tree。

2.4.2 节点和客户端术语

- **矿工节点（Miner Node ）**：或称为用户(User)，矿工节点负责执行共识过程，并在正式出块时获得奖励。任何运营矿工软件的用户都可以在锁定一部分资金作为押金后获得成为矿工的资格。矿工需要监听自己分片的所有消息和来自全网的区块头信息、Merkle Root 和部分 Merkle Tree 信息，以确认待处理 Transaction 的有效性。矿工节点还会自己维护一个由尚未确认的 Transaction 组成的 Transaction 池，为之后的区块提议聚合潜在的 Transaction。M 表示所有矿工节点的集合。
- **存储节点（Storage Node)**：存储节点会被分配给一个分片，为该分片服务，负责存储所有在该分片上产生的历史 Transaction。存储节点的职责是响应和服务来自矿工节点的数据请求，也通过提供存储和数据来得到奖励。它们的硬件要求比矿工节点更高。但值得注意的是，存储节点只是网络服务的提供者，不参与共识，对数据没有任何增删篡改能力，因此在网络中完全没有决策“权力”，它们并不会造成中心化风险，与 Merkle RO 的去中心化宗旨一以贯之。S 表示所有存储节点的集合。
- **轻节点（Light Node)**：或称为客户端，是指提交新 Transaction 的节点。它们不需要存储任何区块信息，只需要存储自己的 Transaction 记录。轻节点的硬件要求最低，一台普通的手机就可以成为一个客户端。i 表示所有轻节点的集合。

2.4.3 数据存储术语

- **Merkle Tree**：是一种用于数据存储的二叉树形状的数据结构，其中每个叶节点代表一个数据块，直到根节点的每个非叶节点都是其子节点的加密散列。这样的哈希层次结构能够及其快速和安全地验证某个数据是否真实地存在于 Merkle Tree 中，例如验证一定代币数量的 Transaction 数据。
- **Merkle Root**：是 Merkle Tree 的哈希根（头部节点）。具有 Merkle Root 的用户可以检查内容的 Merkle Path 来快速验证某个数据是否真实存在于 Merkle Tree 中。
- **Merkle Path**：是某个叶子在 Merkle Tree 中到根节点的校验路径，用于安全验证叶子节点的内容。

- **Main Merkle Tree**：是存储节点中用来存储用来存储所有 Transaction 的 Merkle Tree 的名称。Main Merkle Tree 的叶子包含来自全网所有分片的历史输出。
- **Block Merkle Tree**：指每个生成的区块中的输出所组成的 Merkle Tree，存储着这个区块中记录的所有输出。每当一个区块被确认时，这个区块的 Block Merkle Tree 就会被添加到 Main Merkle Tree 中。
- **Top Main Merkle Tree**：是 Main Merkle Tree 的头部部分，不包含 Block Merkle Tree 的具体内容。

3. Merkle RO 分片机制

分片是 Merkle RO 设计的核心。类似于所有现代科技公司都在使用分布式计算以处理大量数据和业务， Merkle RO 将网络拆分为多个片区，从而能够并行执行和处理事务，这极大地增强了区块链系统对实体经济的大规模业务的响应能力。这些片区就被称为分片(shard)。作为 Merkle RO 区块链系统的基础组成部分，Merkle RO 的协议包含了自动的分片维护与运行。当某个分片内的 Transaction 流量过高时，Merkle RO 可以创造新的分片来分担过多的负载。新的分片会被分配入一些矿工节点和存储节点，这些节点可以立刻开始支持新分片并产生新的区块。

一个分片是 Merkle RO 网络的一个子集，它可以被写为

$$(C_{i,r}, (B_{i,h})_{1\le h\le r}, (H_{i,h})_{1\le h\le r}, MT_{i,r}, M_{i,r}, C_{i,r}, S_{i,r})$$

其中

- i是分片的序号；
- r 是当前分片内部区块链的高度；
- Ci,r是在当前分片上运行的共识算法；
- Bi,h是高度为 h 的区块；
- Hi,h 是高度为 h 的区块头部；
- MTi,r是当前的 Main Merkle Tree；
- $M_{i,r} \subseteq M, C_{i,r} \subseteq C, S_{i,r} \subseteq S$分别是当前分片在出高度为 r 的块的时候的矿工节点，轻节点和存储节点的集合。

每个分片会负责维护一条彼此独立的区块链。轻节点会根据其公匙地址被分配入不同的分片，Transaction 会根据接受者的地址被发往相应的分片，并由该分片负责处理并写入区块。每个分片会周期性地随机选取一批矿工以参与自己分片的拜占庭共识并产生新的区块。每个分片还会有一些存储节点专门负责存储该分片生成的区块，并负责更新该分片的 Out 的状态（是否被使用）。

3.1 矿工选择

Merkle RO 是一个基于分片的区块链系统，其中矿工作为分片的一部分负责执行共识算法。分

片内的矿工不应该是一成不变的，否则黑客攻击单个分片的难度会显著低于攻击整个网络，从而因为分片导致安全性能的退化。所以每个分片内的矿工会周期性随机变化。在这里我们遇到的一个核心问题是：分片是如何选择由哪些矿工来参与自己的共识和出块事务的。显然，我们应该让这个选择机制尽可能随机公平，不应该偏向于任何一个矿工。然后，一个单纯的随机选择机制是不足的：在缺乏一个足够强大公平的选择机制的情况下，恶意矿工可以发动女巫攻击，冒充多个用户，从而参与尽可能多的分片，最终达到控制共识结果的目的。一个经典的用于防止女巫攻击的选择机制是工作量证明(PoW)，它要求矿工持续进行计算来达成共识。然而，正如很多区块链文献中讨论的那样，工作量证明是低效并极为浪费的。

鉴于此，Merkle RO 使用了权益证明(PoS)。这是一种在有效避免能源浪费的前提下还能保证很高安全性的算法。每个矿工需要将他拥有的一部分代币锁定作为押金，这代表了它在 Merkle RO 中拥有的“股权”。矿工会根据在分片内产生的随机种子和他的密匙通过一个随机数生成器生成一个随机数，并根据这个随机数以及他的押金数量来决定他是否会被分配入某个分片内。分配入的概率和他的押金数成正比。值得一提的是，一个矿工可以同时加入多个分片。

矿工押金

Merkle RO 要求矿工在加入分片共识之前锁定一部分资金作为押金。矿工通过发送一个特殊的 Transaction（押金 Transaction）来锁定资金。当押金 Transaction 在他所属的分片内被确认并写入区块后，这个节点就可以作为矿工开始工作了。押金 Transaction 的输出是一个特殊的输出（押金 Out)，它必须通过一个取款 Transaction 来解锁这笔资金之后才能被花费。

一个押金 Transaction 包含了以下信息：

- 输入：一个或几个属于该用户的 Outs;
- 输出：一个转给用户自己的押金 Out;
- Transaction 类型：”D”，代表押金 Transaction

押金 Transaction 被确认后，它会被记录在区块头部信息中广播给全网，所有其他的节点就会得知该矿工已经锁定了一笔资金作为押金。之后该矿工会向全网广播一个心跳信息，表示他当前在线，从而有资格参与重分片时的矿工选择。矿工节点在发送了心跳后会开始监听所有分片的区块确认信息并等待，一直到某个分片开始进行重分片（该信息可以从区块头部信息中获取）。该矿工会生成随机数并判断自己是否已经被分配到该分片中，而其他矿工通过验证该矿工产生的随机数以及他的押金数来确认该矿工确实被选入了当前分片。

为了避免分片将一个长期不在线的矿工选入自己的分片，矿工的心跳信息有效时间为 24 小时。当心跳信息过期后，矿工需要重新发送一次心跳信息以确认自己在线。

取款 Transaction

押金 Transaction 输出的 Out 是被锁定的，它不能直接被作为另一个普通 Transaction 的输入来被花费。如果允许直接花费押金 Out，那么女巫攻击者可以通过以下步骤来低成本创建小号：

- 先以地址 addq 发送一个押金 Transaction。
- 当该 Transaction 被确认并且地址 addq 发送过一次心跳消息之后，将该押金 Transaction 转账给 addr2。
- 当该 Transaction 被确认并且地址 addr2 发送过一次心跳消息之后，将该押金 Transaction 转账给 addr3……

以此类推，在 24 小时的心跳消息激活期间，该用户可以使用多个地址锁定同一笔资金作为押金。因此，我们需要一笔特别的提款 Transaction 来解锁押金。

3.2 重分片

假设网络中的矿工总数是 W，c 是一个预设的诚实阈值，从而使得只要诚实的矿工数量队 2cW，我们就认为共识的结果是可被信任的。定义>cN)为在网络运行了一段时间 t 之后，在这期间诚实节点的数量总是大于 cW 的概率。传统区块链的安全性假设基于：当网络的总节点数 W 足够大，并且运行的时间 t 足够久之后，存在一个足够小的数 P，使得

$$P_t(N_h \geq cN) \geq 1 - p.$$

假设坤是分片;中矿工的总数，而 Wh,;是分片;中诚实矿工的数量。因为坤显著小于 W，我们可以得到

$$P_t(N_h \geq cN) \geq P_t(N_{h,i} \geq cN_i).$$

显然，$P_t(N_{h,i} \geq cN_i)$随着 t 的缩小而增长。为了保证分片不导致安全性的退化，我们需要限制

$t' < t$，从而使得

$$P_{t'}(N_{h,i} \geq cN_i) \geq 1 - p.$$

为此，我们每个 t'的时间（例如，每隔几分钟）就需要将矿工重新分配，以降低某个特定分片受到攻击的可能性。攻击者想要实现双重花费的前提是几分钟内攻陷某个分片内随机选出的数百上千个矿工的大多数，而这个可能通常来说是微乎其微的（通常来说攻击者很难在短时间内弄清分片内到底有哪些节点）。即使偶然有一个攻击者幸运地在很短的时间内攻陷了某个分片内的大多数矿工，只要经过几分钟，当这个分片进行重分片的时候，该攻击者就失去了对这个分片的控制。

可验证随机函数（VRF）

重分片机制基于可验证随机函数（VRF)。它是一个由 Micali, Rabin 和 Vadhan [16]介绍，并由 Dodis 和 Yampolskiy[17]进行了改进的伪随机数生成器。它 的优点是接收端在接收到生成的伪随机数之后，无需与生成伪随机数的节点进行进一步通信即可验证 VRF 的输出，从而使得 VRF 成为随机选择的理想工具。Dfinity[18]，Algorand[19]和 Ouroboros Paros [20]都将 VRF 作为它们的伪随机数生成器，并将其纳入为共识方案的一部分。Merkle RO 采用了[21]中描述的 VRF 结构。在这里我们将对该实现的 VRF 的安全性和伪随机性进行一点详细的分析。

设$a:N \to N \cup \{*\}$，$b:N \to N$以及$c:N \to N$是三个多项式函数。一个输入长度为$a(\lambda)$，输出长度为$b(\lambda)$和安全级别为$s(\lambda)$的 VRF 由三个多项式时间复杂度的算法(F_{GEN}, VRF, F_{VER})组成。其中

- F_{GEN}是公私匙生成函数。它是一个概率性函数，接受长度为 1 的一元字符串，并输出公匙 PK 和私匙 SK：

$$F_{GEN}(1^{\lambda}) = (PK, SK).$$

- VRF是主要的伪随机数生成函数。它的输入是私匙和随机种子 X，输出两个二进制串：生成的伪随机数δ和验证证明π。

$$VRF(SK,x) = (\delta(SK,x), \pi(SK,x)).$$

- $FVER$是一个验证函数。它根据输入的验证值几和一些公共信息（比如种子 x 和生成者的公匙)来判断随机数δ是否是通过正确的流程生成的随机数。

$$F_{VER}(PK,x,\delta,\pi) = True\ or\ False$$

一个满足条件的 VRF 必须具有以下的属性：

1. **正确性**：至少有有 $1-2^{-\Omega(\lambda)}$的概率，以下的两个条件成立：

•输入输出范围正确性：对于任何$x \in \{0,1\}^{a(\lambda)}$， 我们都可以得到$\delta(SK,x) \in \{0,1\}^{b(\lambda)}$。

•完全可证明性：对于任何$x \in \{0,1\}^{a(\lambda)}$，如果$VRF(SK,x) = (\delta,\pi)$，那么$P(F_{VER}(PK,x,\delta,\pi) = True) > 1 - 2^{-\Omega(\lambda)}$。

2. **唯一可证明性**：对于任意的$PK,x,\delta_1,\delta_2,\pi_1,\pi_2$，当$\delta_1 \neq \delta_2$时，对任何$i \in \{1,2\}$，以下不等式成立：

$$P(F_{VER}(PK,x,\delta_i,\pi_i) = True) < 2^{-\Omega(\lambda)}$$

重分片过程

Merkle RO 中的矿工会收到来自所有分片的区块头信息。区块头信息中会包含：

- 该分片的当前链高度h;
- 当前区块的随机种子Qr。

矿工根据分片内链的当前高度决定是否进行重分片。在目前的协议中，我们设定当分片内区块链每增长 n 个块，或者分片在上一次重分片$(t0)$以来已经运行了至少ts的时间，会触发重分片。从数学的角度来描述，则重分片会在以下两个条件之一被满足的时候被触发：

- $h \equiv 0\ (mod\ n)$或者$t \geq t_0 + t_s$

当重分片开始后，所有已经交付了押金的矿工都可以选择参与重分片。参与重分片的矿工的押金并不是以它的绝对代币数来计数的，而是以所谓“押金单位”为计数。一个押金单位代表多少 Merkle RO 代币是由当时分片的拥挤程度以及整个系统的安全需求来进行动态调整的。

假设矿工m的押金为αm押金单位。这个矿工会通过 VRF 来使用他的私匙 SK 和当前区块的随机种子Qr生成一个伪随机数 prob 和它的验证 ver

$$prob, ver = VRF(SK, Q^r).$$

假设系统设置的安全级别为$0 < s' < 1$（我们会在后文详细介绍安全级别的概念），那么当这个矿工产生的随机数$prob < 1 - (1-s')^{\alpha_m}$时，它就被选入了这个分片。根据 VRF 的伪随机性，存在一个极小的正数ϵ，使得这个矿工被选入分片的概率符合以下不等式：

$$|P(m \in M_{i,r}) - (1 - (1 - s')^{\alpha_m})| < \epsilon.$$

如果 m’是另一个矿工，那么这个不等式可以等价理解为下列不等式：

- 当 $\alpha_m > \alpha_{m'}$ 时，$P(m \in M_{i,r}) > P(m' \in M_{i,r})$;
- 当 $\alpha_m < \alpha_{m'}$ 时，$P(m \in M_{i,r}) < P(m' \in M_{i,r})$;
- 对于任意两个分片 i 和 i'，$P(m \in M_{i,r}) = P(m \in M_{i',r})$。

这说明的是，一个矿工被不同分片选中的概率是相同的，而押金高的矿工有更高的概率被某个分片选中。

安全等级 s’

安全系数 s’决定了某个分片内矿工数量的多寡。当安全系数较高时，矿工有更大的概率被选入分片，从 而使得分片内有更多的矿工。通过提高安全系数，可以使得系统：

- 更安全，因为分片内矿工数增加，使得攻击者更难攻陷分片内的大多数矿工；
- 效率降低，因为矿工数量多了，使得达成共识所必须的交流和网络传输增加。

区块链的安全需求相对来说是比较高的。我们当前默认的安全等级 s’为

$$s' = \frac{1}{分片的总数}$$

每个分片内的矿工数量的期望值是相同的。然而，在其他的应用程序中，应用程序的维护者可以自由设置符合自己需求的安全等级，从而使得安全需求较低的应用程序可以获得更快的运行速度和更低的共识成本。我们称这种技术为弹性分片。在当前，大多数的区块链项目的安全性是固定的，而弹性分片技术给予了用户更高的自由度，从而在安全和速度中达成更好的平衡。

安全性分析

假设 p 是网络中的诚实比例，c 是诚实阈值。我们要求整个网络中诚实矿工的数量是 $N_h = \rho N$，并且重分片后某分片内的诚实节点数 $N_{i,h} \geq cN_i$。通过前文的分析，一个有αm个押金单位的押金的矿工被选入某个安全系数为 s’的分片的概率为

$$p = 1 - (1 - s')^{\alpha_m}。$$

为了简单起见，我们假设所有矿工有相同数额的押金。设 B(n,p)为实验数是 n，成功概率是 p 的二项分布，并设 $N(\mu, \sigma^2)$ 为平均值μ，方差σ2 的正态分布。我们可以定义两个随机变量：

- $X \sim B(N - N_h, p) \approx N(p(N - N_h), (N - N_h)p(1 - p))$ 为被选入分片的作恶矿工数量。
- $Y \sim B(N_h, p) \approx N(pN_h, N_h p(1 - p))$ 为被选入分片的诚实矿工数量。

我们有

$$P(N_{i,h} \geq cN_i) = P(Y \geq c(X+Y)) = P((1-c)Y - cX \geq 0).$$

定义随机数$W = (1-c)Y - cX$。可知$W \geq 0$代表着某个分片内的诚实节点数满足该分片的诚实需求。因为 x 和 y 是两个独立的时间，我们可知 w 满足正态分布$N(\mu, \sigma^2)$，其中

$$\cdot \mu = (1-c)pN_h - cp(N - N_h) = pN_h - cpN = p(\rho - c)N$$

$$\cdot \sigma^2 = (1-c)^2 N_h p(1-p) + c^2(N - N_h)p(1-p) = [(1-c)^2\rho + c^2(1-\rho)]p(1-p)N$$

设 Z 为标准正态分布 N(0,1)，则

$$P(W \geq 0) = \Phi\left(\frac{p(\rho - c)N}{\sqrt{[(1-c)^2\rho + c^2(1-\rho)]p(1-p)N}}\right)$$

$$= \int_{-\infty}^{\frac{p(\rho-c)\sqrt{N}}{\sqrt{[(1-c)^2\rho+c^2(1-\rho)]p(1-p)}}} \frac{1}{\sqrt{2\pi}} e^{\frac{x^2}{2}} dx,$$

因此

$$P(N_{i,h} \geq cN_i) = 1 - \int_{-\infty}^{a} \frac{1}{\sqrt{2\pi}} e^{\frac{x^2}{2}} dx,$$

其中$a = \frac{p(\rho-c)\sqrt{N}}{\sqrt{[(1-c)^2\rho+c^2(1-\rho)]p(1-p)}}$。

假设某个矿工质押了 10 个单位的押金，安全等级 s’ = 0.1，诚实率 p = 0.8 以及诚实阈值 c = 0.75。我们 可以将$P(W \geqslant 0)$视为一个关于 N 的函数。图 4 是一个$log(N)$和$log(1-P(W \geqslant 0))$关系。

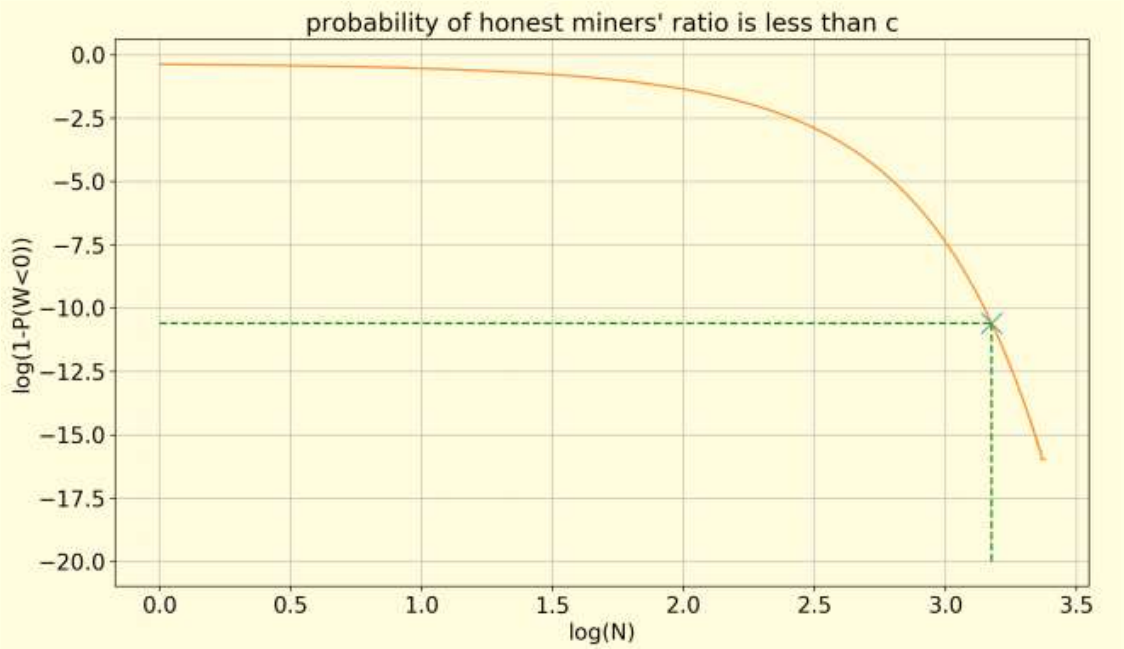


图 4 ： log(N)对比$log(1-P(Ni,ℏ\geqslant cNi))$的图。在 N=1500 （大约$log(N)\approx 3.17$）的时候，诚实节点数低于阈值的概率小于$10^{-10}$。

可见当$N\geqslant 1500$时，被分片选中的诚实节点数大于阈值cNi的概率低于10^{-10}。作为对比，比特币当前运行的节点数为 10424，以及以太坊的节点数为 14383。所以，一个分片选中的诚实节点数低于预设阈值的比例的概率微乎其微。即使发生了小概率事件，一个分片内的诚实节点数低于了预设的诚实阈值，攻击者依然无法发送并通过错误的 Transaction。他只能阻碍当前产生新的区块，并且只要分片进行重分片，他就会失去对该分片的控制力。

3.3 分片分裂

Merkle RO 通过使用分片分裂来增加分片的数量，从而动态地实现可伸缩性。如果网络意识到一个特定的分片内的 Transaction 数量始终处于过载状态，那么这个分片会被分裂成两个分片，每个新生成的分片会包含一半原分片的账户。
假设原来的分片是

$$S_{i,r}=\left(C_{i,r},(B_{i,h}),(H_{i,h}),MT_{i,r},M_{i,r},C_{i,r},S_{i,r}\right)$$

那么，当分片分裂之后，该分片会分成两个新的分片

$$S_{2i,r}=\left(C_{i,r},(B_{i,h}),(H_{i,h}),MT_{i,r},M_{2i,r},C_{2i,r},S_{2i,r}\right)$$

$$S_{2i+1,r}=\left(C_{i,r},(B_{i,h}),(H_{i,h}),MT_{i,r},M_{2i+1,r},C_{2i+1,r},S_{2i+1,r}\right)$$

其中

$$C_{i,r}=C_{2i,r}\sqcup C_{2i+1,r}$$

分片的分裂不会影响到存储节点内存储的内容。

尤其是当分片分裂刚发生的时候，原始分片的存储节点需要同时为两个新分片提供服务一小段时间。当两个新分片都有了足够的存储节点提供服务之后，同时服务于两个分片的存储节点可以选择停止为其中一个新分片提供服务，以避免它的网络和存储负担过度。

在分片分裂之后，新分片会立刻开始重分片并重新选举自己的矿工（$M_{2i,r}$和$M_{2i+1,r}$）。由于轻节点账户是根据地址分配给不同的分片的，我们会将分片 i 的地址分配给新分片 2i和 2i+1。如果原始分片提供的地址是以 00 开始的，那么在分裂后，新的分片 2i负责开始为 000 的地址，以及新的分片 2i+1 负责开始为 001 的地址。如图 5 所示：

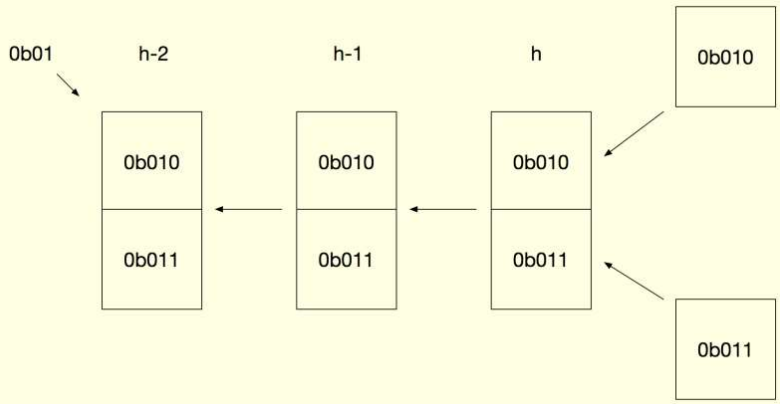


图 5 基于账户地址进行分片分裂

3.4 片内共识

在 Merkle RO 对分片的定义中

$$S_{i,r} = \left(C_{i,r}, (B_{i,h}), (H_{i,h}), MT_{i,r}, M_{i,r}, C_{i,r}, S_{i,r}\right)$$

不同的分片其实可以拥有不同的共识算法 C;，。理论上来说，分片内使用的算法可以是任何可靠的共识算法，包括 PoW(虽然我们不推荐使用 PoW)。在 Merkle RO 中，我们会选择使用拜占庭共识算法，该算法假设正在运行的诚实节点的数量占全部节点数量的 2/3 以上。拜占庭算法拥有最终一致性，代表着每个分片的区块链是唯一的，并且分叉的概率非常低。拜占庭共识算法的例子有 pBFT[22]、Tendermint[23]和 BA*[19]，目前 Merkle RO 团队正在研究和开发一个在速度和扩展性方面的都适合我们需求的拜占庭共识算法。

一旦分片完成矿工的选择，分片会通过 VRF 来选择矿工进行共识过程中不同的事务，比如投票和出块。使用 VRF 的好处是，并非所有矿工都需要参与共识过程中的每个任务，这可以显著减少网络流量并加快共识的速度。同时，因为决定进行具体事务的矿工的随机数是不可操控的，并且是可以被其他用户验证的，所以即使并不是所有的矿工都参与到了每个步骤，Merkle RO 的共识也可以提供很高的安全性。

接着，我们介绍用于共识的分片内矿工选举过程。我们将每两个块的验证时间之间的长度称为区块时间。每个区块时间会被分解为多个子任务，大多数子任务都是运行某种算法来对某个决定进行投票。假设某个区块时间为 r，当前子任务为 t。从前一个区块的区块头中我们可以获取一个随机种子$Qr-1$，每个当前分片内的矿工会根据此生成一个随机数

$$h = VRF(SK, hash(Q^{r-1}, r, t))$$

如果一个矿工生成的随机数h小于某个预先被决定好的阈值pr,t，则会被选中参与当前的子任务t。

分片内选举子任务执行人和通过 VRF 进行重分片的矿工选举的流程很类似，但是有个很重要的区别：在分片内部，矿工并不会根据他的押金数而获取不同的权重。所有矿工在分片内都有同等的投票权，这使得某个拥有了大量权益的矿工无法通过押上大量资金而在某个分片内获取过高的投票权。

在区块时间中，第一个任务是创建新的区块。在共识的开始，每个被选中进行区块创建任务的矿工都会将一部分自己缓存的 Transaction 打包形成一个区块 Bu 并将其在分片内进行广播。在新的区块创建的过程中，所有被选中的矿工生成的随机数 h 也会同时作为区块的优先度。所有接收到潜在区块的用户会缓存收到的所有潜在区块，并在最终选择优先度最高的那个矿工生成的区块。

4.Merkle RO 存储和传输分片机制

本节将介绍 Merkle RO 的分片存储方案和传输方案。

Merkle RO 的区块链网络是完全的分片，这意味着我们不仅要在矿工之间分配计算负载，还要在节点之间划分存储负载。通过这样的设计，矿工只需要与同一分片中的其他矿工通信就可以获取和处理 Transaction，从而显著地降低了传输成本。Merkle RO 是第一个实现完全分片的区块链系统，在计算、存储、传输三个层面均进行分片，对区块链扩容至产业级容量十分重要。

实际上，存储负载也是区块链系统目前面临的一个主要瓶颈。实现高 TPS(每秒事务处理数）的区块链也会相应地产生极大的存储负载。如果一条公链的 TPS 达到 2000，假设每个 Transaction 的数据量大约 400 字节，那么每个月产生的存储数据量将有 1931GB 大小。这种情况下，让每个节点存储所有区块数据是完全不可能的。

Merkle RO 通过存储分片来解决这个问题。在 Merkle RO 中，矿工节点和存储节点进行了分工：矿工节点负责生成区块，在系统中掌握投票权，但不需要存储全部数据；而存储节点负责存储和提供数据，充当存储服务的提供商，但不掌握任何决策和增删改权力。Merkle RO 倾向于尽可能地减少矿工的本地存储和传输量，以便可以有大量的普通计算设备加入到挖矿网络中，从而使出块权力能够始终保留在普通用户手中，保证公平性和去中心化，从而提高网络的健壮性和效率。

Merkle RO 的存储和传输分片解决方案有以下特点：

- 矿工节点不再保留也不再需要完整的全账本。假设有一个 TPS 可达到 VISA 水平（平均 2000)的区块链系统，如果仍然像现存的区块链系统一样，每个矿工都要存储一份全账本，那么每年这个账本将增加 23T 的数据，并且随着时间的推移线性增长。这对于普通计算设备来说是一个巨大的数字，这种方式也限制了普通设备加入区块链网络。而在 Merkle RO 中，大多数数据由存储节点保存，矿工在任何时候，都只需要保存不到 1G 的数据。
- 分片内 Gossip 协议的实现。在比特币这样的传统区块链系统中，消息通过 gossip 协议广播给全网的全部用户。而在 Merkle RO 中，我们实现的新的 gossip 协议使得仅向特定分片内的

用户广播 Transaction 成为了可能，允许节点只接收他们感兴趣或与他们有关的 Transaction 信息。

- 存储节点能够提供可被验证的、值得信赖的服务，而不具备中心化风险。虽然 Merkle RO 的大多数数据都是存储节点存下来的，矿工向存储节点发送请求获取数据，但矿工也必须存储足够的信息以验证返回的数据是准确合法的。Merkle RO 的设计能够使存储节点提供的所有信息都可以被矿工验证，并且所有未 Merkle RO 存储分工验证的信息都无法使用。这不仅保证了数据的合法性，而且使得存储节点仅能作为服务商提供可靠的服务，而不能拥有增删或篡改数据的权力。
- 矿工无需承担额外的网络传输。矿工节点只需要在本地缓存 Main Merkle Tree 的 Merkle Root 就足够完成区块和 Transaction 的验证。在 Merkle RO 的设计中，矿工仅仅需要新生成区块的信息就足以更新 Main Merkle Root，从而无需对存储节点造成额外的传输压力，矿工就能够拥有可产生新的区块和重分片的全部信息。

4.1 存储方法

Transaction 输出的 Merkle Tree

Merkle RO 的存储节点以 Merkle Tree 的形式存储着 Transaction 的输出状态。这些 Transaction 被打包成区块，区块中的输出状态的 Merkle Tree，称为 Block Merkle Tree(块默克尔树）。将所有的 Block Merkle Tree 进行哈希编译，就生成了 Main Merkle Tree(主默克尔树），包含了全账本的信息。

接受者的账户 ID 决定了一笔 Transaction 将在哪个分片中被存储，该分片的存储节点负责维护 Main Merkle Tree 中它所在的分片的 Block Merkle Tree 的信息。Merkle Tree 提供了一种存储和高效通讯的方式，既能存储 Transaction 输出，又能支持其他节点通过验证 Merkle Path 来验证 Transaction 输出的合法性。

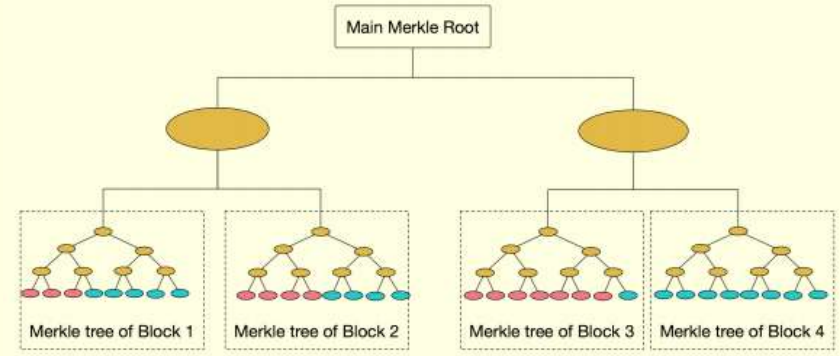
在网络中，矿工需要跟踪以下信息：

- 所有的区块头（Block Header)
- 所有分片的 Main Merkle Tree
- 根据所有分片的 Merkle Root 更新 Merkle Path

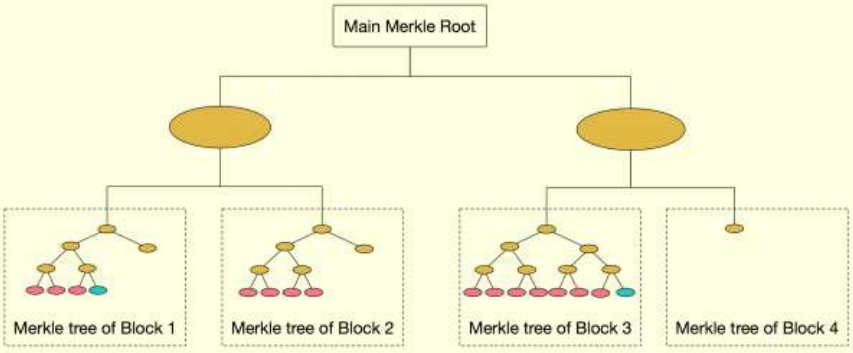
存储节点需要知道所有的区块，但只负责维护和更新它所负责的分片的账户的 Transaction 输出。因此，不同分片的存储节点会存有不同的 Main Merkle Tree(因为存储节点只负责自己的分片，其他分片的区块信息没有更新）。一旦 Transaction 状态更新，区块的哈希就会立即改变，导致 Block Merkle Tree 也会改变。这样使得每个存储节点都能记录下相同的 Transaction，但只需要更新它负责处理的 Transaction。存储节点也可以选择性地删除它不负责的 TransactionOut，只需要在必要的位置保留相应的哈希值即可。

每个新的区块中都包含一组刚被确认的 Transaction，这些 Transaction 存储在 Block Merkle Tree 中。这些 Transaction 的输出需要被记录下来，它们将在未来作为某笔 Transaction 的输入被花费。对于每个经过共识产生的区块，负责的矿工都会构建 Block Merkle Tree 并把它添加到区块头中。在 Block Merkle Tree 的内部，每个输出都用 0/1 来标记，表示这笔输出是否已经被花费掉。

这是输出根据接受者地址和 Transaction 哈希值进行排序，使相邻分片的 Transaction 相邻，以便于后续的验证。由于这个严格的排序规则的存在，在确定的 Transaction 列表的情况下，生成的 Block Merkle Tree 也是确定的。这一方案在图 7 中展示：



(a) *The Full Main Merkle Tree*



(b) *The Pruned Main Merkle Tree Stored by the Storage Nodes*

图 6 存储节点中的 Main Merkle Tree。其中红色点是分片 1 的输出，蓝色点是分片 2 的输出，橙色点是 Merkle Tree 的中间节点。图（a)是完整的 Merkle Tree，图（b)是分片 1 的存储节点维护的 Main Merkle Tree，剪枝掉了分片 1 不负责存储和管理的输出以及一些不必要的中间节点。

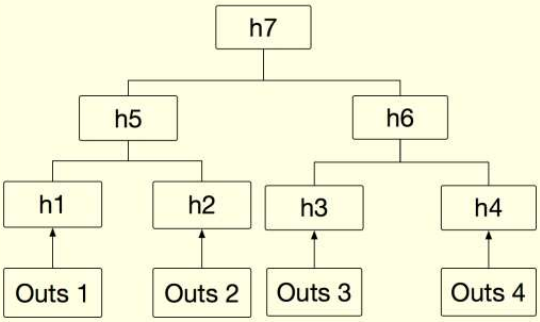


图 7:一个 Block Merkle Tree。新的输出被标记为 Out 1 至于 Out 4。哈希值 h1 至 h4 分别是相应 out 的哈希值：hj_ = hash(Outs i, 0);哈希值 h5、h6 和 h7 由它们的子节点的哈希值计算而来。

主默克尔树（Main Merkle Tree)：

存储节点将所有的历史输出都保存在一个默克尔树中，这棵默克尔树就成为 Main Merkle Tree。它由两部分构成，Top Merkle Tree 和各个 Block Merkle Tree。Top Merkle Tree 的叶子节点是所有 Block Merkle Tree

的根节点。当有新的区块需要添加时，存储节点就会将新的区块的 Block Merkle Tree 加到 Top Merkle Tree 中。我们在图 6(a)中说明了这种结构，为了便于理解，图中假设只有两个分片。

回想在第二章中的定义，输出由一个值和一个接受者地址组成。分片 1 中的存储节点只对分片 1 中的输出地址感兴趣，为了提高存储效率，它可以丢弃输出到其他分片的 Transaction 的子树，只保留它在本分片需要维护的输出的 Merkle Root

当一个用户发起一个 Transaction 后，存储节点会计算出这个 Transaction 中涉及到的 Out 的 Merkle Path，并将其与 Transaction 一起转发给矿工节点。矿工节点会在本地维护一份 Main Merkle Tree 的根信息，并用这份信息和收到的 Merkle Path 比较，从而判断收到的 Out 数据的正确性以及确认该笔 Out 尚未被花费。如果一个作恶的存储节点发送了错误的信息给矿工，那么他必然无法提供一个可通过验证的 Merkle Path，于是矿工会拒绝使用错误的信息作为 Transaction 的凭证。所以，一个作恶节点唯一能对系统造成的危害是其拒绝提供服务，但是在一个分片内能够提供服务的存储节点往往不止一个，所以矿工总是可以轻松地转移向另一个存储节点索取数据。

4.2 区块确认消息

为了验证存储节点提供的信息，矿工需要维护所有分片的 Main Merkle Tree 的根信息。所以每当某个分片产生了一个新的区块并更新了它的 Main Merkle Tree，它都必须将这个消息通知所有的矿工以便他们更新 Main Merkle Tree 的信息。

但是某个分片只是单纯地通知矿工这个新的 Main Merkle Tree 根信息是不足够的，因为矿工还需要一个方法来确定这个信息是真实的，已经通过共识的。Merkle RO 通过要求分片广播所有参与共识的投票者的投票以及其签名来保证该信息的正确性。每当某个分片达成共识之后，已经确认产生新的区块的矿工会将所有的签名投票信息与区块头信息打包并广播至全网。这个消息就叫做区块确认消息。这个消息足够使矿工能够验证信息来源的可靠，并且更新他缓存在本地的 Main Merkle Tree 根。分叉的可能性极小。一个作恶的矿工通过发送虚假的区块确认消息来诱发区块链分叉的可能性是微乎其微的，因为包含在区块确认信息中的投票信息必须包含投票者的签名和资质信息。一个作恶者需要入侵一个分片内的大部分矿工才有可能收集到足够的签名和资质信息。尽可能降低不必要的网络传输。如果所有的矿工在达成共识后都会广播一份区块确认信息，那么这会造成相当巨大的网络压力。Merkle RO 的协议中使用了 gossip 网络，而每个矿工最多只会发送或转发一份区块确认消息，已经被转发过的区块确认消息会被丢弃。这保证了不会因为发送区块确认消息而导致过高的网络负载。

4.3 矿工的更新

在这一章中，我们会介绍一下矿工是如何保证自己始终有足够的信息去更新自己的区块链账本的。

首先我们强调一下，矿工需要保存在本地的信息包括

- 分片的区块头信息
- 所有分片的 Merkle Root 信息
- 一部分用来帮助更新 Main Merkle Tree 的 Merkle Path 信息

当一个矿工负责产生新的区块时，他负责计算并在区块头部添加两个 Merkle Root：该区块内的 Block Merkle Root 信息和新的 Main Merkle Root 信息。该矿工需要进行以下的更新操作：

- 将 Main Merkle Tree 中被使用的 TransactionOut 的状态修改为 1，并据此计算新的 Main Merkle Tree 根信息；
- 另外，将上一个区块出块以来收到的其他分片的 Block Merkle Root 添加入当前 Main Merkle Tree；
- 最后，需要将缓存池内的 Transaction 的 Merkle Path 根据新的 Main Merkle Root 进行更新。

Merkle RO 设计了一个流程，使得矿工可以在无需获取完整的 Merkle Tree 的信息的前提下更新自己的本地缓存信息。为此，每个矿工需要获取如下信息：

- 新的区块中所有输入的 Transaction 的 Merkle Path；
- 从该分片更新上一个区块到现在为止所有来自于其他分片的新的区块的头部信息；
- 当前 Main Merkle Tree 最右侧的 Merkle Path 信息。这个信息是用来帮助矿工将新产生的区块的 Block Merkle Tree 根加入 Main Merkle Tree 的，这个信息会在某个矿工第一次加入分片后从存储节点处获取。

接下来我们为矿工如何更新本地信息提供更多的细节。

通过修改已使用 Transaction 的状态而改变 Merkle Root

区块创建者首先要追踪他所创建的区块内的所有 Transaction 并负责计算出将这些 Transaction 的输入状态修改为“已使用”以后新的 Merkle Tree 的根值。第一步是获取某个输入 Transaction 的信息：

Pseudocode 1: *getHash*

```
INPUT:
    P_t = ((H_0, S_0), (H_1, S_1), ..., (H_n, S_n)): Merkle
        path of a transaction t
    i: The location of the required hash
OUTPUT:
    H: A hash value in the merkle path
    S ∈ {L, R}: The side of the hash value
PROCEDURE:
    H ← H_i
    S ← S_i
```

矿工需要将这个输入 Out 的状态从 0 修改为 1，代表该 Transaction 已经被使用，然后计算一个新的 Main Merkle Tree 的根值。矿工仅仅只需要这个输入 Out 的 Merkle Path(而不是完整的 Merkle Tree)就可以完成这个更改。在这里我们提供了修改一个输入 Out 的伪代码，修改多个 Out 的逻辑是非常类似的。

Pseudocode 2: *updateRoot*

```
INPUT:
    t: Transaction
    P_t: Merkle path of the transaction t
OUTPUT:
    R: New Merkle Root
PROCEDURE:
    R ← hash(input, 1)
    for i in P_t; do
        ℋ, 𝕊 ← getHash(P_t, i)
        if 𝕊 = 𝕃; do
            R ← hash(ℋ, R)
        else; do
            R ← hash(R, ℋ)
    Output R
```

通过添加新的区块而改变 Merkle Root

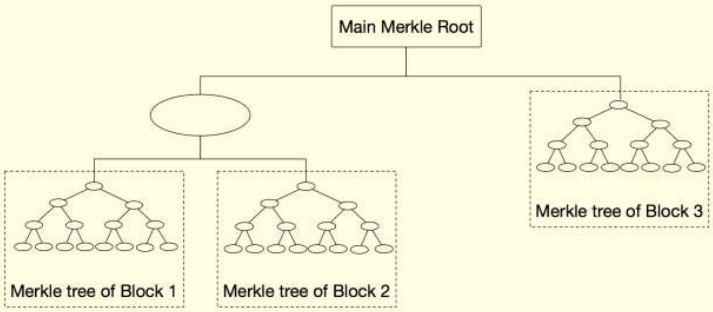
在更新两个区块之间，矿工会收到来自其他分片的被确认的区块的信息。当一个矿工负责产生新的区块并更新 Merkle Tree 根值的时候，这个矿工被要求负责将这段时间收到的（来自其他分片的）新的区块也添加入 Main Merkle Tree 根中。

矿工会将所有的来自其他分片的区块按照它们的分片序号 i 和区块所在高度 hi 按照(i, hi)的字典序进行排序，并按照该顺序将这些区块依次挂入 Main Merkle Tree。每个分片的新高度会被记录并在分片内广播。值得一提的是，每个分片的新高度一定会大于等于该分片在上一轮的高度，并且如果某个区块的新高度和上轮相等，那代表着在当前分片出两个分片的过程中没有收到来自该分片的新的区块。

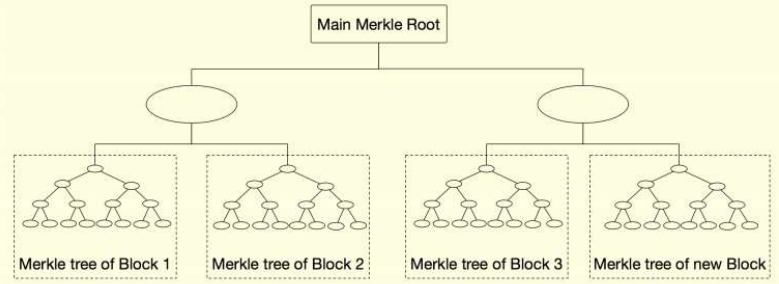
由于网络的异步性，不同的新的区块提议矿工所缓存的来自其他分片的区块是不完全一样的，从而他们计算出并包含在他们的潜在区块中的 Main Merkle Root 可能会有所不同。不过最终只有一个区块，亦即只有一个 Main Merkle Root 能够通过共识。由于在已知当前分片新产生的区块的情况下，Main Merkle Root 是可以根据当前每个分片的最新高度推测出来的。所以只要矿工在新产生的区块内附加上他收到的所有分片的最新高度，那么其他矿工就有了足够的信息来验证该区块内的 Main Merkle Root 的准确性。

我们设计了一个机制，使得矿工仅仅需要 Block Merkle Tree 的根值就可以完成这步更新。这解决了一个分片的重要问题：当网络扩展时，大量的跨分片信息传播会产生严重的网络传输压力，并且大幅度限制扩展性。在我们的方案中，我们尽力限制了跨分片信息的传播，使得无论网络如何扩展，来自同一个分片的信息量都被限制在一个近乎固定的程度。为此，我们需要矿工保存 Main Merkle Tree 的最右侧 Path，用以定位新的区块被添加的位置。

在 Merkle RO 中，新的区块总是被添加在 Main Merkle Tree 的最右侧。当矿工试图添加一个新的区块的时候，会遇到两种情况。当总 Merkle Tree 未满的时候，矿工只需要直接将区块添加在下一个空置的叶子节点处即可；当 Main Merkle Tree 全满时，矿工需要增加树的高度，使得树的右侧出现新的空置叶子节点。图 8 和图 9 中分别表示了这两个操作。

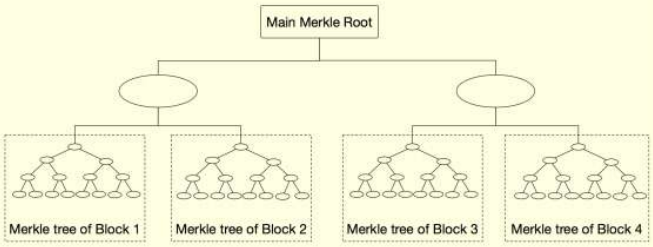


(a) *Original main Merkle tree*

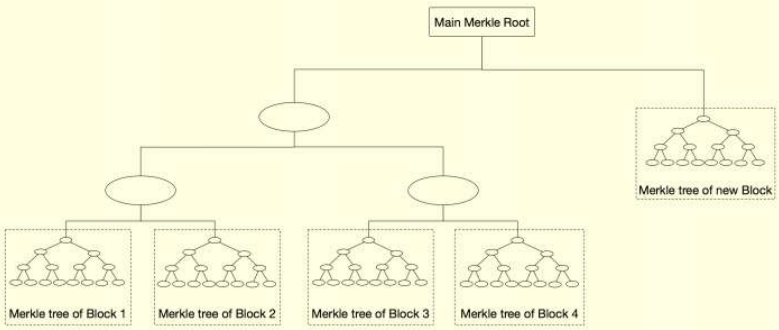


(b) *New main Merkle tree*

图 8 此图描述了一个 Block Merkle Tree 如何被挂到一个不完整的 Main Merkle Tree 上。



(a) *Original main Merkle tree*



(b) *New main Merkle tree*

图 9 此图描述了一个 Block Merkle Tree 如何被挂到一个完整的 Main Merkle Tree 上。

(a)原 Main Merkle Tree (b)新 Main Merkle Tree

为了执行这些操作，矿工需要不停更新最右侧的 Merkle Path，从而使得负责产生新的区块和更新新的 Main Merkle Tree 根值的矿工不但能添加当前分片的新的区块，还可以添加来自其他分片的区块。

在新的区块被确认后更新矿工本地缓存

当矿工收到一个来自自己分片的新的区块后，他需要更新一些本地缓存的信息：

- Main Merkle Tree 的根值
- Main Merkle Tree 最右侧的路径/Path
- Transaction 缓存池中的 Transaction 的 Merkle Path

更新的逻辑见如下代码：

Pseudocode 3: *updatePath*

```
INPUT:
    P_t^old: Merkle path of pending transaction
    P: Merkle path of a confirmed transaction
    O: An output

OUTPUT:
    P_t^new: New Merkle path of the pending
        transaction
PROCEDURE:
    H_old ← hash(O, 0)
    H_new ← hash(O, 1)
    if H_old in P_t^old; do
        P_t^new ← replace H_old by H_new in P_t^old
    for i in P; do
        H_tmp, S_tmp ← getHash(P, i)
        if S_tmp = L; do
            H_old ← hash(H_tmp, H_old)
            H_new ← hash(H_tmp, H_new)
        else; do
            H_old ← hash(H_old, H_tmp)
            H_new ← hash(H_new, H_tmp)
        if H_old in P_t^old; do
            P_t^new ← replace H_old by H_new in P_t^old
    Output P_t^new
```

4.4 存储节点的更新

接下来我们介绍一下存储节点是如何更新自己的存储内容的。

在这里我们强调一下，存储节点需要保存所有的收款人是该分片的历史 TransactionOut 的数据。因为这些 Transaction 是按照收款人分类的，它们可能来自于不同的分片。存储节点在处理片内产生的 Transaction 和片外产生的 Transaction 的时候，处理逻辑是有所区别的。

当某个分片产生了一个新的区块之后，出块的矿工会整合一个包含了所有属于相同分片的 TransactionOut 的 Output 信息，并将该信息发往相应的分片。存储节点会监听所有发往自己分片的 Output 信息（来自其他分片），以及来自自己的出块信息（因为直接收到完整的区块信息，所以无需等待 Output 信息）。当它收到分片的 Output 信息，并将生成的 Block Merkle Tree 挂在自己的 Main Merkle Tree 上，并更新根值。

接下来我们详细介绍一下这些步骤。

Output 信息

每当分片;产生了一个包含了发往分片 j 的地址的 Transaction 的区块，矿工都会生成一个从分

片;至分片 j 的 output 信息。这个信息包含了

- 这个区块内包含的 TransactionOut 的总数;
- 所有发往分片 j 的 TransactionOut 中，最左边（排序第一的 Transaction 之前）的 Transaction 的信息和路径;
- 所有发往分片 j 的 TransactionOut 中，最右边（排序最后的父易之后）的父易的信息和路径;
- 所有发往分片 j 的 TransactionOut 的具体内容。

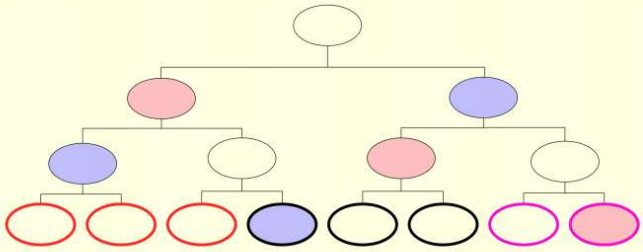
Output 信息代表着在分片;产生了一个包含了一些发往分片 j 的 TransactionOut 的新的区块。分片 j 的存储节点在收到 Output 消息之后，会根据消息内包含的数据重建一个仅仅包含了发往分片彡的 TransactionOut 的部分 Block Merkle Tree，以便分片 J/的存储节点将这个 Block Merkle Tree 挂入 Main Merkle Tree。

更新来自其他分片的区块信息

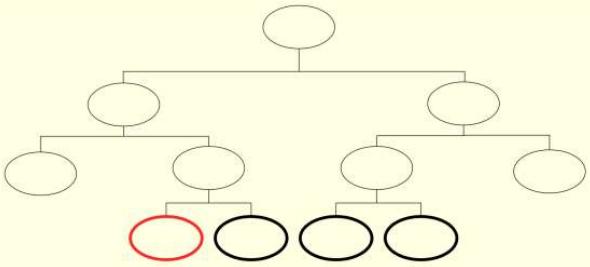
当一个存储节点收到一个来自其他分片的 Output 消息之后，它并不会立刻将这个消息更新入 Mam Merkle Tree 中，而是将这个消息缓存。

存储节点只有在收到来自自己的分片的新的区块后才会更新 Main Merkle Tree。更新时，首先它会从来自其他分片的 output 信息中提取出相应的数据，并重建一棵部分 Block Merkle Tree (只包含属于自己的分片的 TransactionOut)。图 10 给出了一个部分 Merkle Tree 的例子。

自己分片的出块信息之后，它会处理所有的来自其他



(a) *Full Merkle Tree of Output Messages*



(b) *Pruned Merkle Tree of Output Messages*

图 10:红色圆圈表示分片 1 的输出，黑色边框的圆圈表示分片 2 的输出，粉色边框的圆圈表示分片 3 的输出。假设有一个分片 2 的存储节点，它仅对分片 2 的输出感兴趣。图（a)是存储节点接收到的完整的输出消息的 Merkle Tree，蓝色圆圈就是分片 2 的最左路径/Path，红色圆圈则是分片 2 的最右路径/Path。图（b)是被剪枝后的输出消息的 Merkle Tree，该存储节点根据最左和最右路径进行了剪枝后将该 Merkle Tree 保存起来。

(a)完整的输出 Merkle Tree(b)剪枝后的输出 Merkle Tree

更新来自自己分片的区块信息

一个新的区块中会包含着新的 Merkle Tree 根值以及这个新的区块被创建时，其他分片的区块高度。
当一个来自同一分片的区块被收到以后，这个区块内包含了矿工当前收到的所有分片的最高高度 hi。存储节点会根据其他分片的区块高度，将自己缓存的 Output 消息重建 Partial Block Merkle Tree，并将这些重建的 Block Merkle Tree 挂载在自己的 Main Merkle Tree 的最右端。最后它会将收到的自己分片的 Block Merkle Tree 挂在新的 Main Merkle Tree 的最右端。当 Main Merkle Tree 更新完成之后，存储节点会计算并检查自己生成的 Main Merkle Tree 的根值是否和区块内提供的根值相等。因为网络是异步的，有可能存储节点会漏收一部分的来自其他分片的 Output 信息。当这个情况发生的时候，存储节点会在等待 5 秒之后，试图直接向其他分片索要它漏掉的 Output 消息，以便自己更新自己的 Main Merkle Tree。
将上述的步骤整合，就是存储节点更新自己的存储内容的过程，从而保证了自己存储的 Transaction 信息是及时的。

4.5 Merkle RO 如何抵抗操纵？

“操纵”指的是攻击者作出特定行动以获得最高的利益。攻击者的行动空间是利用 自己和盟友所能控制的资产和账户进行转账操作。其中，转账操作的金额不超过攻击 者拥有的资产数目；转账的发起方是其能够控制的账户，包括攻击者及其盟友创建的账户，以及愿意提供资产中转服务的服务商账户等等。攻击者能够获得利益一般由其知晓私钥的账户的评分决定。如果有多个这样的账户，一种简单的情况是，攻击者的利益正比于这些账户的得分总和。当然，先前提到的服务商账户的私钥不受攻击者掌控。

4.6 存储和传输的总结

综上，Merkle RO 利用矿工和存储节点之间的分工，实现了区块链系统的存储分片和传输分片。矿工节点负责处理 Transaction、达成共识、产生区块，存储节点负责存储和提供本分片的所有历史数据和 Transaction 细节。
为了能够让整个系统进行安全、可验证的数据交互，矿工节点需要存储所有区块头、所有分片的 Main Merkle Tree 的根当前分片 Main Merkle Tree 的右侧路径。这样的设计使得矿工能够留有所有分片的区块链状态的摘要，并依据这个摘要处理与摘要已知的 Transaction，同时将矿工的硬件存储要求保持在非常低的水平，使得普通计算设备也能够担任矿工。存储节点为某个特定分片服务，为该分片存储区块链状态。这个存储解决方案使得区块链系统中的大多数通信都是在分片内部进行的，尽可能地减少了跨分片的数据交互，我们称之为传输分片。
通过对分片方案的进一步设计，Merkle RO 减轻了任意时间内任意网络部分的负载压力，从而实现一个快速并可扩展的区块链协议。

5.总结

工作量证明（Proof of Work)和权益证明（Proof of Stake)是区块链中两个最致命的共识机制。

比特币使用了中本聪创造的工作量证明共识算 法，要求通过计算复杂的随机数来竞争产生共识。工 作量证明能够提供较高的安全性，但它需要计算设备 持续不停地进行大量的随机数计算，这个过程是非常 无效且耗能的。这使得比特币系统在2017年消耗的电 量等同于整个爱尔兰岛消耗的电量。因为巨额的计算 成本，工作量证明不足以满足Merkle RO对高性能的需求。

实际上，工作量证明进行计算的本身并不直接用 于共识过程，它的主要目的是为了让用户证明身份， 以避免网络受到女巫攻击，或者作恶的用户伪装成多 个节点发起攻击。如果记账权仅仅是随即从现有的在 线用户中选择一个节点，那么攻击者就可以通过伪造 多个小号来尽可能提高自己被选中的概率，从而使自 己拥有操纵全网的能力。工作量证明强迫每个用户提 供一个自己拥有一部分有代价资源的证明（也就是算 力资源），证明自己有资格参与共识，并使得创建小 号失去意义。可以这样理解，比特币使用算力证明， 本质上来说类似早期HashCash曾使用工作量证明的方 法来验证垃圾邮件。

工作量证明还提供了另外一个重要功能：可验证 的随机性。它在证明了用户的身份的同时，又可以从 已有用户中选出记账的节点，并保证这个过程是随机的矿工得到了一个工作量证明的结果，他就既通过足 够量的计算证明了自己的身份（不是小号），又通过 提供一个随机数证明了自己赢得了这张计算彩票。这 个机制之所以有效。是基于人们相信，作恶节点很难 获取全网大部分的算力资源，而算力资源是进行工作 量证明的基础。

Merkle RO选择用权益证明（Proof of Stake)来替 代工作量证明，以避免对能源的浪费，但同时也能够 维持同等级别的安全性。作为工作量证明的替代，权 益证明是根据用户的资金量（也就是权益）来随即选 择记账者，资金量越多的用户越有可能被选中。在一 个较为成熟的网络中，控制大量的资金量是比较困难 的，并且提供一定量的资金证明也能够证明用户的身份，这也是权益证明为什么被认为也是足够安全的（实际上，这里还有一个隐含原因，一个控制了大量 资金的用户大概率不会做坏事，因为整体上这会削弱 整个系统，从而对大户产生极大的不良影响）。在 Merkle RO中，我们要求用户必须先锁定一部分资金作 为押金，才能够成为矿工节点参与共识和出块。当用 户通过权益证明证明了自己的身份后，我们使用VRF 以一种随机可验证的方式，从已经质押押金的矿工节 点中选出记账者，从而替代了工作量证明的验证和选 举的两个功能。

的、公平的、可以被其他用户验证的。当某个比特币的矿工得到了一个工作量证明的结果，他就既通过足 够量的计算证明了自己的身份（不是小号），又通过 提供一个随机数证明了自己赢得了这张计算彩票。这 个机制之所以有效。是基于人们相信，作恶节点很难 获取全网大部分的算力资源，而算力资源是进行工作 量证明的基础。

Merkle RO选择用权益证明（Proof of Stake)来替 代工作量证明，以避免对能源的浪费，但同时也能够 维持同等级别的安全性。作为工作量证明的替代，权 益证明是根据用户的资金量（也就是权益）来随即选 择记账者，资金量越多的用户越有可能被选中。在一 个较为成熟的网络中，控制大量的资金量是比较困难 的，并且提供一定量的资金证明也能够证明用户的身份，这也是权益证明为什么被认为也是足够安全的（实际上，这里还有一个隐含原因，一个控

制了大量 资金的用户大概率不会做坏事，因为整体上这会削弱 整个系统，从而对大户产生极大的不良影响）。在 Merkle RO 中，我们要求用户必须先锁定一部分资金作 为押金，才能够成为矿工节点参与共识和出块。当用 户通过权益证明证明了自己的身份后，我们使用 VRF 以一种随机可验证的方式，从已经质押押金的矿工节 点中选出记账者，从而替代了工作量证明的验证和选 举的两个功能。

当前区块链系统的可扩展性问题，是阻碍区块链技术在现实世界中大规模应用的主要问题。在众多的区块链扩容技术方案中，即便是目前最受关注的分片方案，也仍存在一定的局限性。虽然分片技术在并行化处理上取得了一定进展，但仍然受制于传输和存储的瓶颈，并未实现完全的可扩展性。Merkle RO 所设计的世界上首个区块链系统的全维度（计算、传输和存储）分片架构，希望将区块链技术真正推向实体经济。
Merkle RO 使用可验证随机函数（VRF)将网络中的矿工动态划分为若干分片，并行处理 Transaction。矿工以公平和公开可验证的方式动态轮换、随机重新分配到分片中，以保证安全性、抵御 Sybil 攻击。Merkle RO 的工程设计简洁优雅，通过节点的分工（轻节点、矿工节点、存储节点）实现传输和存储的分片，确保计算过程的任何部分都不再受到单个节点或汇总机制的瓶颈限制，确保普通矿工在区块链系统中始终拥有决策权力。Merkle RO 的架构能够实现线性扩展吞吐量，同时维护了区块链的核心价值、去中心化及安全性。这一全维度分片方案，能够使得 Merkle RO 作为公链底层平台，真正可靠地支持现实世界的大规模去中心化应用。

参考文献

[1] Nakamoto,Satoshi,Bitcoin:A Peer-To-Peer Electronic Cash System. (2008)
[2] Wood, Gavin. Ethereum: A Secure Decentralised Generalised Transaction Ledger. (2014).
[3] Larimer, Daniel et al. EOS.IO Technical White Paper v2. (2017)
[4] The Internet of Services Foundation. lost: The Next-Generation, Secure, Highly Scalable Ecosystem For Online Services. (2017). https://iost.io/iost-whitepaper/
[5] The QuarkChain Foundation. QuarkChain: A HighCapacity Peer-to-Peer Transactional System (2018)
[6] Kwon, Jae and Ethan Buchman. Cosmos:A Network of Distributed Ledgers (2018)https://cosmos.network/cosmos-whitepaper.pdf
[7] The AElf Foundation AElf - A Multi-Chain ParallelComputeing Blockchain Framework (2017)
[8] Poon, Joseph, and Vitalik Buterin. Plasma: ScalableAutonomous Smart Contracts. (2017)
[9] Poon, Joseph, and Thaddeus Dryja. The BitcoinLightning Network: Scalable Off-Chain Instant Payments. (2016)
[10]Popov, Serguei The Tangle. (2018).https://iota.readme.io/docs/whitepaper
[11]Baird, Leemon, Mance Harmon, and Paul Madsen. Hedera: A Governing Council & Public Hashgraph Network. (2018)
[12]Liu, Chunming, Daniel Wang, and MingWu. Vite: A High Performance

AsynchronousDecentralized Application Platform
(2018).https://www.vite.org/whitepaper/vite_en.pdf
[13]Li, Chenxing, Peilun Li, Wei Xu, Fan Long, andAndrew Chi-chih Yao. Scaling Nakamoto Consensusto Thousands of Transactions per Second. (2018)
[14]The Zilliqa Team. The Zilliqa Technical Whitepaper.(2017), https://docs.zilliqa.com/whitepaper.pdf
[15]The Ethereum Foundation. Sharding Roadmap.https://github.com/ethereum/wiki/wiki/Shardingr oadmap
[16]Micali, Silvio, Michael Rabin, and Salil Vadhan.Verifiable Random Functions. In Foundations of Computer Science, 1999. 40th Annual Symposiumon, pp. 120-130. IEEE, 1999.
[17]Dodis, Yevgeniy, and Aleksandr Yampolskiy. A Verifiable Random Function with Short Proofs and Keys.In International Workshop on Public Key Cryptography, pp. 416-431. Springer, Berlin, Heidelberg,2005.
[18]Hanke, Timo, Mahnush Movahedi, and DominicWilliams. DFINITY Technology Overview Series, Consensus System. (2018)
[19]Gilad, Yossi, Rotem Hemo, Silvio Micali, GeorgiosVlachos, and Nickolai Zeldovich. Algorand: ScalingByzantine Agreements For Cryptocurrencies. (2017)
[20]David, Bernardo Machado, Peter Gazi, AggelosKiayias, and Alexander Russell. Ouroboros Praos:An Adaptively-Secure, Semi-Synchronous Proof-Of Stake Protocol. IACR Cryptology ePrint Archive2017 (2017): 5
[21]Daniel J. Bernstein, Niels Duif, Tanja Lange, Peter Schwabe, and Bo-Yin Yang High-speed high-security signatures Journal of Cryptographic Engineering 2 (2012), 77-89. https://cr.yp.to/papers.htmled25519
[22]Castro, Miguel, and Barbara Liskov. Practical Byzantine Fault Tolerance. In OSDI, vol. 99, pp.173-186. 1999.
[23]Buchman, Ethan, Jae Kwon, and Zarko Milosevic. The Latest Gossip on BFT Consensus. (2018)
[24]de Vries, Alex. Bitcoin’s Growing Energy Problem. Joule 2, no. 5 (2018): 801-805.
[25]Back, Adam. Hashcash-A Denial of Service Counter Measure. (2002)