DOI:10.7544/issn1000-1239.2020.20190552
57(7): 1555-1567, 2020

一种面向公有链的轻量级可扩展技术

陈 幻 王意洁
(并行与分布处理国家重点实验室(国防科技大学) 长沙 410073)
(国防科技大学计算机学院 长沙 410073)
(chenhuan245@gmail.com)

A Lightweight Scalable Protocol for Public Blockchain

Chen Huan and Wang Yijie
(*National Laboratory for Parallel and Distributed Processing* (*National University of Defense Technology*), *Changsha* 410073)
(*College of Computer*, *National University of Defense Technology*, *Changsha* 410073)

Abstract Blockchain technology solves the fundamental problem of building trust in an untrusted environment, which is regarded as a new disruptive technology after clouding computing, the IoT, and the artificial intelligence. However, the current public blockchains face two serious problems: on one hand, the low system throughput cannot meet the needs of large-scale applications; on the other hand, the ever-growing nature of blockchain could be quite cumbersome for validators since they consume much disk and RAM resources. Existing works were often used to improve system throughput, ignoring the increasingly serious problem of blockchain data growth for long run. Thus, in this work, we propose the PocketChain, a scalable and storage-friendly lightweight protocol that can achieve high throughput and low storage without sacrificing decentralization and security. Firstly, PocketChain uses a stateless client design, using the RSA accumulator to combine the large state into one short commitment, so that the validators can only store the block headers, greatly reducing the disk and RAM requirements. Secondly, PocketChain applies the stateless client to the sharding technology, which not only improves the system throughput, but overcomes the state migration problem caused by periodical reshuffling of sharding technology. This can further increase the security of sharding by improving the reshuffling frequency. The experiment results show that the PocketChain can reduce the storage overhead of validators and linearly improve the system throughput.

Key words public blockchain; sharding; RSA accumulator; stateless client; unspent transaction outputs (UTXO)

摘 要 区块链技术解决了在不可信环境下建立信任的基础难题,被视为继云计算、物联网和人工智能之后的又一项颠覆性技术.然而,目前公有链面临2大根本难题:1)较低的系统吞吐率无法满足大规模

收稿日期:2019-08-15;**修回日期**:2019-12-27
基金项目:国家重点研发计划项目(2016YFB1000101);国家自然科学基金项目(61379052);国家教育部科研创新基金项目(2018A02002);湖南省自然科学杰出青年基金项目(14JJ1026)
This work was supported by the National Key Research and Development Program of China (2016YFB1000101, the National Natural Science Foundation of China (61379052), the Science Foundation of Ministry of Education of China (2018A02002), and the Natural Science Foundation of Hunan Province of China for Distinguished Young Scholars (14JJ1026).
通信作者:王意洁(wangyijie@nudt.edu.cn)

运用的需求;2)持续增长的账本和状态数据,对节点磁盘和内存容量提出了较高要求.已有的扩容技术往往只针对提升系统吞吐,忽略了区块链数据增长对节点存储资源消耗的严重问题.为此,提出了PocketChain,一种对存储友好的轻量级扩容技术,在不牺牲去中心化与安全性的前提下,实现高吞吐和低存储的特性.首先,针对数据增长问题,PocketChain 采用无状态客户端设计,使用 RSA 累加器对状态进行压缩,使得验证节点只需存储区块头部信息,大大降低节点对磁盘和内存的需求.其次,PocketChain 将无状态客户端运用于分片技术架构下,在提升系统吞吐的同时,克服分片周期性随机重组导致的状态迁移问题,从而能进一步提升分片重组频率,增加分片系统安全性.实验结果表明:该方法能够有效降低节点存储需求,并线性提升系统吞吐.

关键词 公有链;分片;RSA 累加器;无状态客户端;未消费交易输出

中图法分类号 TP393

区块链技术[1-3]被视为继云计算、物联网、人工智能之后的又一项颠覆性技术,受到了金融机构、政府部门以及各科技企业的高度关注,将开启下一代互联网——价值交换网络的新时代.区块链解决了在不可信环境中建立信任的基础难题,具有去中心化、不可篡改、不可抵赖、抗审查等特征,能够广泛运用于支付清算、物联网、政务服务、医疗、物流、博彩娱乐等多种领域.

然而,区块链存在 2 个严重的问题:1) 系统吞吐率低.目前比特币每秒处理交易数量(transactions per second, TPS)在 3～7 笔之间,以太坊 TPS 在 7～15 笔之间.相比之下,主流的中心化支付系统,例如 Visa 和 MasterCard,平均每秒能处理 2 000 笔交易,峰值性能能够达到 56 000 TPS.低吞吐率导致网络中未处理的交易无限堆积,交易平均处理时间延长,使得区块链无法运用于实时领域.此外,随着越来越多的分布式应用(distributed applications, DApps)运行于同一条链上,低吞吐率的问题将会越发严重,极大地限制了区块链的运用.2)每个验证节点都需要独立保存一份完整的历史数据账本和对应的状态信息,这将消耗大量的存储资源.一方面,账本大小随着新区块的产生持续增长,存储全部历史区块需要消耗大量的磁盘空间.可以预测在不久的将来,区块链账本数据将轻松到达 1TB,超过目前绝大多数普通商用机的存储能力.另一方面,由于区块链历史数据过于庞大,从区块中检索信息需要消耗大量的时间开销.为了加速对交易和区块的验证,验证节点会在内存中维护一个叫验证状态的索引结构.尽管验证状态相比于历史区块占用的空间较少,但他们需要被快速访问,通常放在内存数据库中(例如 LevelDB),对节点内存容量提出了较高要求.如果验证状态大小超过了节点的内存容量,一部分状态信息需要保存在磁盘或其他二级存储器中.由于二级存储器访问时间较长,容易遭受分布式拒绝服务(distributed denial of service, DDoS)攻击.例如在 2016 年 9 月[4],攻击者利用以太坊 EXTCODESIZE 指令的漏洞,让交易频繁访问放置在硬盘上的状态信息,导致了单个区块的验证时间长达 60 s.

区块链历史数据以及验证状态不断增长的特点,将带来以下问题.首先,由于对磁盘和内存容量的较高要求,只有少量的“超级”节点能够运行完整的区块链全节点,将导致中心化的出现.其次,大量资源较少的节点只能运行轻客户端(例如移动钱包、轻节点),它们依赖全节点提供的查询服务,不能独立验证交易和区块的有效性,容易遭受长城等类型的攻击.此外,对于新加入网络的全节点,它们需要同步从创世区块到最近的所有历史区块,并建立验证状态的索引结构,需要消耗大量的时间开销.

针对系统吞吐率低的问题,研究人员提出了多种扩容技术.然而,绝大多数扩容技术无法完全破除区块链的三角难题,即同时实现安全、去中心化和可扩展.目前,分片技术被广泛视为一种能够解决区块链三角难题的有效方法,通过将交易和状态进行分割,使得不同的分片能够并行处理不相关的交易,线性提升系统的处理能力.然而,为了避免恶意节点集中到某个特定分片发起攻击,需要周期性地将分片节点进行重组.由于分片节点只存储了当前分片的数据,当被划分到其他分片后,需要下载新分片的数据,这个过程被称为状态迁移.状态迁移需要下载大量数据,不仅中断了系统的处理,同时也限制了分片切换的频率.

针对节点存储资源占用高的问题,研究人员首先提出了账本剪枝技术[5],将对于验证交易和区块无用的历史数据进行裁剪,只维护验证状态信息.该方法可以有效降低整个区块链的账本大小和历史区块同步问题.然而,账本剪枝技术只解决了历史数据对于

磁盘的负担,并不能减轻状态对于内存的需求.针对内存占用高的问题,研究人员进一步提出了无状态客户端[6-10]的概念.无状态客户端利用承诺(commitment)技术,将不断增长的状态压缩到固定字节的承诺.但与账本剪枝技术不同的是,节点并不需要保留具体的验证状态.然而,无状态客户端面临3个挑战:

1) 证明信息过大,增加网络带宽开销.基于默克尔树(Merkle)构建的承诺需要提交从叶子节点到根节点的Merkle路径,以目前比特币未消费交易输出(unspent transaction outputs, UTXO)集合和以太坊账户的规模,构建的Merkle证明分别是交易大小的4倍与20倍.

2) 承诺更新代价高.基于Merkle树的承诺更新,需要已修改节点的所有数据;而基于通用累加器的承诺更新,删除操作需要进行大量的计算.

3) 由于每个证明只对应1个承诺,当承诺更新后,会导致交易有效性证明过期无效.由于网络传输延迟过高,新发起的交易并不能得到及时地处理,交易过期将会成为一个常态问题.让用户频繁提交更新后的证明信息无疑会增加用户使用的负担.

基于3个挑战的问题,构建一种可扩展的轻量级区块链,不仅具有较高的系统吞吐率,同时让所有节点只需利用少量的存储资源(包括磁盘和内存),便可以独立验证和打包交易,成为了加密货币领域一个迫切需要解决的难题.为此,本文提出了PocketChain,一种面向UTXO模型的公有链的轻量级扩容技术.首先,PocketChain使用无状态客户端设计,创新性地提出了基于RSA累加器构建的已消费交易输出(spent transaction outputs, STO)承诺技术,极大提升了状态累加器的更新效率,使得验证节点只需存储区块头便可以验证交易的有效性.其次,将本文提出的无状态客户端与分片技术相集合,利用无状态客户端低存储的优势,克服分片周期性随机重组导致的状态迁移问题,从而能进一步提升分片重组频率,增加分片系统安全性.

综上所述,本文聚焦区块链数据膨胀和低吞吐的问题,提出了PocketChain,一种面向UTXO模型的可扩展轻量级区块链.本文的主要贡献有4个方面:

1) 提出了基于RSA累加器构造的具有批处理能力的增量STO承诺技术,将状态累加器更新效率从$O(n^2)$提升到$O(n)$;

2) PocketChain使用STO承诺技术构建无状态客户端,验证节点只需要保存历史区块头以及少量的缓存数据,有效降低了对磁盘和内存的负担;

3) PocketChain将无状态客户端与分片技术进行结合,避免了分片周期性随机重组导致的状态迁移问题,从而能进一步提升分片重组频率,增加了分片系统的安全性;

4) 实验结果表明:在不牺牲去中心化与安全性的前提下,PocketChain能够极大降低节点的存储压力,并且系统吞吐率能够随着分片数量的增加线性增长.

1　相关工作

在本节中,我们主要介绍无状态客户端与分片技术的相关概念以及研究现状.

1.1　无状态客户端

1.1.1　无状态客户端系统结构

定义1. 区块链状态.给定一个数据结构$S_{\text{state}}=\{s_1,s_2,\cdots,s_n\}$,表示系统中当前所有账户(或地址)的快照.节点可以通过访问S_{state}来验证交易的合法性,从而避免遍历所有历史区块.这样的数据结构称为区块链状态.

需要注意的是,状态信息对于区块链完整性并不是必须的.由于状态信息相比历史区块链占用的存储空间较少,可以放入内存数据库中,从而提高检索效率,对于加速交易的验证具有十分重要的作用.根据验证节点是否保存完整的状态信息,区块链可以分为2类:状态区块链与无状态区块链.

状态区块链的系统结构如图1所示,每个节点根据历史区块建立状态信息,并用状态信息来验证交易的合法性.当新区块产生后,根据区块中的具体交易,对状态进行更新.状态区块链的优点在于,用户不需要保存任何状态就可以发起交易.然而,状态区块链存在状态"爆炸"的隐患.随着有效账户(或地

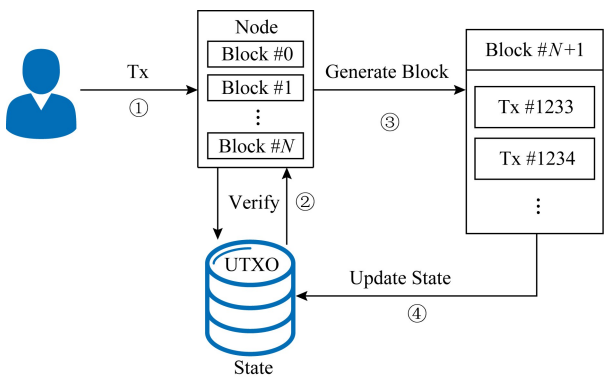


Fig. 1　Stateful blockchain architecture

图1　状态区块链系统结构

址)的增加,状态空间不断增长,一方面需要节点不断提升内存容量,另一方面会降低验证交易和区块的效率.

相比之下,无状态区块链只需要维护状态的承诺,不需要保存任何具体的状态信息.如图 2 所示,每个区块头包含当前状态的承诺,每个节点只需要保存历史区块头,并根据用户提交的交易有效性证明对交易进行验证.无状态区块链的优点在于,不需要保留具体的历史区块和维护状态信息,降低了节点对磁盘和内存的需求,使得配置较低的节点(包括手机等移动设备)能够运行完整的验证节点.其缺点在于,承诺更新效率低下,无法适用于高吞吐的环境下.

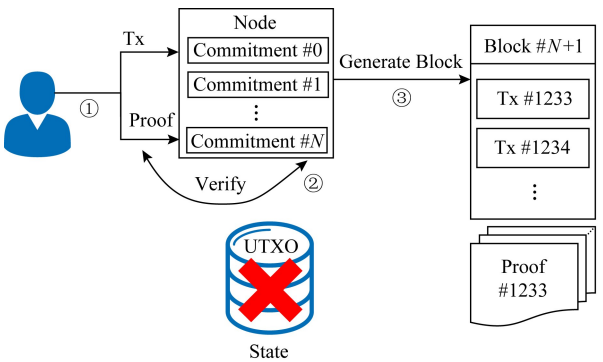


Fig. 2 Stateless blockchain architecture

图 2 无状态区块链系统结构

1.1.2 无状态区块链研究现状

无状态区块链概念首次由以太坊创始人 Buterin[7] 提出,用于解决以太坊状态爆炸问题.使用默克尔帕特里夏树(Merkle patricia tree, MPT)存储所有用户账户的状态信息.矿工节点在打包区块时,需要负责生成相关账户状态的 Merkle 证明,并附加在每个区块的后面.全节点只需要存储状态树根,便可以验证区块.然而,每笔交易的默克尔证明的字节大小是交易的 20 倍,极大地增加了网络带宽开销.

Todd[9] 基于默克尔山脉(Merkle mountain range, MMR)提出了交易输出(transactions outputs, TXO)承诺技术来解决比特币 UTXO 无限增长的问题.它将旧的 UTXO 进行归档,允许全节点丢弃归档的具体数据.当交易消费已归档 UTXO 时,全节点(或服务节点)生成对应的默克尔分支.验证节点根据默克尔分支对 MMR 进行重建,从而能够对旧的 UTXO 进行验证,并相应更新 MMR.证明 MMR 中的任意状态,可以仅使用大小为 lb n 的证明.然而,正如作者提到的,检查承诺是否被正确更新会产生较大的开销.

Reyzin 等人[8] 提出使用加密认证数据结构来提升交易验证效率.矿工节点需要持有完整的状态,在处理交易时对状态进行更新,并发布每笔交易导致状态改变的证明.相比之下,验证节点仅需要持有状态的简短摘要,验证证明并计算出新摘要.由于验证节点不需要存储具体的状态信息,使得配置较低的节点依然能够独立验证交易,提升了系统的安全.

Chepurnoy 等人[6] 提出了一种无状态交易验证的通用框架,称为 EDRAX.所有节点,包括矿工和验证节点只需持有最新被确认的区块,便可以验证交易并更新状态.他们提供了 EDRAX 的 2 种实例:一种使用稀疏默克尔树(sparse Merkle tree, SMT)构建基于 UTXO 模型的区块链;另一种使用代数向量承诺(vector commitment, VC)构建基于账户模型的区块链.然而,EDRAX 并没有考虑证明过期问题导致用户频繁提交更新后的证明,增加了用户使用的负担.

Boneh 等人[10] 首先针对通用累加器和向量承诺,提出了多种批处理技术,极大地提升了累加器的验证效率并降低了传输开销.其次,使用 Class Group 未知阶群构建无陷门的通用 RSA 累加器,避免进行可信的初始化设置.最后,将以上具有批处理能力、无陷门特征的 RSA 累加器用于创建基于 UTXO 模型的无状态公有链,极大地降低了交易的证明大小和验证开销.然而,在不知道群阶的情况下,RSA 累加器批量删除操作的复杂度为 $O(n^2)$,效率低下.基于 RSA 累加器构建的 UTXO 承诺,需要进行大量删除操作,导致累加器更新效率低,无法适用于高吞吐的环境下.

1.2 分片技术

1.2.1 区块链分片技术基础模型

分片技术在运用于区块链之前,已经在数据库领域得到了广泛运用,用于提升数据库的吞吐能力,例如 OpenDHT[11], Google spanner[12] 等.简单来说,分片技术将数据库的不同表(或相同表中的不同行)分别放置在不同的服务器中,根据数据访问请求的对象,将请求分发到对应的服务器.从而,数据库的响应能力能随着服务器数量的增加呈线性增长,极大地提升了数据库的服务能力.类似地,在区块链中,可以将单链拆分多条并行运行的子链(分片).从实现的角度,分片可以进一步分为交易分片和状态分片.交易分片根据特定规则,将网络中的交易分发到不同的子链,各子链并行验证和打包交易,从而使区块链的吞吐量实现质的提升.状态分片将区块链的验证状态(广义上来讲,包括历史数据)分为多个

不相交的子集,各分片只需保存其中的一个状态子集和处理与状态相关的交易,从而降低每个节点的数据存储量,如图 3 所示.实现状态分片往往意味着同时实现交易分片.

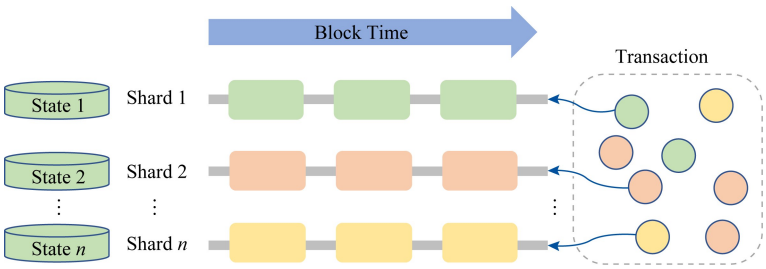


Fig. 3　State sharding architecture

图 3　状态分片架构

1.2.2　区块链分片技术研究现状

Elastico[13]是最早面向公有链的交易分片技术.首先,各节点需要进行第 1 轮工作量证明(proof of work, PoW)建立身份,避免遭受女巫攻击.为了防止恶意节点集中到特定分片中,利用绝对的算力(或权益)对该分片进行攻击(称为 1%攻击),需要将节点随机地分配到各个分片.分片形成后,分片内部采用 PBFT[14]协议达成共识,形成 micro block 发往 0 号分片.0 号分片通过验证和打包 mirco block,再次运行 PBFT 共识,生成最终区块.然而,Elastico 不支持跨片交易处理,并且分片内部采用传统的 PBFT 协议,参与共识的节点数量较少,具有较高的错误率.此外,Elastico 为了避免 1%攻击,需要周期性地对节点进行重新分配.为了避免每次分片切换时下载大量新分片的数据,各节点需要保存所有分片的数据,节点存储压力较大.

Zillqa 团队[15]在 Elastico 的基础上,进行了架构和性能优化.在设计架构上,提出双链架构,一条是交易链,一条是目录服务链.交易链上保存交易数据,目录服务链存放矿工元数据信息.其中最近的 c 个节点形成最终委员会,负责划分分片以及收集各分片的 micro block 形成 final block.在共识方面,Zillqa 借鉴了 CoSi[16]多重签名技术,并用数字签名代替 MAC,极大提升了 PBFT 的扩展性,使得 PBFT 可以适用于几百个节点.此外,Zillqa 第 1 次创新性地提出数据流编程框架,利用全网算力来实现大规模计算,如 MapReduce 和神经网络计算等.虽然 Zilliqa 在共识方面有较大改善,但是依然没有实现状态分片.

OmniLedger[5]首次实现了状态分片并提出了并行共识算法.通过移除块的全序要求,将块组织成有向无环图(directed acyclic graph, DAG)结构,增加了系统的吞吐量、降低了交易确认延迟.此外,OmniLedger 使用原子提交协议来处理跨分片交易;并使用账本剪枝技术,引入检查点,将检查点之前的历史数据进行裁剪,降低节点的存储压力.然而,原子提交协议采用以用户为中心的 2 阶段提交方法,需要用户参与跨分片交易处理;每次分片切换,都会导致状态迁移.

RapidChain[17]在 OmniLedger 的基础上,加入了基于纠删码[18-20]的信息分发技术来加快大区块的传播速度,实现了覆盖通信、计算与存储的全分片技术.为了降低状态迁移代价,RapidChain 采用了 Cuckoo 协议,每次分片切换只需替换部分节点.

以太坊 2.0[21]在阶段 0 将引入基于权益证明(proof of stake, PoS)的信标链(beacon chain),完成从 PoW 到 PoS 共识机制的转变.在阶段 1 将引入分片,分片是以太坊网络未来扩容技术的核心,允许同时进行交易、存储和信息处理,从而线性地增加整体网络的吞吐量.根据以太坊 2.0 规范,信标链将支持 1024 个分片链,每条分片链将有 128 个节点进行验证工作.信标链的关键功能是管理 PoS 协议以及所有的分片链:在每一步为每个分片随机选择区块的提议者,组织验证者进入委员会,对拟议的区块进行投票;对验证者实施奖励和处罚;执行交联(crooslinks)的处理,将整个分片系统连接在一起,并协助完成跨片交易处理.整体而言,以太坊 2.0 分片技术较为完善,能够同时实现交易分片和状态分片,但每个分片依然会面临数据与状态增长问题.

2　系统简介

2.1　主要思想

针对目前公有链系统吞吐率低下和节点存储压

力不断增加的难题,本文提出了 PocketChain,它需要满足 3 个目标:

1) 低存储.节点不需要保留具体的历史区块和状态信息便可以验证交易的有效性,大大降低运行验证节点的存储资源开销.

2) 高吞吐.系统吞吐率能够随着网络节点数量的增长线性增加.

3) 安全与去中心化.系统不牺牲任何安全性和去中心化的特性.

2.2 挑 战

为了降低验证节点的存储开销,已有的账本剪枝技术只能降低历史账本大小.随着系统中有效地址和账户的不断增加,节点内存开销逐渐加大.无状态客户端采用累加器将状态数据压缩到固定字节的承诺,能够同时解决历史区块与状态数据膨胀问题,但面临交易证明信息过大(基于 Merkle 的累加器)、承诺更新代价高(通用累加器)的挑战.

此外,分片技术被广泛视为提升区块链吞吐率最有效的方法之一,已被多个公有链采用.然而,为了避免恶意节点集中到某个特定分片发起定向攻击,需要周期性地将节点随机分配到各个分片中.由于每个分片只保留了当前分片的数据,当节点划分到其他分片后,需要同步下载新分片的数据,这个过程称为状态迁移.状态迁移需要下载大量数据,不仅增加了网络带宽开销,同时限制了分片切换的频率,对系统安全性造成负面影响.

综上所述,本文将结合无状态客户端与分片技术来解决区块链数据增长与吞吐率低下的双重难题,其面临的主要挑战有 2 点:1)构造证明信息短小、更新效率高的状态累加器.2)克服分片技术周期性随机重组导致的状态迁移问题.

2.3 技术路线

无状态客户端的关键在于构造证明信息短小、更新效率高的状态累加器.RSA 累加器建立在冥模运算之上,具有证明信息短小、固定等优点,能够满足以上需求.此外,RSA 累加器支持批处理操作,多个证明可以合并为一个证明,能有效减少交易证明的字节大小,并降低验证的时间开销.在公有链环境下,没有任何可信的第三方,需要借助未知阶群,构造无陷门的 RSA 累计器.然而,在不知道群阶的情况下,RSA 累加器删除操作的复杂度为 $O(n^2)$,添加操作的复杂度为 $O(n)$.当需要进行大量删除操作时,累加器的更新效率将快速下降.例如 UTXO 承诺技术需要动态删除大量已消费的交易输出,导致 UTXO 承诺更新效率低下.

针对以上问题,在 3.2 节中,本文首先创新性地提出了 STO 承诺技术,使用 RSA 累加器将所有已消费的交易输出进行压缩.STO 是一个只支持添加操作的数据结构,从而避免进行代价较高的删除操作.接着,PocketChain 基于 STO 承诺技术建构无状态客户端.用户为了证明交易有效,需要提供交易输入的存在性证明和交易输入的未花费证明.输入存在性证明可以通过输入产生时的 Merkle 路径进行自证;输入的未花费证明可以通过提交 STO 非成员见证.只要输入不在 STO 集合内,便不存在双花.最后,在第 4 节中将无状态客户端运用于分片技术架构下,利用低存储的优势,降低分片技术周期性随机重组导致的状态迁移代价.

3 基于 RSA 累加器的无状态客户端

无状态客户端设计将交易验证的一部分工作分配给用户,用户在发起交易时,需要附加提交交易有效性证明.验证节点和矿工节点只需保存状态的简短摘要,根据交易有效性证明和当前状态摘要进行交易验证.无状态客户端不需要存储具体区块和状态信息,大大降低了节点的磁盘和内存压力.然而,目前无状态客户端主要面临交易有效性证明过大、状态累加器更新效率低,以及交易频繁过期问题.

为此,本节针对 UTXO 模型的公有链,提出了基于 RSA 累加器的无状态客户端 PocketChain.首先,使用无陷门、具有批处理能力的 RSA 累加器压缩状态,使得证明信息简洁短小,并且能将多个证明合并为一个证明,有效降低了证明的字节大小;将 RSA 累加器作用于 STO 集合,STO 是一个只支持添加操作的数据结构,避免了进行代价较高的 RSA 累加器的删除操作,大大提升了累加器的更新效率;最后,引入缓存,使得交易可以在接下来的 n 个区块时间内被处理,避免每次新区块产生后都需要重新提交更新后的有效性证明.

3.1 累加器

1993 年累加器(accumulator)首次被 Benaloh 等人[22]提出,能够将一个大集合的数据压缩到一个短小的、固定字节的值,称为累加值.对于每一个在集合内的数据,都存在一个成员见证(membership witness)证明该数据在集合内.对于不在集合内的数据,无法伪造成员见证.Camenisch 等人[23]进一步提出了动态累加器(dynamic accumulator),能够高效地向累加值添加或删除指定元素.在动态累加器的

基础上,Li 等人[24]提出了通用累计器(universal accumulator),不仅支持成员见证,对于不在集合内数据,也能够生成非成员见证(non-membership witness),证明该数据不在集合内.目前,累加器已广泛运用到多个领域,包括成员测试、数据外包时的隐私保护、匿名电子现金系统等.

RSA 累加器是在强 RSA 的假设下,基于 RSA 模数的模幂运算.其最简单的形式工作为:累加器的秘钥是 RSA 的模数 $N=pq$ 的因子,其中 p 和 q 为强素数,累加值初始化为 $g\in\mathbb{Z}_N$.对于任意大小的素数集合 $S=\{x_1,x_2,\cdots,x_n\}$,其累加值为

$$A(S)=g^{x_1\times x_2\times\cdots\times x_n} \bmod N. \tag{1}$$

对于任何在集合 S 中的数据 x_i,其成员见证 w_i 为 S 集合中除去 x_i 的累加值:

$$w_i=g^{x_1\times x_2\times\cdots\times x_{n-1}\times x_{n+1}\times x_n} \bmod N. \tag{2}$$

验证成员见证 w_i 是否正确可以通过判断:

$$w_i^{x_i}=A(S) \bmod N. \tag{3}$$

是否成立.

传统的 RSA 累加器在初始化时需要生成 2 个强素数 p 和 q,任何知道素数 p 和 q 的人都可以使用欧几里得定理 $\varphi(N)=(p-1)(q-1)$ 计算 RSA 群的阶,从而可以进一步伪造任何成员证明和非成员证明.然而,在公有链环境下,不存在任何可信第三方,如何确保 RSA 累加器秘钥不泄露是需要解决的首要问题.这里我们借鉴了 Boneh 等人[10]的工作,他将动态累加器建立在未知阶的群之上,避免进行可信的初始化设置.Class Group 作为一种虚构的二次方阶群,可以用于构建去信任化的 RSA 累加器.相关工作本文不做展开,读者可以参考 Boneh 以及其他相关工作.

3.2　STO 承诺技术

在 UTXO 模型的区块链中,状态由所有的 UTXO 组成.每笔交易消费已有的 UTXO,并产生新的 UTXO 记录.已有的工作将 RSA 累加器作用于 UTXO 集合上,通过构建 UTXO 承诺来压缩状态.当新区块产生后,从 UTXO 承诺中删除所有已消费的 UTXO 并添加新产生的 UTXO.然而,在不知道群阶的情况下,从 RSA 累加器中删除元素的复杂度为 $O(n^2)$,而添加操作的复杂度仅为 $O(n)$,频繁地从累加器中删除元素将导致累加器更新效率低下.

为此,本文提出了 STO 承诺技术,使用 RSA 累加器将所有 STO 进行压缩.由于 STO 是一个只支持添加操作的数据结构,避免了进行代价较高的累加器删除操作.

3.3　基于 STO 承诺的无状态客户端

PocketChain 基于 STO 承诺技术构建无状态客户端,其系统结构如图 4 所示,主要包含 3 个模块:STO 承诺、证明生成与更新.

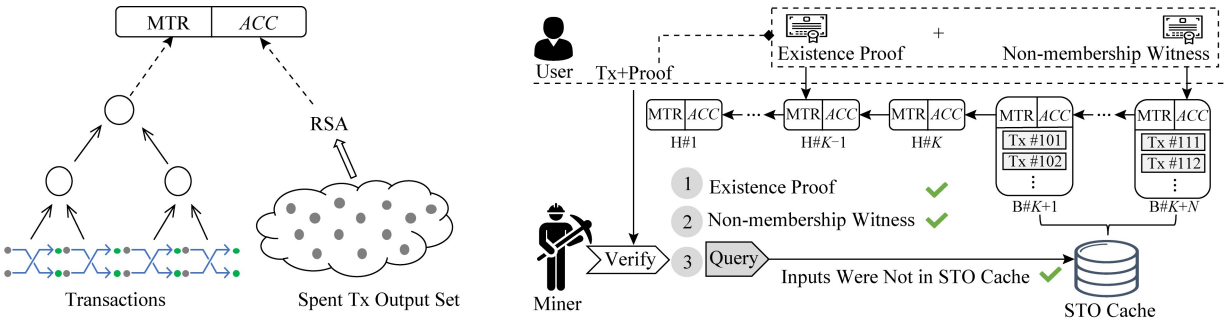


Fig. 4　System architecture of PocketChain

图 4　PocketChain 系统结构

3.3.1　STO 承诺

每个区块头除了包含交易的 Merkle 树根(Merkle transactions root, MTR),还包含了由所有已消费的交易输出组成的 STO 累加值 ACC.当新区块产生后,需要向 ACC 中添加区块中已消费的交易输出,构造新的 STO 累加值.此外,RSA 累加器的输入必须限制为素数才能保证其无碰撞的特性,为此我们需要将 STO 转变为素数,这可以通过函数 $HashToPrime$[25]进行转化.具体过程如算法 1 所示:

算法 1. 状态累加器批处理更新与验证算法.

/ * RSA 累加器初始化函数 * /

Func $GGen(\lambda)$:

① $G_{\mathrm{RSA}}\leftarrow GGen(\lambda)$;

② $A_0\leftarrow G_{\mathrm{RSA}}$;

③ return A_0.

/* 累加器更新函数 */

Func $AccUpdate(A_t, tx)$:

① $p=1$;

② for each u in $tx.inputs$ do

③ $p^*=HashToPrime(u)$;

④ end for

⑤ $A_{t+1}\leftarrow A_t^p$;

⑥ $Q\leftarrow NI\_POE\_PROVE(A_t, p, A_{t+1})$;

⑦ return A_{t+1}, Q.

/* 累加器验证函数 */

Func $AccVerify(A_t, tx, A_{t+1}, Q)$:

① $p=1$;

② for each u in $tx.inputs$ do

③ $p^*=HashToPrime(u)$;

④ end for

⑤ return $NI\_POE\_VERIFY(A_t, p, A_{t+1}, Q)$.

1) $Gen(\lambda)$首先根据系统安全参数 λ 对系统进行初始化,返回累加初始值 A_0;

2) $AccUpdate(A_t, tx)$根据新区块所消耗的交易输出,对累加器进行批处理更新.首先,使用 $HashToPrime$ 对每个交易的输入进行求质运算,并计算它们的乘积,然后进行冥模运算更新累加值.最后使用非交互式 NI-POE 协议提交正确更新的证明 Q.

3) $AccVerify(A_t, tx, A_{t+1}, Q)$根据交易和证明 Q,使用 NI-POE 协议验证累加值是否被正确更新.

其中,NI-POE[10,26]协议允许证明人 P 向验证人 V 证明(u, w, x)满足 $w=u^x$,V 只需进行少量计算,便可以相信 P 进行了正确计算.NI-POE 协议能够很大程度上降低验证累加器被正确更新的代价.

3.3.2 交易有效性证明生成

交易有效性证明需要包括 2 个部分:交易输入的存在性证明和交易输入的未花费证明.其中,输入的存在证明可以使用交易输入产生时的 Merkle 证明,包含从叶子节点到 MTR 的路径.由于不同的交易输入可能产生于不同的区块,所以无法对存在性证明进行合并.

交易输入的未花费证明是对当前 STO 累加值的非成员见证.由于 STO 包含了所有已消费的输入,只要能提供非成员见证,就能证明输入没有被消费.具体过程如算法 2 所示.其中,非成员见证生成算法 $NonMemWitCreate$ 以 UTXO 生成时所在区块的 STO 累加值作为初始值,获取之后所有的 STO 记录进行求质运行并计算乘积,利用通用累加器的非成员见证生成算法进行生成.类似地,$NonMemWitVerify$ 算法对非成员见证进行验证,具体可以参考 Li 等人[24]关于通用累加器的非成员见证生成与验证算法.值得注意的是,交易输入的未花费证明可以被合并.例如 2 个分别产生于区块 t 与区块 $t+1$ 的输入,我们可以构造一个从区块 t 开始的合并证明,因为产生于区块 $t+1$ 的输入并不能提前被消费.

算法 2. 证明生成、更新和验证算法.

/* 针对当前累加值 A_m,生成 x_n 的非成员见证 */

Func $NonMemWitCreate(A_n, STO_{n:m}, A_m, x_n)$:

① $p=1$;

② $x_n\leftarrow HashToPrime(x_n)$;

③ for each sto in $STO_{n:m}$ do

④ $p^*=HashToPrime(sto)$;

⑤ end for

⑥ $a, b\leftarrow Bezout(x_n, p)$;

⑦ $d\leftarrow A_n^a$;

⑧ return $u_m(x_n)\leftarrow(d, b)$.

/* 非成员见证验证算法 */

Func $NonMemWitVerify(A_n, A_m, x_n, u_m(x_n))$:

① $x_n\leftarrow HashToPrime(x_n)$;

② $d, b\leftarrow u_m(x_n)$;

③ check $dx\ A_m^b=A_n$.

/* 将 x_t 的非成员见证从 A_n 更新到 A_m */

Func $NonMemWitUpdate(A_n, u_n(x_t), x, STO_{n:m})$:

① $d, b\leftarrow u_m(x_n)$, $p=1$;

② for each sto in $STO_{n:m}$ do

④ $p^*=HashToPrime(sto)$;

⑤ end for

⑥ $a_0, b_0=Bezout(HashToPrime(x_t), p)$;

⑦ $r=a_0 b$;

⑧ return $u_m(x_t)\leftarrow(dA_n^r, b_0 b)$.

3.3.3 交易有效性证明更新

交易输入的存在性证明使用输入产生时的 Merkle 路径,该证明保持不变,不需要更新.然而,交易输入的未花费证明是对当前 STO 集合累加值的非成员见证,当 STO 累加值更新后,需要同步更新未花费证明.

用户在发起交易时提交的最新有效性证明,可能会因为网络传播延迟或矿工无法及时打包交易,导致交易有效性证明会过期.让用户重复提交更新后的有效性证明无疑会增加用户使用的负担与体验.为此,PocketChain 引入了 STO 缓存机制,每个验证节点保存最近 n 个区块(例如 1 h 或 1 d)的数

据以及对应的 STO 集合,通过验证输入存在性证明与未花费证明后,附加验证输入不在 STO 缓存内,从而允许交易在发起后的 n 个区块时间内被处理,避免用户重复提交证明信息.

对于那些长期没有更新的未花费证明,用户可以选择自己更新或交由服务节点更新.用户自己更新的算法见 *UpdateNonMemWit*,通过获取上次更新之后所有的 STO 记录,逐个区块地进行更新.然而,在计算资源较少的环境下,更新非成员证明较为耗时.在忽略网路传输延迟的情况下,一个 2.6 GHz Intel Core i5 处理核心每秒只能处理 150 个添加操作.因此,我们引入了服务节点,服务节点利用算力为用户更新证明从而获得报酬.

4　基于分片的扩容技术

分片技术的主要思想可以简单描述为分而治之,将单链结构拆分为多条并行运行的子链(分片),通过增加分片数量来提升系统的处理能力.分片技术可以分为交易分片和状态分片.交易分片仅仅实现交易的并行处理,每个节点依然需要保存所有分片的数据.当大量提升系统吞吐后,节点的存储压力极具增大.相比之下,状态分片不仅仅将交易的处理并行化,每个分片只需保存与自己处理交易相关的状态信息,降低了分片的存储压力.然而,为了防止恶意节点集中到某个特定分片发起攻击,目前的分片技术通常建立在周期性随机重组的架构下,每次分片重组都需要重新下载新分片的数据和状态,节点的带宽开销增加,并且严重限制了分片切换的频率.为此,本节将无状态客户端与分片技术相结合,利用无状态客户端低存储的特点,降低分片切换时的状态迁移量.介绍了分片周期性随机重组的系统结构,将分片技术与无状态客户端进行结合,消除分片周期性随机重组导致的状态迁移问题.

4.1　分片周期性随机重组

分片技术将全网算力(或权益)划分为多个部分,每个分片只拥有全网算力的一小部分,为了避免恶意节点集中到某个特定分片发起定向攻击,目前绝大部分基于分片的扩容技术都采用周期性随机重组的架构.如图 5 所示,每个阶段开始,全网节点都会根据上一阶段末产生的新的随机数进行分片重组,节点无法预知它将分配到具体哪个分片中.此外,为了防止分片内节点串通作恶,分片切换的周期越短,系统越安全.

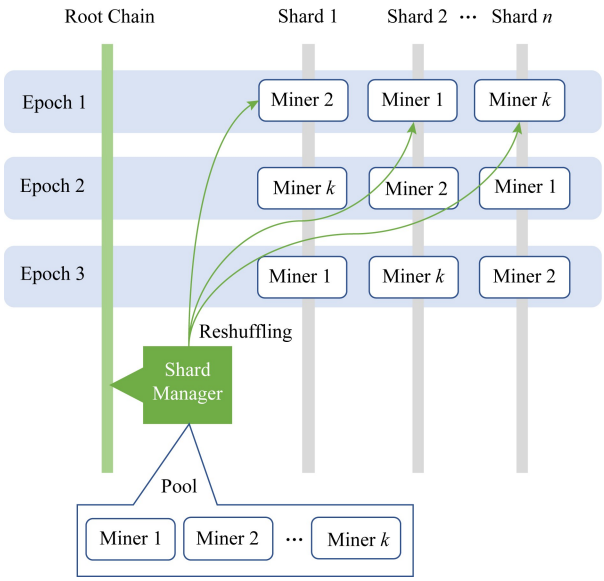


Fig. 5　Periodically reshuffling architecture of sharding

图 5　分片周期性随机重组结构

然而,周期性随机重组的架构面临 2 矛盾:1)在支持状态分片的情况下,每个分片只保留了当前分片的状态与数据,当切换到其他分片后,需要重新下载新分片的状态与数据,这个过程称为状态迁移.状态迁移极大地增加了网络带宽开销,同时也限制了分片切换的频率;2)在不支持状态分片的情况下,节点存储每个分片的状态与数据,虽然不存在状态迁移问题,但节点的存储压力急剧增大.

4.2　无状态客户端降低状态迁移代价

在第 3 节中,我们介绍了无状态客户端,节点只需要保存区块头以及最近 n 个区块的数据,不仅降低了节点存储历史区块的开销,同时将 GB 级的状态压缩到固定的 KB 级,降低了节点对内存的开销.在周期性随机重组的分片架构下,主要面临分片重组时的状态迁移问题,无状态客户端设计能够完美解决以上问题.在支持状态分片的情况下,每次分片切换只需要下载新分片的区块头和少量的缓存数据,大大降低状态迁移代价.此外,由于区块头占用极少的存储空间,验证节点有能力存储所有分片的区块头,在每次分片重组后,节点可以不进行任何状态迁移,可以立即切换到新分片开始工作.

5　实验与结果

在本节中,我们使用本文提供的技术构建了一个轻量级、可扩展的区块链模型 PocketChain,并测试了系统的吞吐量和节点的存储开销.

5.1 实验环境与设置

我们基于 RSA 累加器、Merkle 树以及非成员见证的源码实现了 PocketChain 的无状态客户端.其中,RSA 累加器使用 3 072 b 的模,素数采用 128 b,Merkle 节点为 32 B,这些参数在密码领域被公认为安全的.我们使用 60 个物理节点构建了本地集群,每个节点配置一个 8 核 Intel Xeon E5-1620 3.6 GHz 处理器、48 GB 内存、3 TB 硬盘和 10 Gb/s 的以太网卡.在测试分片的扩展性时,我们使用 docker 模拟了 1 800 个节点.

5.2 RSA 累加器更新效率

本节我们测试了 RSA 累加器在未知秘钥的情况下,针对批量删除和添加操作的更新效率.累加器的更新效率直接影响了系统的吞吐能力.其中,Boneh 使用 RSA 累加器构建 UTXO 承诺,需要进行大量删除操作,而 PocketChain 使用 RSA 累加器构建 STO 承诺,只需要进行添加操作.

图 6 展示了 PocketChain 与 Boneh 累加器的更新效率对比.实验结果表明,PocketChain STO 承诺的更新效率远大于 Boneh UTXO 承诺的更新效率,并且随着操作数量的增加,性能差距越明显.例如,当操作数量达到 1 000 时,Boneh 累加器更新时间是 PocketChain 的 50 倍.这就导致,依赖于 RSA 累加器删除操作的无状态客户端设计面临严重的性能问题.

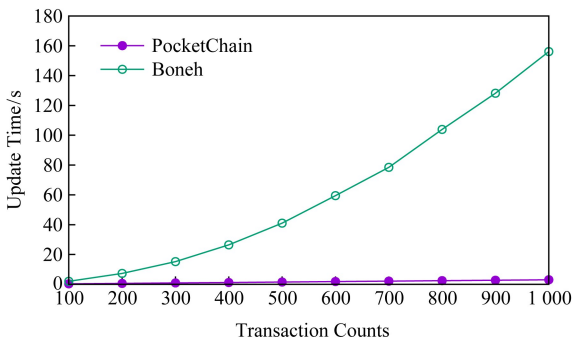


Fig. 6 Performance of accumulator update

图 6 累加器更新效率

我们综合考虑累加器更新效率、交易验证时间开销,测试了系统支持的最高吞吐.我们假设每个区块包含 500 笔交易,每笔交易有 2 个输入和 2 个输出.同样,我们与 Boneh 的工作进行了对比.实验结果如表 1 所示,Boneh 由于累加器更新效率低,最高吞吐只能实现 3.2 TPS,相比之下,PocketChain 能实现 100 TPS.

Table 1 A Single Chain Maximum TPS

表 1 单链系统最高吞吐

Accumulator	Tx Verification Time/s	Acc Update Time/s	Maximum TPS
Boneh	0.01	156	3.2
PocketChain	0.01	3.01	100

5.3 单节点存储

在本节中,我们测试了单链环境下节点的存储压力.我们获取了比特币从 2015-01—2019-01 的历史交易数据,并将其运用到 PocketChain 中.此外,我们将 STO 缓存设置为 1 d,足够绝大多数交易被处理.

图 7 展示了 PocketChain 和比特币的内存开销对比.比特币内存开销从 2015—2019 年增加了 4 倍,目前已经到达了 2.7 GB,相比之下,PocketChain 采用无状态客户端设计,每个节点只需要保存 58 MB(最近一天)的 STO 缓存数据.

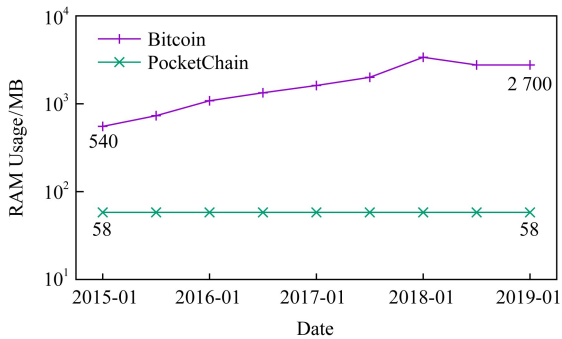


Fig. 7 RAM usage comparison between Bitcoin and PocketChain

图 7 比特币与 PocketChain 内存使用量对比

图 8 展示了 PocketChain 与比特币硬盘使用开销对比.比特币需要保存所有历史区块的数据,目前已经占用了 233 GB 的硬盘存储空间.相比之下,

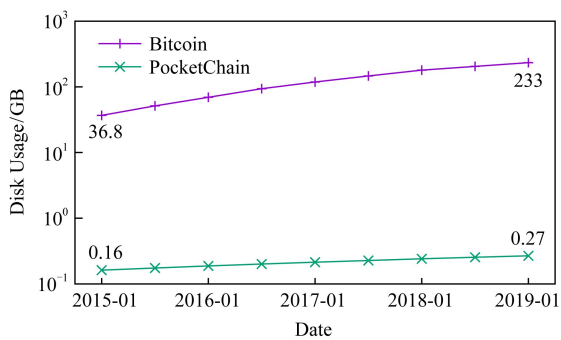


Fig. 8 Disk usage comparison between Bitcoin and PocketChain

图 8 比特币与 PocketChain 磁盘使用量对比

PocketChain 只需要保存区块头,增长速度缓慢,目前仅需 0.27 GB 的存储空间.

5.4　分片扩展能力

本节我们首先测试了分片的扩展能力,然后与最新的分片工作进行了对比,结果如表 2 所示.其中:k 表示分片数量.

在 TPS 方面,所有的工作都能线性提升系统吞吐.在磁盘消耗方面,OmniLedger 采用基于检查点账本剪枝技术,只需存储从上一个检查点到最近的历史区块,能够大大降低磁盘存储空间;相比之下,PocketChain 仅需要存储所有历史区块头,由于区块头占用极少的存储空间,依然能有效降低磁盘存储.

Table 2　Comparison of PocketChain with State-of-the-art Sharding-based Blockchain Protocols

表 2　PocketChain 与最新的分片协议的对比

Protocol	TPS	Disk	RAM	State Sharding	Data Migration
Elastico[13]	Linearly	All History Block	All State	Unsurport	
Zilliqa[15]	Linearly	All History Block	All State	Unsurport	
OmniLedger[5]	Linearly	Ledger Pruning	$1/k$ State	Surport	$1/k$ State
RapidChain[17]	Linearly	All History Block	$1/k$ State	Surport	$1/k$ State
SSChain[27]	Linearly	$1/k$ History Block	$1/k$ State	Surport	0
PocketChain	Linearly	Block Header	Constant Sized	Surport	0

在内存的消耗方面,OmniLedger,RapidChain,SSChain 支持状态分片,只需存储 $1/k$ 的状态,随着时间的增长,状态空间依然保持增长.相比之下,PocketChain 使用无状态客户端,节点只需保存最近 n 个区块的状态信息(MB 级),内存消耗为常数.

在状态迁移方面,OmniLedger 和 RapidChain 在分片切换时,都需要下载大量状态信息;SSChain 在双层架构下,利用经济激励机制避免了分片重组.PocketChain 可以存储所有分片链的区块头,分片切换时不需要下载额外数据;在没有保存其他分片链的情况下,也只需要同步区块头,大大降低同步带宽和时间开销.

5.5　去中心化与安全性分析

PocketChain 采用无状态客户端来降低节点存储压力,节点仅需存储历史区块头与最近 n 个区块,硬盘消耗增长缓慢,内存开销固定,大大降低了运行一个矿工节点和验证节点的存储成本,去中心化程度得到提升.相比之下,目前的公有链,例如比特币和以太坊,硬盘和内存资源消耗随着时间快速增长,运行矿工节点和验证节点的成本不断增加,可以预测在不久的将来,只有少数“超级”节点能够完成交易的验证,系统中心化程度逐渐增加.

在安全性方面,使用 Class Group 构建 RSA 累加器,避免了可信第三方的初始化设置.此外,分片重组频率与分片系统的安全性密切相关,在贿赂攻击模型下,恶意节点可以通过贿赂改变任何诚实节点的行为.假设全网节点数量为 m,分片数量为 n,恶意节点数量为 k,分片切换周期为 x.在随机分组下,每个分片初始分配的恶意节点数量为 k/n.假设攻击者平均每 t 时间成功贿赂 1 个诚实节点,那么必须保证:

$$\frac{k}{n}+\frac{x}{t}\leqslant\frac{1}{2}\times\frac{m}{n},$$

即 $x\leqslant\frac{m-2k}{2n}t$ 才能保证系统安全.

PocketChain 将无状态客户端运用于分片技术架构下,由于无状态客户端低存储的特点,降低了分片周期性随机重组导致的状态迁移代价,从而可以提升分片重组频率,进一步增加分片系统的安全性.

6　结论与下一步工作

本文聚焦于公有链低吞吐和数据增长问题,提出了一种面向 UTXO 模型的轻量级、可扩展的公有链 PocketChain.首先,PocketChain 使用具有批处理能力的 RSA 累加器构建无状态客户端,验证节点只需保存历史区块头和最近的 n 个区块,避免存储大量历史区块和状态信息.然后,将无状态客户端运用于分片技术架构下,一方面通过增加分片数量,能大大提升系统的处理能力;另一方面,由于无状态客户端低存储的特点,克服了分片周期性随机重组导致的状态迁移问题,从而能进一步提升分片重组频率,增加分片系统安全性.实验结果表明,本文提出的 PocketChain 能有效解决验证节点存储增长问题,并降低分片技术周期性随机重组导致的状态迁移代价.

本文提出的将无状态客户端与分片技术相结合的方法,还遗留 2 个问题有待进一步研究:1)针对服

务提供者提出恰当的经济激励机制.对于那些长久没有更新未消费证明的用户,他们依赖服务提供者的计算服务,如何激励服务提供者为用户服务是一个有待解决的经济问题;2)进一步降低见证更新频率.基于 RSA 累加器的状态压缩方法,每次累加值更新后,对应的(非)成员见证都需要同步更新.虽然 PocketChain 引入了缓存机制,仅仅避免了在交易没被及时处理的情况下频繁更新见证的问题.如何设计更优秀的累加器、降低见证的更新频率,是一个非常值得研究的方向.

参 考 文 献

[1] Swan M. Blockchain: Blueprint for a New Economy [M]. Sebastopol: O'Reilly, 2015: 1-130

[2] Nakamoto S. Bitcoin: A peer-to-peer electronic cash system [EB/OL]. (2008-08-18) [2019-08-11]. http://bitcoin.org/bitcoin.pdf

[3] Wood G. Ethereum: A secure decentralised generalised transaction ledger [EB/OL]. (2017-04-12) [2019-08-11]. https://gavwood.com/paper.pdf

[4] Buterin V. Transaction spam attack: Next steps [EB/OL]. (2016-09-06) [2019-08-11]. https://blog.ethereum.org/2016/09/22/transaction-spam-attack-next-steps/

[5] Kokoris-Kogias E, Jovanovic P, Gasser L, et al. OmniLedger: A secure, scale-out, decentralized ledger via sharding [C] //Proc of the 39th IEEE Symp on Security and Privacy. Piscataway, NJ: IEEE, 2018: 583-598

[6] Chepurnoy A, Papamanthou C, Zhang Yupeng. Edrax: A cryptocurrency with stateless transaction validation[EB/OL]. (2018-10-14)[2019-08-11]. http://Eprint.iacr.org/2018/968.pdf

[7] Buterin V. The stateless client concept [EB/OL]. (2017-10-01) [2019-08-11]. https://ethresear.ch/t/the-stateless-client-concept/172

[8] Reyzin L, Meshkov D, Chepurnoy A, et al. Improving authenticated dynamic dictionaries, with applications to cryptocurrencies [C] //Proc of the 21st Int Conf on Financial Cryptography and Data Security. Berlin: Springer, 2017: 376-392

[9] Todd P. Making UTXO set growth irrelevant with low-latency delayed txo-commitments [EB/OL]. (2016-05-17) [2019-08-11]. https://petertodd.org/2016/delayed-txo-commitments

[10] Boneh D, Bünz B, Fisch B. Batching techniques for accumulators with applications to IOPS and stateless blockchains [C] //Proc of the 39th Annual Int Cryptology Conf. Berlin: Springer, 2019: 561-586

[11] Rhea S, Godfrey B, Karp B, et al. OpenDHT: A public DHT service and its uses [J]. ACM SIGCOMM Computer Communication Review, 2005, 35(4): 73-84

[12] Corbett J C, Dean J, Epstein M, et al. Spanner: Google's globally distributed database [J]. ACM Transactions on Computer Systems, 2013, 31(3): 73-91

[13] Luu L, Narayanan V, Zheng Chaodong, et al. A secure sharding protocol for open blockchains [C] //Proc of the 26th ACM SIGSAC Conf on Computer and Communications Security. New York: ACM, 2016: 17-30

[14] Vukolić M. The quest for scalable blockchain fabric: Proof-of-work vs BFT replication [C] //Proc of Int Workshop on Open Problems in Network Security. Berlin: Springer, 2015: 112-125

[15] Zilliqa Team. The zilliqa technical whitepaper [EB/OL]. (2017-08-10) [2019-08-11]. https://docs.zilliqa.com/whitepaper.pdf

[16] Syta E, Tamas I, Visher D, et al. Keeping authorities "honest or bust" with decentralized witness cosigning [C] //Proc of the 37th IEEE Symp on Security and Privacy. Piscataway, NJ: IEEE, 2016: 526-545

[17] Zamani M, Movahedi M, Raykova M. RapidChain: Scaling blockchain via full sharding [C] //Proc of the 2018 ACM SIGSAC Conf on Computer and Communications Security. New York: ACM, 2018: 931-948

[18] Wang Yijie, Li Sikun. Research and performance evaluation of data replication technology in distributed storage systems [J]. Computers & Mathematics with Applications, 2006, 51(11): 1625-1632

[19] Wang Yijie, Pei Xiaoqiang, Ma Xingkong, et al. TA-Update: An adaptive update scheme with tree-structured transmission in erasure-coded storage systems [J]. IEEE Transactions on Parallel and Distributed Systems, 2018, 29(8): 1893-1906

[20] Wang Yijie, Xu Fangliang, Pei Xiaoqiang. Research on erasure code-based fault-tolerant technology for distributed storage [J]. Chinese Journal of Computers, 2017, 40(1): 236-255 (in Chinese)
(王意洁, 许方亮, 裴晓强. 分布式存储中的纠删码容错技术研究[J]. 计算机学报, 2017, 40(1): 236-255)

[21] Buterin V. Ethereum 2.0 sharding-faq [EB/OL]. (2019-04-18) [2019-08-11]. https://github.com/ethereum/wiki/wiki/Sharding-FAQ

[22] Benaloh J, De Mare M. One-way accumulators: A decentralized alternative to digital signatures [C] //Proc of EUROCRYPT'93. Berlin: Springer, 1993: 274-285

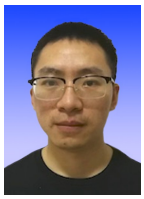
[23] Camenisch J, Lysyanskaya A. Dynamic accumulators and application to efficient revocation of anonymous credentials [C] //Proc of the 22nd Annual Int Cryptology Conf. Berlin: Springer, 2002: 61-76

[24] Li Jiangtao, Li Ninghui, Xue Rui. Universal accumulators with efficient nonmembership proofs [C] //Proc of the 5th Int Conf on Applied Cryptography and Network Security. Berlin: Springer, 2007: 253-269

[25] Fouque P A, Tibouchi M. Close to uniform prime number generation with fewer random bits [C] //Proc of the 5th Int Colloquium on Automata, Languages, and Programming. Berlin: Springer, 2014: 991-1002

[26] Boneh D, Bonneau J, Bünz B, et al. Verifiable delay functions [C] //Proc of the 38th Annual Int Cryptology Conf. Berlin: Springer, 2018: 757-788

[27] Chen Huan, Wang Yijie. SSChain: A full sharding protocol for public blockchain without data migration overhead [J]. Pervasive and Mobile Computing, 2019, 59: 101055



Chen Huan, born in 1991. PhD candidate in the College of Computer of National University of Defense Technology. Received his BS degree in software engineering from the University of Electronic Science and Technology of China, in 2014, and received his MS degree in computer science and technology from the College of Computer of National University of Defense Technology, China, in 2017. His main research interests include blockchain and distributed systems.

Wang Yijie, born in 1971. Professor in the National Key Laboratory for Parallel and Distributed Processing, National University of Defense Technology. Received her PhD degree from the National University of Defense Technology, China in 1998. She was a recipient of the National Excellent Doctoral Dissertation (2001), a recipient of Fok Ying Tong Education Foundation Award for Young Teachers (2006) and a recipient of the Natural Science Foundation for Distinguished Young Scholars of Hunan Province (2010). Her main research interests include distributed storage, big data analysis and cloud computing.