软件学报 ISSN 1000-9825, CODEN RUXUEW
2024,35(11):5279−5305 [doi: 10.13328/j.cnki.jos.007050] [CSTR: 32375.14.jos.007050]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

ElasticDAG: 弹性图式区块链*

岳镜涛[1,2,3,4], 肖 江[1,2,3,4], 张世桀[1,2,3,4], 程 凤[1,2,3,4], 陈汉华[1,2,3,4], 金 海[1,2,3,4]

[1](大数据技术与系统国家地方联合工程研究中心 (华中科技大学), 湖北 武汉 430074)

[2](服务计算技术与系统教育部重点实验室 (华中科技大学), 湖北 武汉 430074)

[3](集群与网格计算湖北省重点实验室 (华中科技大学), 湖北 武汉 430074)

[4](华中科技大学 计算机科学与技术学院, 湖北 武汉 430074)

通信作者: 肖江, E-mail: jiangxiao@hust.edu.cn

摘 要: 图式区块链采用有向无环图 (directed acyclic graph, DAG) 的并行拓扑结构, 相较于基于串行拓扑结构的传统链式区块链, 能够显著提升系统性能, 已受到业界广泛关注. 然而, 现有图式区块链的共识协议与存储模型高度耦合, 缺乏灵活性, 难以适应多元化应用需求. 同时, 大部分图式区块链在共识协议层面上缺乏灵活性, 局限于概率性共识协议, 难以兼顾确认延迟和安全性, 尤其对于延迟敏感型应用很不友好. 为此, 提出弹性图式区块链系统 ElasticDAG, 其核心思想是将存储模型和共识协议进行解耦, 让两者并行、独立地运行, 从而灵活适配多元化应用. 针对提升系统吞吐量和活性的需求, 为存储模型设计自适应区块确认策略和基于划分的确认区块排序算法; 针对降低交易确认延迟的需求, 设计低延迟 DAG 区块链混合共识协议. 实验结果表明, ElasticDAG 原型系统在广域网下的吞吐量高达 11 Mb/s, 并具有 10 秒级确认性能. 与 OHIE 相比, ElasticDAG 在实现同等吞吐量的情况下, 可将确认延迟降低 17 倍; 与 Haootia 相比, ElasticDAG 在实现同等共识延迟的情况下, 可将安全性从 91.04% 提升到 99.999 914%.

关键词: 图式区块链; 混合共识协议; BFT 协议; 存储模型; 弹性区块链系统

中图法分类号: TP393

中文引用格式: 岳镜涛, 肖江, 张世桀, 程凤, 陈汉华, 金海. ElasticDAG: 弹性图式区块链. 软件学报, 2024, 35(11): 5279–5305. http://www.jos.org.cn/1000-9825/7050.htm

英文引用格式: Yue JT, Xiao J, Zhang SJ, Cheng F, Chen HH, Jin H. ElasticDAG: Elastic DAG-based Blockchain. Ruan Jian Xue Bao/Journal of Software, 2024, 35(11): 5279–5305 (in Chinese). http://www.jos.org.cn/1000-9825/7050.htm

ElasticDAG: Elastic DAG-based Blockchain

YUE Jing-Tao[1,2,3,4], XIAO Jiang[1,2,3,4], ZHANG Shi-Jie[1,2,3,4], CHENG Feng[1,2,3,4], CHEN Han-Hua[1,2,3,4], JIN Hai[1,2,3,4]

[1](National Engineering Research Center for Big Data Technology and System (Huazhong University of Science and Technology), Wuhan 430074, China)

[2](Services Computing Technology and System Lab (Huazhong University of Science and Technology), Wuhan 430074, China)

[3](Cluster and Grid Computing Lab (Huazhong University of Science and Technology), Wuhan 430074, China)

[4](School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan 430074, China)

Abstract: A directed acyclic graph (DAG)-based blockchain adopts a parallel topology and can significantly improve system performance compared with conventional chain-based blockchains with a serial topology. As a result, it has attracted wide attention from the industry. However, the storage model and the consensus protocol of the existing DAG-based blockchains are highly coupled, which lacks the flexibility to meet diversified application demands. Furthermore, most DAG-based blockchains lack flexibility at the consensus protocol level and are limited to probabilistic consensus protocols, which is difficult to take into account confirmation latency and security and is

* 基金项目: 国家重点研发计划 (2021YFB2700700); 国家自然科学基金 (62072197); 湖北省重点研发计划 (2021BEA164)
收稿时间: 2022-07-30; 修改时间: 2023-04-28; 采用时间: 2023-09-01; jos 在线出版时间: 2024-01-24
CNKI 网络首发时间: 2024-01-25

especially unfriendly to delay-sensitive applications. Therefore, this study presents the elastic DAG-based blockchain, namely ElasticDAG. The core idea is to decouple the storage model and the consensus protocol, enabling them to proceed in parallel and independently, so as to flexibly adapt to diversified applications. In order to improve the throughput and activity of the system, an adaptive block confirmation strategy and an epoch-based block ordering algorithm are designed for the storage model. In response to the need to reduce transaction confirmation latency, a low-latency DAG blockchain hybrid consensus protocol is designed. Experimental results demonstrate that the ElasticDAG prototype in WAN can achieve a throughput exceeding 11 Mb/s, and it yields a confirmation latency of tens of seconds. Compared with OHIE and Haootia, ElasticDAG can reduce confirmation latency by 17 times and improve security from 91.04% to 99.999 914% while maintaining the same throughput and consensus latency.

Key words: directed acyclic graph (DAG)-based blockchain; hybrid consensus protocol; BFT protocol; storage model; elastic blockchain system

自中本聪于 2008 年发布比特币白皮书[1]以来, 其底层支撑技术区块链被视作下一代价值互联网的重要基础设施[2]. 区块链可帮助缺乏信任基础的参与者在不依赖第三方的情况下达成共识, 实现可信的数据传输和价值传递, 在数字支付、金融服务和供应链管理等领域具有广阔应用前景[3]. 目前区块链已被列为“十四五”七大数字经济重点产业之一, 上升为国家重要发展战略. 调查显示[4], 我国有超过 800 个覆盖金融、社交、文娱的项目正在使用区块链技术. 然而, 区块链技术的应用场景落地仍面临严峻的性能瓶颈. 以数字支付场景为例, VISA 网络平均每秒处理大约 2 000 笔交易, 延迟为数秒钟[5]. 主流区块链系统, 如比特币[1]和以太坊[6], 每秒仅能处理数笔到数十笔交易, 延迟高达几十分钟, 难以满足实际应用需求. 此外, 如去中心化交易所[7]等去中心化应用程序 (decentralized application) 的快速增长也催生出对高性能区块链的巨大需求.

图式区块链采用有向无环图 (directed acyclic graph, DAG) 的并行拓扑结构, 也被称为 DAG 区块链. 相较于基于串行拓扑结构的传统链式区块链, DAG 区块链可支持高并发操作、降低验证计算开销, 从而显著提升吞吐量. 在 DAG 区块链中, 每个区块可以引用 1 个或多个前驱区块, 同时也可被多个后继区块引用, 因此天然支持异步操作和并发操作. 近期已有 IOTA[8]、Spectre[9]、Conflux[10]、Prism[11]、OHIE[12]、DAG-Rider[13]、Narwhal and Tusk[14]、Byteball[15]和 Haootia[16]等工作尝试设计 DAG 区块链以提升系统整体性能.

然而, 本文通过深入分析发现现有 DAG 区块链仍面临两个重要挑战. 一方面, 现有 DAG 区块链在架构层面上缺乏弹性, 难以灵活地根据多元化应用需求进行定制和调整. 其根本原因在于共识协议和存储模型高度耦合, 特别是共识协议与 DAG 拓扑、数据单元类型等存在着复杂的相互作用关系, 任何修改都可能对整个系统造成巨大影响, 导致 DAG 区块链的设计和开发代价高昂. 另一方面, 在公有链场景下, 现有 DAG 区块链采用概率性共识协议, 缺乏灵活性, 无法兼顾安全性和确认延迟[5,17], 不利于延迟敏感型应用. 以 OHIE 为例, 为了保障高安全性, 其确认延迟超过 10 min.

为了应对上述两个挑战, 一方面, 本文对 DAG 区块链的存储模型和共识机制进行解耦, 实现了弹性 DAG 区块链系统 ElasticDAG. 另一方面, 本文提出了一个低延迟 DAG 区块链混合共识协议, 使得公有 DAG 区块链可兼容现有确定性共识协议, 提高了 DAG 区块链在共识协议层面上的灵活性, 并有效降低了确认延迟.

本文的主要贡献和创新总结如下.

(1) 本文提出了弹性 DAG 区块链系统 ElasticDAG. ElasticDAG 将 DAG 区块链解耦为存储模型和共识机制, 一方面使得共识可与区块创建、传播并行进行; 另一方面又支持不同存储模型和共识协议的复用组合, 并可根据多元化应用需求灵活定制 DAG 区块链.

(2) 为了实现高吞吐和低延迟的性能目标, 本文分别对存储模型和共识机制进行了灵活设计. 针对存储模型, 本文提出了自适应区块确认策略和基于划分的确认区块排序算法, 提高了 DAG 区块链系统的吞吐量和活性. 针对共识机制, 本文提出了低延迟 DAG 区块链混合共识协议, 有效降低了区块/交易的确认延迟.

(3) 本文实现了一个 ElasticDAG 原型系统, 并进行了测试. 实验表明, ElasticDAG 实现了高吞吐和低延迟, 在广域网下的吞吐量高达 11 Mb/s, 并具有十秒级确认性能. 与 OHIE 相比, ElasticDAG 在实现同等吞吐量的情况下, 可将确认延迟降低 17 倍; 与 Haootia 相比, ElasticDAG 在实现同等共识延迟的情况下, 可将安全性从 91.04% 提升到 99.999 914%.

本文第 1 节介绍了本文的相关工作. 第 2 节介绍了 ElasticDAG 系统的总体设计. 第 3 节介绍了存储模型的设

计与实现. 第 4 节介绍了 DAG 区块链混合共识协议的设计与实现. 第 5 节分析了 ElasticDAG 系统的安全性. 第 6 节分析了 ElasticDAG 原型系统的实验测试结果. 第 7 节对本文进行了总结和展望.

1 相关工作

为突破串行拓扑结构造成的性能瓶颈, DAG 区块链采用并行 DAG 拓扑结构组织区块, 支持对区块链的并发更新, 从而有效提升系统吞吐量. 如表 1 所示, 根据共识类型, 现有 DAG 区块链可分为基于概率性共识协议的 DAG 区块链和基于确定性共识协议的 DAG 区块链两类.

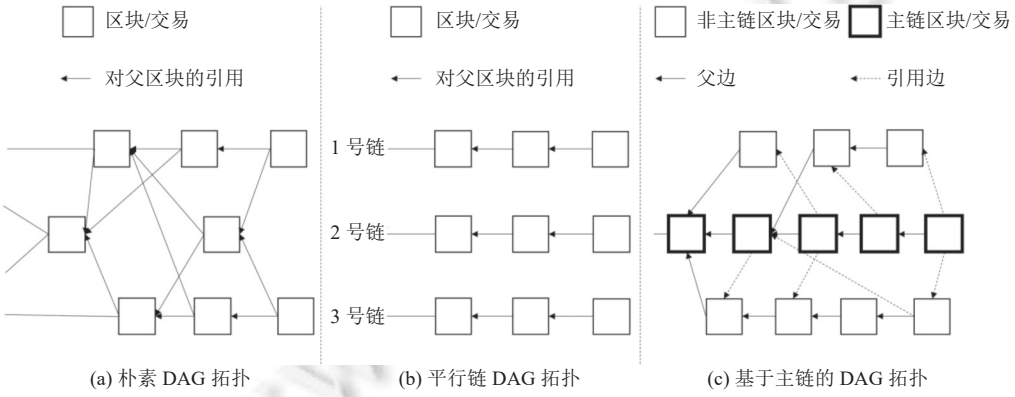
表 1 DAG 区块链的对比

名称	共识协议			存储模型		性能	
	共识类型	公有链	全序	数据单元	DAG拓扑	吞吐量	延迟
IOTA[8]	概率性	√	×	交易	朴素DAG	低	高
Spectre[9]	概率性	√	×	区块	朴素DAG	高	低
Conflux[10]	概率性	√	√	区块	主链DAG	高	高
Prism[11]	概率性	√	√	区块	平行链DAG	高	高
OHIE[12]	概率性	√	√	区块	平行链DAG	高	高
DAG-Rider[13]	确定性	×	√	区块	平行链DAG	高	低
Narwhal and Tusk[14]	确定性	×	√	区块	平行链DAG	高	低
Byteball[15]	确定性	√	×	交易	主链DAG	低	低
Haootia[16]	确定性	√	√	交易	朴素DAG	低	低
ElasticDAG	任意	√	√	任意	任意	高	低

注: 吞吐量低: 低于4 Mb/s, 折合数十至数百tps, 吞吐量高: 高于4 Mb/s, 折合数千tps; 延迟低: 低于1 min, 延迟高: 数分钟至数十分钟

(1) 基于概率性共识的 DAG 区块链. 概率性共识协议的优点是通信复杂度低、可扩展性高和能够直接支持公有链场景. 概率性共识协议的缺点是确认延迟高, 不适用于延迟敏感型应用. 这是因为在概率性共识协议中, 只有那些在 DAG 区块链中具有足够深度或权重的区块/交易才能被安全地确认.

● IOTA 直接采用交易作为 DAG 区块链的数据单元, 被称为基于交易的 DAG 区块链 (transaction-based DAG, txDAG). 由于省略了区块打包的步骤, 交易能够得到及时的验证和上链, 但这也导致 DAG 拓扑中的引用关系复杂且存储开销高. 此外, IOTA 采用如图 1(a) 所示的朴素 DAG 拓扑, 只能为交易提供偏序, 无法确定全序, 难以支持拍卖、智能合约等执行顺序会影响执行结果的应用.



(a) 朴素 DAG 拓扑 (b) 平行链 DAG 拓扑 (c) 基于主链的 DAG 拓扑

图 1 常见 DAG 拓扑

● Spectre 采用区块作为 DAG 区块链的数据单元, 被称为基于区块的 DAG 区块链 (block-based DAG, blockDAG). blockDAG 采用批处理的形式, 一组交易先被打包进区块再上链, 因此相较于 txDAG, 通常具有更高的吞吐量, 但

同时也增加了交易的确认延迟. 由于同样采用朴素 DAG 拓扑, Spectre 无法为交易确定全序. 此外, Spectre 的性能缺乏实验证明.

● Conflux 采用如图 1(c) 所示的主链 DAG 拓扑, 解决了朴素 DAG 拓扑难以提供全序的问题. 主链 DAG 拓扑本质是在朴素 DAG 拓扑的基础上选出部分区块组成一条主链, 然后根据主链区块对其他区块的引用关系来确认和排序其他区块. 在 Conflux 中, 每个区块通过一条父边 (parent edge) 和若干条引用边 (reference edge) 引用前驱区块. 如果忽略引用边, 那么 Conflux 的拓扑结构会从 DAG 退化成一棵树, 可使用 GHOST 协议从树型拓扑结构中选出一条主链, 继而实现对 DAG 区块链中其余区块的确认和排序. 正常情况下, Conflux 在吞吐量和确认延迟两方面都表现优异, 但 Conflux 容易受到活性攻击, 攻击者可以通过保留区块来维持两条备选主链, 导致 Conflux 的共识协议陷入瘫痪. Conflux 通过增加确认延迟来抵抗活性攻击, 但这将导致确认延迟增加至数十分钟, 并持续数个小时.

● Prism 采用如图 1(b) 所示的平行链 DAG 拓扑结构, 其本质是并行运行多条区块链以提高区块产生速度. 平行链 DAG 拓扑有利于实现可证明安全性, 这是因为平行链 DAG 拓扑可以被分解为多条链式区块链, 其理论和安全性分析可基于现有链式区块链的安全性理论. Prism 将比特币共识协议分解为交易打包、提案生成和投票这 3 个阶段, 并分别对各个阶段进行扩展. 其中, 针对交易打包, Prism 并行产生数据区块, 提高了交易打包的效率; 针对提案生成, Prism 单独运行一条提案链, 其中的提案区块通过引用数据区块来形成完整提案; 针对投票, Prism 并行运行多条投票链, 在实现快速投票的同时, 单条投票链仍以较低速率生长, 保证了高安全性. 然而, Prism 的设计仍然局限于工作量证明 (proof-of-work, PoW)[1]类共识协议, 并且要求底层网络是同步网络.

● OHIE 也采用了如图 1(b) 所示的平行链 DAG 拓扑结构. OHIE 并行运行多个区块链实例, 每个实例都是一条基于 PoW 和最长链规则的链式区块链. 新产生的区块被平均分配到多条区块链上, 单条区块链的生长速率仍处于较低水平, 从而避免因频繁分叉造成的安全性问题. 为了提供全序, OHIE 在不同实例之间基于实例编号和区块虚拟高度对确认区块进行排序. 然而, OHIE 的共识协议是 PoW 类共识协议的变体, 确认延迟仍然很高. 此外, OHIE 在实例内部采用最长链法则, 并发产生的分叉区块会被废弃, 不能充分利用 DAG 并发出块带来的性能优势.

(2) 基于确定性共识的 DAG 区块链. 确定性共识协议一般基于通信和投票, 优点是共识一旦达成便不可更改, 区块/交易的确认是确定性的, 因此在延迟方面具有显著优势. 然而, 确定性共识协议也存在着难以支持公有链场景、通信复杂度高和可扩展性差等问题.

● DAG-Rider 是一个基于平行链 DAG 拓扑的异步 BFT 共识协议. 在广播阶段, 每个节点利用可靠广播 (reliable broadcast) 协议[18]广播本地产生的区块, 区块间的引用代表节点对区块的投票. 在随后的共识阶段, 每个节点根据本地 DAG 拓扑确定性地对区块进行确认和排序, 无需额外的通讯和领导者的协调. 尽管 DAG-Rider 在吞吐量和确认延迟方面表现出高性能, 但它只适用于联盟链场景, 且存在可扩展性问题.

● Narwhal and Tusk 通过分离交易分发和交易排序来提高 BFT 协议的性能, 其中内存池协议 Narwhal 负责完成交易分发, DAG 区块链共识协议 Tusk 负责完成交易排序. 在 Narwhal 协议中, 节点产生和收到的区块被存储进本地内存池, 不同节点之间使用可靠广播协议完成内存池同步. Tusk 是 DAG-Rider 的实现和扩展, 它一方面优化了 DAG-Rider 的参数设置, 实现了更低的确认延迟; 另一方面实现了一个垃圾回收机制, 提高了资源利用率. 和 DAG-Rider 一样, Narwhal and Tusk 也只适用于联盟链场景, 同样存在可扩展性问题.

● Byteball 以交易作为数据单元, 并采用了主链 DAG 拓扑, 具有确认延迟低的优点. 在 Byteball 中, 每个节点选择一组信誉良好的节点作为自己的见证人 (witness). 见证人发布的交易被称为见证单元, 节点以见证单元为基础在 DAG 拓扑中选出一条主链, 所有被主链引用的交易成为确认交易. 每个确认交易都有一个主链索引 (main chain index), 它表示第 1 个引用该交易的主链单元在主链上的高度, 当确认交易发生冲突时, 主链索引更低的那个被认为是有效交易. Byteball 在共识层面上完全依赖少数见证人节点, 这导致系统存在中心化风险. 此外, Byteball 没有规定见证人的选择规则, 而是由各个节点自己决定, 这导致不同节点的见证人列表、主链和确认交易排序结果并不完全相同, 因此无法为交易提供全序.

● Haootia 在数据单元和 DAG 拓扑上采用了和 IOTA 相同的设计, 但在共识协议层面, Haootia 采用效率更高

的混合共识协议[19], 即节点先通过 PoW 动态地成为委员会成员, 然后在委员会内部通过 BFT 协议确定性地确认和排序 DAG 区块链中的区块/交易. 然而, Haootia 的混合共识协议存在两个问题: 一方面, Haootia 的委员会选举机制要求在 DAG 区块链之外运行一条链式区块链, 这导致诚实节点的算力被稀释, 可能会影响系统安全性; 另一方面, Haootia 使用 PBFT 协议[20]作为委员会内部的共识协议, 为了保证性能, 其委员会规模较小, 导致委员会很容易被攻击者控制, 存在严重的安全性问题.

综上所述, 目前针对 DAG 区块链的研究已经取得一定进展, 但现有方案仍存在一些局限性: 或只能提供概率性共识, 导致交易确认延迟高 (如 Conflux 和 OHIE); 或在吞吐量方面表现欠佳 (如 Byteball); 或存在安全隐患 (如 Haootia); 或难以适用于公有链场景 (如 DAG-Rider 和 Narwhal and Tusk) 等, 无法兼顾安全性、高性能和支持公有链场景. 此外, 现有 DAG 区块链系统都是针对特定数据单元类型、DAG 拓扑和共识协议设计的, 因此缺乏弹性, 难以灵活调整自身设计以满足多元化应用需求.

2 ElasticDAG 系统总体设计

本节首先介绍 ElasticDAG 的系统模型、网络模型和威胁模型, 接着介绍 ElasticDAG 的设计目标和技术挑战, 最后介绍 ElasticDAG 的系统架构. 表 2 列举了本文常用的符号及其含义.

表 2 本文使用的符号及其定义

符号	定义
$node_i$	全节点i
$\|S\|$	求集合S中的元素数量
$H(m)$	消息m的哈希值
P	攻击者控制的算力与系统总算力的比值
B_j^i	平行链DAG拓扑中的区块, 其中i为区块链实例编号, j为区块高度
q_i	共识队列的第i+1个元素
s	混合共识协议中的委员会规模, 即委员会成员数量
N_{chain}	平行链DAG拓扑中的区块链实例数量

2.1 模 型

(1) 系统模型

ElasticDAG 是一个公有 DAG 区块链系统, 所有参与者都是匿名的, 且随时可以加入或离开网络.

ElasticDAG 中包含两类参与者: 轻节点和全节点. 轻节点相当于客户端, 主要负责产生交易. 由于轻节点的计算和存储资源有限, 既不存储完整的 DAG 区块链, 也不参与 DAG 区块链的维护和共识. 全节点的计算、存储和网络资源充足, 负责处理交易和维护 DAG 区块链, 即每个全节点将在本地保存一份 DAG 区块链副本并参与共识.

ElasticDAG 使用状态复制机 (state machine replication, SMR) 算法[21]作为处理交易的计算层. SMR 算法要求所有节点从相同的初始状态开始, 然后以相同的顺序执行相同的交易, 从而保证所有节点的最终状态相同.

在 ElasticDAG 中, DAG 区块链的区块可包含 1 个交易 (txDAG) 或多个交易 (blockDAG), 并且至少引用一个前驱区块. 区块作为 DAG 区块链的基本数据单元, 全节点通过产生新区块的方式向 DAG 区块链新增数据.

DAG 区块链的拓扑结构使用 $G=(V, E)$ 表示. 其中 V 为全部顶点的集合, 每个顶点表示一个区块; E 为全体边的集合, 每条边表示一个区块对另一个区块的引用关系. 如果存在 $v_1, v_2 \in V$ 和 $(v_1, v_2) \in E$ 那么称 v_1 直接引用了 v_2. 对于 $v_1, v_n \in V$, 如果不存在 $(v_1, v_n) \in E$, 但存在 $v_2, v_3, \ldots, v_{n-1} \in V$ 和 $(v_1, v_2), (v_2, v_3), \ldots, (v_{n-1}, v_n) \in E$, 那么称 v_1 间接引用 v_n. 除特殊说明之外, 本文中的引用包括直接引用和间接引用.

(2) 网络模型

和许多区块链系统[19,22]以及 BFT 协议一样[20,23,24], ElasticDAG 假设 DAG 区块链的底层网络是一个部分同步

网络. 即存在一个已知的消息传递时间上界 Δ 和一个未知的全局稳定时间 (global stabilization time, GST), 使任何在时刻 t 发送的消息都会在 Δ+max(t, GST) 之前到达. 此外, 所有全节点构成一个 P2P 网络, 并通过 Gossip 协议广播消息. 现有研究指出, 任何通过 Gossip 协议广播的消息, 都可以在有限时间内传播到所有节点[25].

(3) 威胁模型

使用 $node_i \in N$ 表示全节点 i, 其中 N 表示所有全节点的集合. N 中的全节点分为两类: 诚实节点和受攻击者控制的拜占庭节点. 攻击者在系统初始化时可指定拜占庭节点, 拜占庭节点的集合用 A 表示, 剩余的节点为诚实节点, 它们的集合用 H 表示. A 中的拜占庭节点可以执行任意操作, 包括延迟、重传、修改来自诚实节点的消息, 但拜占庭节点不能打破网络模型和标准密码学的假设.

每个全节点在本地配置一个可信执行环境 (trusted execution environment, TEE)[26]硬件装置. TEE 可在内存中构建一个安全区域, 区域内的进程可读取或修改区域外的数据, 但区域外的进程不能读取或修改区域内的数据. 拜占庭节点可修改包括操作系统在内的所有安全区域之外的进程的程序和数据, 但不能修改安全区域内部的进程的程序和数据.

诚实节点通过签名来防止攻击者伪造或篡改消息. 具体而言, 全节点在发出消息 m 时, 也会使用自己的私钥对 m 的哈希值 $H(m)$ 签名, 其余节点可使用公开的公钥验证签名. 由于诚实节点不会向其他节点分享自己的私钥, 因此攻击者无法伪造或篡改来自诚实节点的消息而不被发现.

为了抵御女巫攻击[27]、洪泛攻击和控制区块产生速度, 全节点通过 PoW 产生新区块. 令 P 表示攻击者控制的算力与系统总算力的比值. 本文和 Byzcoin[19]等工作一样假设系统始终满足 $P \leqslant 0.25$. 需要注意的是, 虽然早期的研究工作认为比特币可以在 $P<0.5$ 的情况下保证安全性, 但目前关于自私挖矿[28]的研究表明, 比特币只能在 $P \leqslant 0.25$ 的情况下保证安全性, 因此本文继承了比特币对攻击者算力的假设. 最后, 攻击者是缓慢自适应的, 即攻击者可以在系统运行过程中将 H 中的诚实节点转化为拜占庭节点, 但攻击者每次只能转化 H 中的一小部分节点, 且这种转化需要一段较长的时间, 同时攻击者不能打破 $P \leqslant 0.25$ 的假设.

2.2 设计目标和技术挑战

ElasticDAG 系统需要实现以下设计目标.

(1) 弹性: 为了适应多元化应用需求, 系统应该具有弹性, 包括: 1) 可复用组合不同设计; 2) 可灵活定制 DAG 区块链以适应多元化应用需求.

(2) 全序: 所有诚实节点以相同的顺序确认 (提交) 相同的区块.

(3) 确定性共识: 对于任意诚实节点, 如果它在时刻 $t1$ 确认/提交了区块 b, 那么在时刻 $t2$ ($t2>t1$) 也一定会确认/提交 b.

为了实现上述目标, 本文需要解决以下挑战.

为了提高 DAG 区块链在系统架构层面上的弹性, 需要对 DAG 区块链进行解耦, 以实现一个松耦合的 DAG 区块链系统. 解耦设计思想已经在链式区块链中得到了广泛应用[19,29], 但据我们所知, 本文是第 1 个对 DAG 区块链进行系统化解耦的工作. 经过深入分析, 本文发现 DAG 区块链包含存储模型和共识协议两个部分. 然而, 由于存储模型和共识协议之间存在复杂的相互作用关系, 因此难以直接进行解耦. 为此, 本文设计了弹性 DAG 区块链系统 ElasticDAG, 其在系统架构层面上的关键创新是在不影响安全性的前提下, 最小化存储模型和共识协议之间的交互, 让两者可以独立、并行的运行 (第 2.3 节).

由于 DAG 拓扑本身只能提供偏序, 因此 DAG 区块链系统难以为确认区块/交易提供全序. 为此, 必须设计一个确认区块排序算法, 并保证所有诚实节点都能得到相同的确认区块排序结果 (第 3 节).

为了实现确定性共识, 必须采用 BFT 协议作为共识协议. 然而, BFT 协议不能直接在公有链场景下运行, 同时存在可扩展性问题. 为了解决上述问题, 需要设计一个混合共识协议, 其基本思想是选举部分全节点组成委员会, 由于委员会内部相当于一个许可环境, 因此可在委员会内部运行 BFT 协议.

然而, 在 DAG 区块链中运行混合共识协议面临着以下 2 个挑战. 首先, 攻击者可以通过控制委员会来破坏整

个系统的安全性, 因此, 必须设计一种能够抵御攻击的委员会选举机制 (第 4.2 节). 虽然已有一些工作尝试解决这个挑战[16,19], 但它们的委员会选举都是基于链式区块链的, 在 DAG 区块链中会造成算力稀释, 进而影响安全性. 其次是如何在安全性和性能之间取得平衡 (第 4.3 节). 从安全性的角度考虑, 委员会规模越大, 委员会被攻击者控制的概率越小. 然而, 从性能的角度考虑, 委员会规模越小, 委员会内部共识 (BFT 协议) 的性能越好. 虽然已有一些工作尝试解决这个问题, 但仍存在不支持公有链场景[30]、通信拓扑容易受到攻击[19]等问题.

2.3 系统架构

如图 2 所示, ElasticDAG 包含全节点和轻节点两类节点. 全节点的数量较少, 负责处理交易、维护 DAG 区块链和参与共识, 并共同组成一个 P2P 网络. 轻节点的数量较多, 负责产生交易. 每个轻节点随机连接常数个全节点, 并将自己产生的交易发给这些全节点. 全节点在收到轻节点发出的交易后, 通过 P2P 网络和 Gossip 协议向其他全节点广播消息.

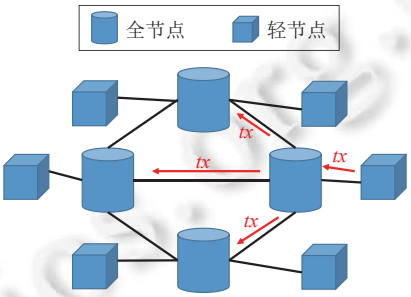


图 2 ElasticDAG 的网络拓扑

如图 3 所示, ElasticDAG 系统由存储模型、共识机制和计算模型组成. 其中, 存储模型负责创建、传播和存储区块, 共识机制负责确认区块, 计算模型负责执行区块中的交易. 存储模型和共识机制可并行运行, 在后者通过动态共识协议确认未确认区块的同时, 前者可以继续创建和传播新区块.

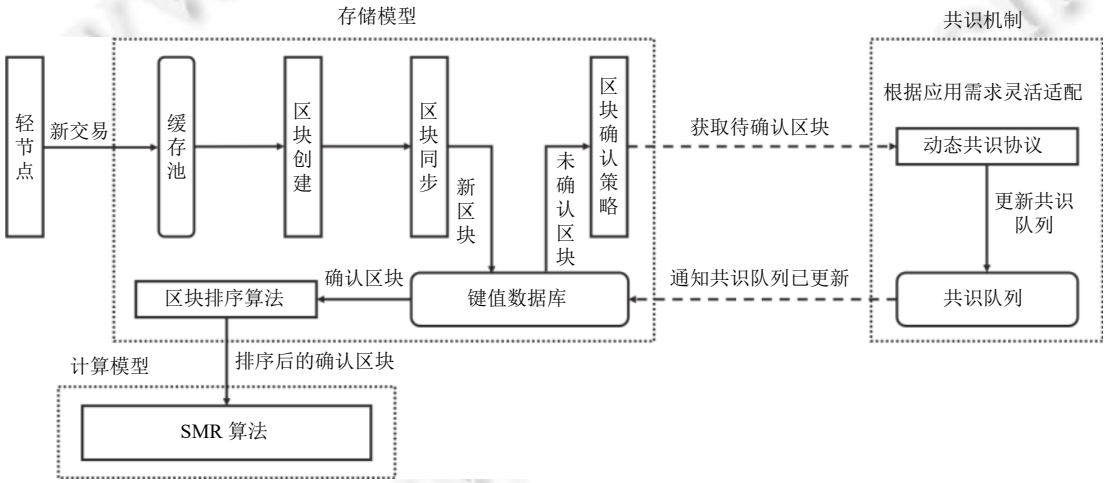


图 3 ElasticDAG 的系统架构

存储模型包含缓存池、键值数据库、区块创建策略、区块同步策略、区块确认策略和区块排序算法. 缓存池用于缓存客户端发出的交易. 键值数据库负责以键值对 ($H(b)$, $\{b, state\}$) 的形式存储区块, 其中 b 表示区块, $H(b)$ 表示 b 的哈希值, $state$=0 和 $state$=1 分别表示 b 为未确认区块和 b 为确认区块. 区块创建策略负责打包交易并创建新区块. 区块同步策略负责完成 DAG 区块链的数据同步, 包括区块的广播、转发以及在新节点加入网络时同

步历史区块. 区块确认策略负责从 DAG 区块链中获取未确认区块, 然后决定哪些区块需要被共识机制直接确认. 区块排序算法负责对确认区块进行排序.

共识机制包含动态共识协议和共识队列两个部分. 其中, 动态共识协议负责对待确认区块集合达成一致, 它的动态性体现在 ElasticDAG 的共识机制可兼容任意共识协议; 共识队列负责以仅追加写的方式依次记录动态共识协议的每一轮输出. 共识队列中的元素主要包含以下 4 个字段: 1) *SerialNum*: 该字段用于记录共识队列元素的序号, 表示其在共识队列中的位置; 2) *PreHash*: 该字段用于存储前一个共识队列元素的哈希值, 以防止拜占庭节点对共识队列进行修改; 3) *Refs*: 该字段存储共识队列元素对 DAG 区块链中区块的引用; 4) *Proof*: 该字段用于验证共识队列元素的合法性. 共识机制按照轮次运行, 第 i 轮运行的步骤如下.

(1) 新建共识队列元素 q_i, 并将前一个共识队列元素的哈希值存储到 q_i.*PreHash* 字段.

(2) 通过存储模型的区块确认策略从 DAG 区块链中获取第 i 轮共识的待确认区块, 然后将这些区块的哈希值存储到 q_i.*Refs* 字段.

(3) 以 q_i 为输入运行动态共识协议.

(4) 动态共识协议运行结束后, 将能够证明共识结果合法性的数据存入 q_i.*Proof* 字段. 例如, 当动态共识协议为 PBFT 协议时, *Proof* 字段用于存储投票消息; 当动态共识协议为 PoW 时, *Proof* 字段用于存储 PoW 的随机数.

(5) 将 q_i 添加到共识队列的尾部.

对于区块 b 和共识队列元素 q_i, 如果有 $H(b)\in q_i$.*Refs*, 那么称 q_i 直接引用了 b. 对于区块 b 和共识队列 q, 如果存在 $q_i\in q$ 和 q_i 直接引用了 b, 那么称 q 直接引用了 b; 如果 q 直接引用了区块 b, 但没有直接引用 b 的祖先区块 b', 那么称 q 间接引用了 b'. 如果一个区块被 q 直接或间接引用, 那么称该区块为确认区块, 否则为未确认区块.

计算模型基于 SMR 算法处理确认交易. 它首先从存储模型获取经过排序的确认区块, 然后根据交易在确认区块内部的顺序以及确认区块之间的先后顺序对确认交易进行排序, 最后根据交易排序结果依次执行各个交易并更新状态机的状态.

在 ElasticDAG 中, 交易 tx 从产生到执行需要经过以下步骤.

(1) 区块创建: 轻节点广播 tx, 全节点收到 tx 后验证其合法性, 如果验证通过, 则将 tx 加入本地缓存池. 区块创建策略从缓存池中选取一个或多个交易, 然后通过 PoW 产生一个新区块 b 以打包这些交易.

(2) 区块传播: 创建 b 的全节点在本地键值数据库中新增记录 $(H(b), \{b, 0\})$, 然后通过区块同步策略广播 b. 其他全节点在收到 b 之后, 首先检查本地键值数据库中是否存在键值为 $H(b)$ 的记录. 如果不存在, 则检查 b 的合法性, 如果检查通过, 则在本地键值数据库中新增记录 $(H(b), \{b, 0\})$. 此时 b 为未确认区块.

(3) 共识: 共识机制与存储模型并行运行. 其中, 共识机制按照轮次运行, 详细的运行步骤已在前文中说明, 因此本部分不再赘述.

(4) 区块确认: 一轮共识结束后, 共识机制更新共识队列并通知存储模型. 存储模型读取更新后的共识队列 q, 并将所有被 q 直接或间接引用的区块的 *state* 设置为 1, 这些区块即为确认区块. 当 b 成为确认区块后, b 中的交易 tx 也成为了确认交易.

(5) 交易执行: 存储模型首先通过区块排序算法对确认区块进行排序, 然后通知计算模型, 计算模型读取排序后的确认区块, 然后基于 SMR 算法执行其中的交易.

ElasticDAG 在系统架构层面上具有以下优势: 1) ElasticDAG 的设计遵循了“低耦合、高内聚”的设计原则, 有利于提升系统弹性. 无论存储模型采用何种设计, 共识机制总是通过存储模型自定义的区块确认策略获取待确认区块, 不需要了解存储模型的内部实现细节. 类似地, 无论共识机制采用何种共识协议, 存储模型总是通过共识队列读取共识结果, 不需要了解动态共识协议具体是哪种共识协议. 这样的松耦合设计提升了系统弹性, 它一方面可支持任意存储模型和共识协议的复用组合, 另一方面可根据多元化应用需求分别设计存储模型和共识协议. 2) 存储模型和共识机制并行运行, 有利于提升系统性能. 具体而言, 共识机制定期从存储模型中读取待确认区块, 然后在内部运行动态共识协议, 并在动态共识协议运行结束时通知存储模型更新确认区块, 动态共识协议与区块的创建和传播互不影响, 因此可并行运行.

ElasticDAG 通过区块创建策略、区块同步策略、区块确认策略和区块排序算法来自定义存储模型. 其中, ElasticDAG 的存储模型可以兼容任意数据单元类型和 DAG 拓扑, 这些特性由区块创建策略决定. 此外, 用户还可以自定义创建区块时是否需要 PoW 或权益证明 (proof-of-stake, PoS)[31]等机制. 针对区块同步策略, 在同步网络/部分同步网络下, 可以直接广播或通过 Gossip 协议广播区块; 而在异步网络下, 则必须使用可靠广播协议. 此外, 区块确认策略可以简单地设计为返回所有未确认区块 (即 *state*=0 的区块) 作为待确认区块. 然而, 更常用和高效的设计是仅返回未确认区块中的 tips 区块, 即那些没有入边的区块[8,16].

3 存储模型设计与实现

平行链 DAG 拓扑可以被分解为多条链式区块链, 其理论和安全性分析可基于现有链式区块链的安全性理论, 因此容易实现可证明安全性, 是目前主流的 DAG 拓扑.

然而, 由于存在分叉, 平行链 DAG 拓扑在吞吐量和活性两个方面均存在不足. 具体而言, 由于区块的产生和传播并行进行, 因此可能出现分叉, 即同一个区块链实例中存在多个高度相同的区块. 现有策略是在每个高度上只确认一个区块[12,32], 如 OHIE 只确认最长链上的区块. 现有策略的优点是能够直接在区块链实例内部实现全序, 有利于简化确认区块排序. 然而, 现有策略也带来了以下问题: 1) 由于一部分区块被抛弃, 因此造成了吞吐量损失, 此外, 这些区块消耗的计算资源和网络资源也被浪费; 2) 被抛弃区块中的交易永远无法得到确认, 这降低了系统活性.

为了在保证一致性和提供全序的同时提高平行链 DAG 拓扑的吞吐量和活性, 必须解决以下两个挑战.

(1) 如何激励全节点维护平行链 DAG 拓扑: 由于分叉区块也能得到确认, 因此全节点缺乏及时同步 DAG 区块链并将新区块添加到最长链末端的动力, 可能会有大量全节点基于未及时同步的本地 DAG 区块链创建新区块, 这将导致大量分叉并破坏平行链 DAG 拓扑在结构上的一致性.

(2) 如何为确认区块提供全序: 由于拜占庭节点可以产生任意高度的分叉区块, 因此确认分叉区块带来的一个挑战是无法再利用链式拓扑结构确定单个区块链实例内部的确认区块顺序, 这增加了设计确认区块排序算法, 为确认区块提供全序的难度.

针对第 (1) 个挑战, 本节设计了一个自适应区块确认策略, 该策略在分叉区块较少时确认全部分叉区块, 在分叉区块较多时随机确认部分分叉区块. 如果有全节点故意产生分叉区块, 那么会增加区块分叉率, 导致自己的区块难以得到确认, 因此该策略可以激励全节点及时更新本地 DAG 区块链并将新区块添加至最长链末端. 当大部分全节点正常产生区块, 区块分叉率维持在正常水平时, 该策略保证了所有分叉区块都能得到确认.

针对第 (2) 个挑战, 本节设计了一个基于划分的确认区块排序算法. 该算法首先根据共识队列对区块的引用关系将确认区块划分到不同轮次, 然后根据轮次的先后顺序对确认区块做初步排序. 由于未来产生的区块不会被现在的共识队列元素引用, 因此划分步骤确保了拜占庭节点在未来无法通过创建分叉区块来破坏排序结果的一致性. 为了最大化利用 DAG 拓扑所提供的偏序信息, 在同一轮次内部, 该算法基于 DAG 拓扑对确认区块排序.

(1) 区块确认策略

为了确认分叉区块, 本节设计了算法 1 所示的自适应区块确认策略, 其中 N_{chain} 为平行链 DAG 拓扑中的区块链实例数量. 算法 1 首先确认所有最长链 (第 1 行), 然后在分叉较少的情况下确认所有分叉链, 或在分叉较多的情况下确认比例为 R 的分叉链 (第 2–5 行). R 是一个系统参数, 并且有 R≈单个区块链实例正常情况下的分叉区块比例.

算法 1. 自适应区块确认策略.

输入: DAG 区块链 G;

输出: 待确认区块集合 S.

1. $S1 \leftarrow \{b \mid$ last block on the longest chain of blockchain instance i, $i = 0, 1, 2, \ldots, N_{\text{chain}}-1\}$
2. $S2 \leftarrow \{b \mid$ the last block on each fork chain$\}$
3. **if** $S2$.length > $N_{\text{chain}} \times R$ then //如果分叉链很多, 那么随机确认 $N_{\text{chain}} \times R$ 条分叉链

4. $S2 \leftarrow$ sortition($S2$, $N_{chain} \times R$)
5. **end**
6. $S \leftarrow S1 \cup S2$
7. **return** S

如果有全节点为了避免同步 DAG 区块链的开销, 基于未及时同步的本地 DAG 区块链创建新区块, 那么会导致大量分叉, 使得自己的区块难以得到确认. 这激励着全节点及时同步 DAG 区块链, 并将自己的区块添加到最长链末端. 在这种情况下, DAG 区块链中的分叉较少, 所有分叉区块都能得到确认.

(2) 确认区块排序算法

为了在确认分叉区块的同时为确认区块提供全序, 本节设计了算法 2 所示的基于划分的确认区块排序算法. 拜占庭节点可通过产生分叉区块来破坏任意基于链式拓扑结构的确认区块排序算法. 为了克服这一挑战, 算法 2 利用了以下性质: 对于区块 B, 如果 q_k 是第 1 个引用 B 的共识队列元素, 那么在 q_k 之后产生的区块一定不会被 q_k 和 q_k 之前的共识队列元素引用. 利用这一性质, 可首先基于共识队列对确认区块做初步划分, 以防止拜占庭节点在未来通过产生分叉区块来篡改确认区块排序结果.

算法 2. 确认区块排序算法.

输入: 共识队列 *consensusQueue*; 确认区块的集合 *cfmBlocks*;
输出: 确认区块的排序结果 *Seq*.

1. *Seq* ← *Block*[]
2. **for** $i \leftarrow 0, 1, 2, \ldots,$ *consensusQueue.length* – 1 **do**
   //共识队列中的每个元素对应一个轮次, 首先将确认区块划分到不同轮次
3. $S \leftarrow \{b \mid$ block in *cfmBlocks* and referenced by *consensusQueue*[i]$\}$
4. delete all blocks referenced by *consensusQueue*[i] from *cfmBlocks*
5. $S1 \leftarrow$ longest chain blocks in S //最长链确认区块
6. $S2 \leftarrow$ remaining blocks in S //分叉确认区块
   //为了最大化利用 DAG 拓扑提供的偏序信息, 基于 DAG 拓扑对最长链确认区块排序
7. **while** $S1$.length > 0 **do**
8. **for** $j \leftarrow 0, 1, 2, \ldots, N_{chain} - 1$ **do**
9. **if** there are blocks of blockchain instance j in $S1$ **then**
10. $b \leftarrow$ the block with the lowest height and belonging to both $S1$ and instance j
11. delete b from $S1$
12. append(Seq, b)
13. **end**
14. **end**
15. **end**
   //对剩余的分叉确认区块排序
16. sort $S2$ by block height, the number of blockchain instance and hash value of block
17. append(Seq, $S2$)
18. **end**
19. **return** *Seq*

算法 2 将共识队列的每个元素看作一个轮次, 然后根据这些元素对确认区块的引用关系将确认区块划分到不

同轮次 (第 3, 4 行). 对于区块链实例 i 中高度为 j 的区块 B_j^i, 如果 q_k 是第 1 个引用 B_j^i 的共识队列元素, 那么 B_j^i 被划分到 q_k 对应的轮次, 即轮次 k+1. 对于不同轮次中的确认区块, 算法 2 根据轮次的先后关系进行排序, 即当 $i<j$ 时, 轮次 i 的确认区块一定都排在轮次 j 的确认区块之前.

同一轮次的区块按照以下方法排序: 根据区块是否在当前最长链上, 确认区块被分成最长链确认区块和分叉确认区块, 它们的集合分别记作 $S1$ 和 $S2$ (第 5, 6 行). 以图 4 为例, 在轮次 1 中, $\bar{B}_1^1$ 不在最长链上, 因此为分叉确认区块. 在轮次 2 中, $\bar{B}_2^1$ 和 $\bar{B}_3^1$ 在最长链上, 因此为最长链确认区块, 虽然此时 $\bar{B}_1^1$ 也在最长链上, 但由于 $\bar{B}_1^1$ 属于轮次 1, 因此仍为分叉确认区块.

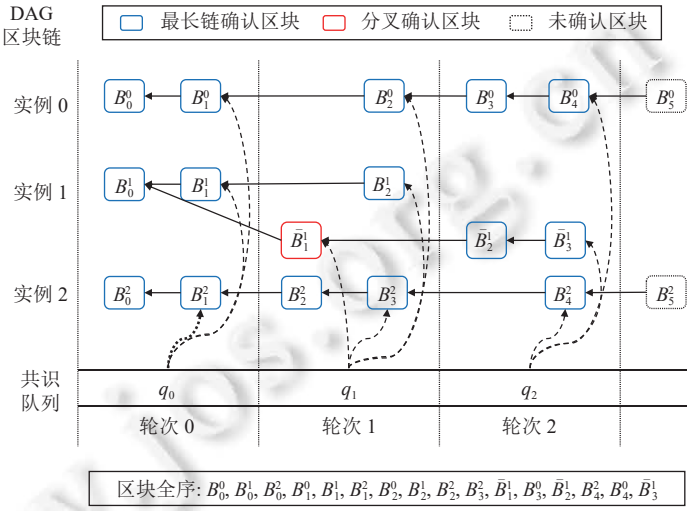


图 4 确认区块排序算法

对于最长链确认区块集合 $S1$, 获取每个区块链实例中高度最低的最长链确认区块, 然后根据区块链实例的编号对获得的区块排序, 最后从 $S1$ 中删去已排序的区块, 不断重复上述过程, 直到 $S1$ 为空 (第 7–15 行).

对于分叉确认区块集合 $S2$, 首先根据区块高度对 $S2$ 中的区块进行排序, 对于高度相同的区块, 根据区块所属的区块链实例编号排序, 对于高度和编号都相同的区块, 根据区块哈希值的大小进行排序 (第 16 行).

以图 4 中的轮次 1 为例, 最长链确认区块集合为 $S1=\{B_2^0, B_2^1, B_2^2, B_3^2\}$, 分叉确认区块集合为 $S2=\{\bar{B}_1^1\}$. 首先对最长链确认区块排序, 按照区块链实例编号顺序依次从 $S1$ 中取出高度最低的最长链确认区块, 得到 B_2^0, B_2^1, B_2^2, 此时 $S1=\{B_3^2\}$. 重复上述过程直到 $S1$ 为空, 得到 B_2^0, B_2^1, B_2^2, B_3^2. 最后对分叉确认区块排序, 得到轮次 1 中的区块排序结果为 B_2^0, B_2^1, B_2^2, B_3^2, $\bar{B}_1^1$.

根据确认区块的排序结果和交易在区块内部的顺序, 可以以连续递增的方式为确认区块中的交易分配一个序号. 序号不仅可以用于确定交易在 SMR 算法中的执行顺序, 还可以用来处理可能存在的重复交易. 对于重复交易, ElasticDAG 只会以最小序号执行一次.

为了保证高吞吐, 本节设计的 DAG 区块链以区块作为数据单元, 此外, 为了抵御洪泛攻击和控制区块产生速度, 区块通过 PoW 产生. 下面将简单介绍区块创建策略和区块同步策略的设计.

(3) 区块创建策略

如图 5 所示, 区块是区块链的基本数据单元, 可分为主体和头部 (head) 两个部分. 其中, 主体以数组方式存储交易列表, 头部存储维护区块合法性和区块链结构的元数据. 头部主要包含以下 6 个字段: 1) *RefRoot*: 该字段存储潜在父区块组成的 Merkle 树的树根, 后文将详细介绍潜在父区块的定义; 2) *Height*: 该字段存储区块的高度; 3) *Timestamp*: 该字段存储区块的创建时间; 4) *Parent*: 该字段存储对父区块的引用, 即父区块的哈希值; 5) *TxRoot*: 在创建区块时, 以交易列表中的交易作为叶子节点计算 Merkle 树[33], *TxRoot* 用于存储这棵树的树根. 此外, 需要

注意的是, 区块主体只存储交易列表, 不会存储计算 Merkle 树的树根过程中产生的中间数据 (图 5 中用虚线方框表示); 6) *Nonce*: 在 PoW 过程中, 不断改变 *Nonce* 的值, 直到满足 H(区块头部)≤D, 其中 D 是预先定义的目标难度.

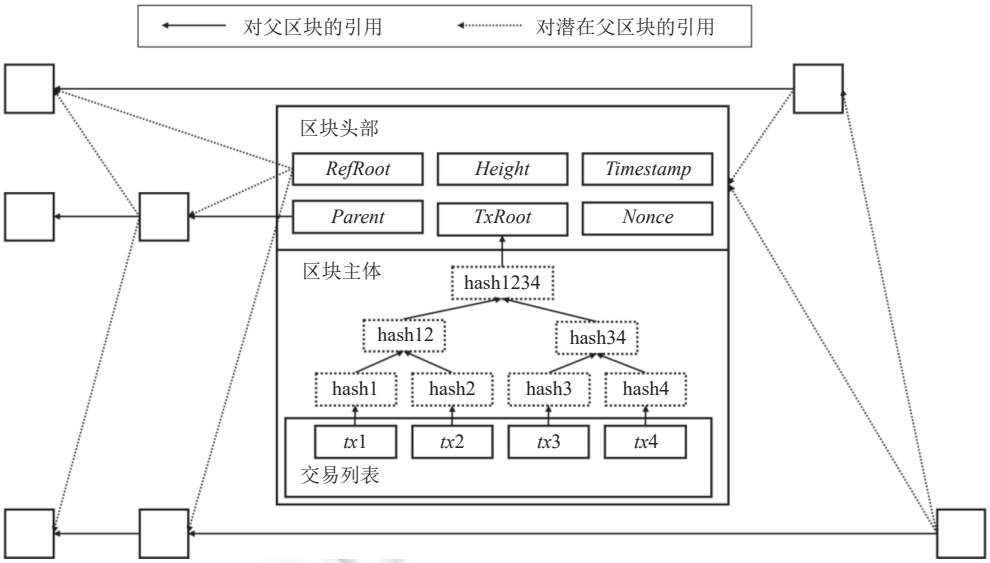


图 5 区块结构

本文对潜在父区块的定义如下: 对于任意节点 $node_i \in N$, 令 G_i^t 表示 $node_i$ 在时刻 t 的本地 DAG 区块链, $instance_j \in G_i^t$ ($j = 0, 1, 2, \ldots, N_{chain}-1$) 表示 G_i^t 的第 j 个区块链实例, $instance_j$ 中高度最高的合法区块即为 $instance_j$ 的潜在父区块. 需要说明的是, 如果 $instance_j$ 的最大高度上包含不止一个区块, 那么 $node_i$ 可任意选择其中的一个区块作为 $instance_j$ 的潜在父区块.

如图 6 所示, 区块创建策略的设计如下: 1) 在创建区块 b 时, 从本地 DAG 区块链的各个区块链实例中获取潜在父区块, 使用 REF_b 表示 b 的潜在父区块集合; 2) 以 REF_b 中的区块作为叶子节点计算 Merkle 树, 设 *root* 为 Merkle 树的树根; 3) 将 *root* 存入 *RefRoot*, 然后进行 PoW, 最后根据 $H(b.head)$ 将 b 随机分配到某个区块链实例中; 4) 产生一个可证明 *Parent* 是以 *RefRoot* 为树根的 Merkle 树中的叶子节点的 Merkle 证明[12].

上述区块创建策略具有以下优点. 首先, PoW 产生的区块的头部哈希值是随机的, 因此新区块会被随机且均匀地分配到各个区块链实例. 这可以避免区块在 1 个或几个区块链实例中集中产生, 从而造成大量分叉. 其次, 任意节点均可基于 Merkle 证明验证一个区块是否引用了正确的父区块, 因此区块分配过程具有不可篡改性, 避免了拜占庭节点集中算力攻击单个实例. 以图 6 为例, 可通过以下步骤来验证 B_2^1 是否是 b 的父区块: 1) 根据 $H(b.head)$ 计算 b 是否被分配到了实例 1; 2) 基于 *Height* 和 *Parent* 验证 b 对 B_2^1 的引用是否合法; 3) 基于 Merkle 证明$\{h_0, h_{23}\}$, 计算 $h_1=H(Parent.head)$, $h_{01}=H(h_0|h_1)$, $root= H(h_{01}|h_{23})$, 最后对比计算得到的 *root* 和 b 中存储的 *RefRoot* 是否相等, 从而验证 B_2^1 是否属于 REF_b.

(4) 区块同步策略

区块同步策略使用 Gossip 协议广播新区块 b, 以及 b 对其父区块的引用和对应的 Merkle 证明, 并利用 TCP 协议完成历史区块的同步. 在收到新区块 b 时, 首先检查 b 的合法性, 如果检查通过, 那么将 b 添加到本地 DAG 区块链. 合法性检查包括: 1) DAG 区块链中是否存在 b 的父区块, 如果不存在, 那么缓存 b 直到 b 的父区块加入 DAG 区块链; 2) b 的 PoW 是否有效; 3) b 是否被分配到了正确的区块链实例中; 4) 验证 b 对父区块的引用和对应的 Merkle 证明是否有效.

对于缓存中的任意区块 b, 由于 ElasticDAG 的底层网络是部分同步网络, 因此如果 b 的父区块存在, 那么一定能够在有限时间内收到 b 的父区块. 因此, 可实现一个基于超时机制的缓存回收策略. 首先, 记录每个区块进入缓存的时间. 其次, 定期扫描缓存中的区块. 对于缓存中的区块 b, 如果 DAG 区块链包含 b 的父区块, 那么从缓存中删除 b, 并再次尝试将 b 加入 DAG 区块链; 否则, 检查 b 在缓存中存在的时间是否超过预先设置的阈值, 如果是, 则从缓存中删除 b.

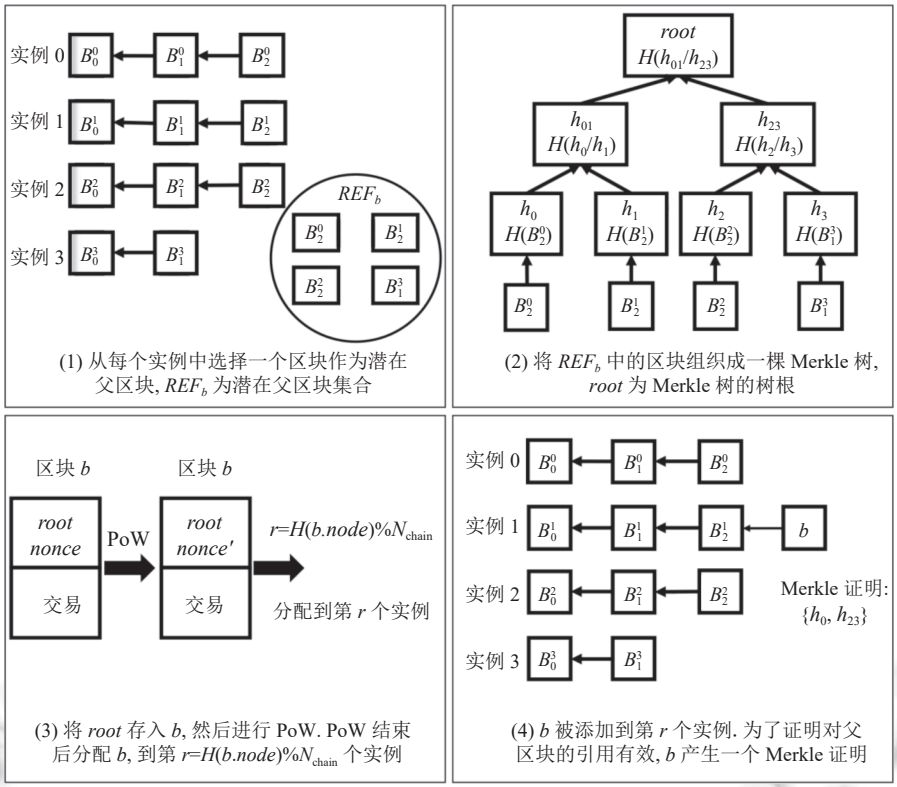


图 6 区块创建策略

4 DAG 区块链混合共识协议

本节为 DAG 区块链设计了一个混合共识协议, 该协议既可以降低确认延迟, 又可以提高 DAG 区块链在共识协议层面上的灵活性. 一方面, 混合共识协议可提供确定性共识, 区块/交易可以被立即确认, 而不需要故意增加延迟以保证安全性, 因此可有效降低确认延迟. 另一方面, 混合共识协议可以从两个方面提高 DAG 区块链在共识协议层面上的灵活性: 1) 混合共识协议可兼容任意 BFT 协议, 因此可适应多样化的网络和威胁环境; 2) 通过调节委员会规模, 可以适应多样化应用需求, 如针对延迟敏感型应用, 可减小委员会规模以降低延迟, 针对高安全性需求应用, 可扩大委员会规模以增加安全性.

本节首先介绍了 DAG 区块链混合共识协议面临的挑战和设计思路, 随后介绍了混合共识协议的委员会选举机制, 最后介绍了混合共识协议的委员会内部共识机制.

4.1 挑战和设计思路

为了在公有 DAG 区块链中实现安全、高效的混合共识协议, 需要解决以下两个挑战.

(1) 如何实现安全的委员会选举: 混合共识协议一般基于链式区块链和时间窗口完成委员会选举, 然而 DAG

区块链的拓扑结构复杂多变, 无法直接运行时间窗口, 也难以作为委员会选举的依据. 现有解决方法是在 DAG 区块链之外添加一条专门用于委员会选举的链式区块链. 这样的设计一方面增加了系统复杂度; 另一方面会降低系统安全性, 例如, DAG 区块链和链式区块链均通过 PoW 产生区块, 因此诚实节点的算力被稀释, 而攻击者则可以集中算力攻击链式区块链以破坏混合共识协议.

(2) 如何实现安全性-性能平衡: 混合共识协议的安全性与委员会规模 s 呈正相关关系, 而性能与 s 呈负相关关系, 现有混合共识协议缺乏灵活性, 很难取得一个合理的 s, 使混合共识协议可以在安全性和性能之间取得平衡. 从安全性的角度考虑, s 越大, 攻击者就越难控制委员会, 混合共识协议的安全性也就越高. 为了使委员会故障概率可以忽略不计, 需要令 $s \geqslant 646$. 从性能的角度考虑, 委员会内部一般通过 BFT 协议达成一致, 由于 BFT 协议存在可扩展性问题, 因此为了保证混合共识协议的性能, 一般需要令 $s \leqslant 100$.

为了解决上述两个挑战, 如算法 3 所示, 本节针对公有 DAG 区块链场景设计了一个新型混合共识协议.

算法 3. 基于 DAG 区块链的混合共识协议.

输入: 选举间隔 T; 委员会规模 s; 共识队列 *consensusQueue*; DAG 区块链 G

1. *candidateBlocks* ← {}
2. **for** $i \leftarrow 0, 1, 2, \ldots$ **do**
   //选择候选区块
3. **for** $j \leftarrow 0, 1, 2, \ldots, T-1$ **do**
4. *candidateBlocks* ← *candidateBlocks* ∪ {b | b in G and referenced by *consensusQueue*[$i \times T + j$]}
5. **if** $i > 0$ **do**
6. delete all blocks referenced by *consensusQueue*[$(i-1) \times T + j$] from *candidateBlocks*
7. **end**
8. **end**
9. *virtualChain* ← sort *candidateBlocks* by the hash value of block //产生本轮执行的虚拟链
10. *committee* ← the first s blocks in *virtualChain* //本轮执行的委员会
11. **for** $j \leftarrow 0, 1, 2, \ldots, T-1$ **do**
12. *res* ← TrustBFT(*committee*) //也可以是其他 BFT 协议
13. push(*consensusQueue*, *res*) //更新共识队列
14. **end**
15. **end**

针对第 (1) 个挑战, 第 4.2 节提出了一种可适应任意 DAG 拓扑的虚拟链委员会选举方法, 其核心思想是首先在共识队列上运行一个时间窗口, 选择最近一段时间的确认区块作为候选区块, 然后对候选区块排序以形成虚拟链, 最后在虚拟链上完成委员会选举. 由于虚拟链仅在逻辑上存在, 因此可以避免在 DAG 区块链之外添加一条链式区块链所带来的问题.

针对第 (2) 个挑战, 通过理论分析, 第 4.3 节发现通过提升 BFT 协议的容错能力, 可以在不影响安全性的前提下缩小委员会规模, 从而灵活地实现安全性-性能平衡. 为此, 第 4.3 节提出了新型 BFT 协议 TrustBFT, TrustBFT 只需要 $n=2f+1$ 个节点即可容忍 f 个拜占庭节点, 而不是传统 BFT 协议所需的 $n=3f+1$ 个节点.

4.2 委员会选举机制

为了确保安全性和公平性, 委员会选举需要实现以下目标: 1) 随机性: 为了防止拜占庭节点控制委员会或提前预知选举结果, 委员会的选举过程必须是随机的; 2) 抵御女巫攻击: 攻击者可以通过伪造多个身份来干扰委员会选举, 为了抵御这种攻击, 委员会选举需要引入 PoW 等抵抗女巫攻击的机制; 3) 周期性: 自适应攻击者可以缓慢地将委员会中的诚实节点转变为拜占庭节点, 以达成控制委员会的目的. 为了阻止这种攻击, 需要定期重新选举委

员会.

如图 7 所示, 委员会选举包含以下步骤: 1) 在共识队列上运行一个时间窗口, 选择最近一段时间内产生的共识队列元素; 2) 根据时间窗口, 可以找到哪些区块是在最近一段时间内被确认的, 这些区块为候选区块, 并将它们按照哈希值排序, 最终形成一条虚拟链; 3) 选择虚拟链的前 s 个区块, 这些区块的创建者组成了委员会; 4) 一段时间后, 时间窗口向后移动, 回到步骤 1), 以重新进行委员会选举.

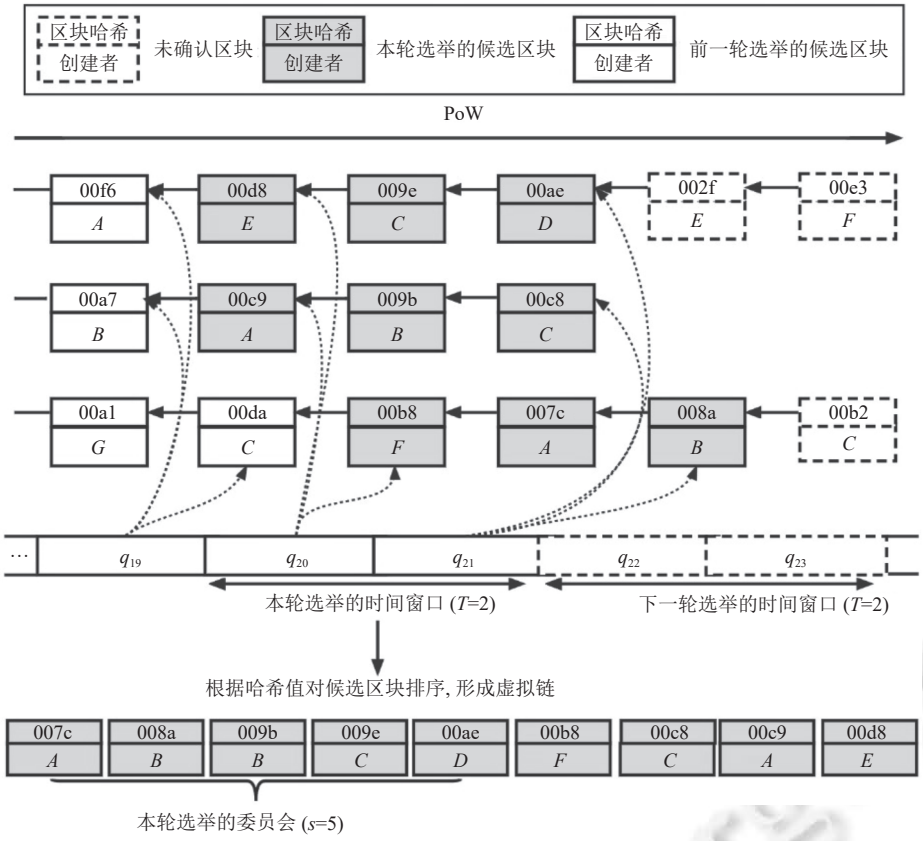


图 7 基于虚拟链的委员会选举机制 (以平行链 DAG 拓扑, T=2 和 s=5 为例)

本节设计的委员会选举机制实现了随机性、抵御女巫攻击和周期性 3 个目标: 1) 由于委员会选举机制基于确认区块, 而确认区块的产生是随机的, 因此委员会选举机制具有随机性; 2) 在 DAG 区块链中, 为了抵御洪泛攻击和控制区块产生速度, 区块通过 PoW 产生, 这也使得委员会选举机制能够抵抗女巫攻击; 3) 每次时间窗口移动都会产生新的候选区块集合和虚拟链, 相当于重新选举委员会.

算法 3 显示了混合共识协议的具体设计. 算法 3 按轮执行, 每一轮执行包含委员会选举 (第 3–10 行) 和运行 BFT 协议 (第 11–14 行) 两个步骤. 以图 7 为例, 在第 10 轮执行开始时, 时间窗口为 $[q_{20}, q_{21}]$, DAG 区块链中有 9 个区块满足仅被 q_{20} 和 q_{21} 引用的条件, 它们组成了本轮执行的候选区块, 在图 7 中用灰色表示. 随后, 按照哈希值对候选区块排序以形成虚拟链, 取虚拟链上的前 s=5 个区块, 这 5 个区块的创建者就是本轮执行的委员会. 最后, 委员会内部运行 TrustBFT 协议 (或其他 BFT 协议), 在 TrustBFT 协议提交了 T=2 个提案后, 进入下一轮执行. 本轮执行提交的两个提案将被包含在下一轮执行的时间窗口 $[q_{22}, q_{23}]$ 中.

在运行算法 3 之前, 需要设置委员会规模 s 和选举间隔 T. 其中, s 规定了委员会中应该包含多少个节点, 它的大小决定着混合共识协议的安全性和性能, 第 4.3 节详细讨论了 s 的设置对安全性和性能的影响. T 规定了委员会

选举的间隔, 即委员会内部的 BFT 协议每提交 T 个提案就重新进行一次委员会选举. 需要注意的是, T 的设置不能太小, 否则会导致委员会选举过于频繁而影响性能; 同时 T 的设置也不能太大, 否则会导致攻击者有充足的时间去控制委员会. 本节设计的混合共识协议大约每 3–6 h 就重新选举一次委员会, 这一时间间隔既保证了委员会选举不会过于频繁而影响性能, 又保证了攻击者没有足够的时间去控制委员会. 对应的 T 为 1 000–10 000, 其具体值受到 BFT 协议延迟的影响.

由于在第 1 轮执行开始时, DAG 区块链中不存在确认区块, 因此第 1 轮执行的委员会选举无法进行. 为了解决这个问题, 需要以硬编码的方式为混合共识协议提供一个初始委员会. 具体操作如下: 1) 向 DAG 区块链中添加至少 s 个区块; 2) 向共识队列中添加 T 个元素, 且需要保证共识队列至少引用了 s 个区块.

由于委员会内部通过运行 BFT 协议来确认区块, 因此当 BFT 协议的领导者是拜占庭节点时, 可以通过只确认自己或同谋挖出的区块来干扰委员会选举. ElasticDAG 通过定期更换 BFT 协议的领导者节点来解决这一问题: 规定每个视图最多可提交 n 个提案, 即 BFT 协议的领导者最多只能连续提交 n 个提案, 然后强制启动视图切换协议来更换 BFT 协议的领导者. 即使 BFT 协议的当前领导者是拜占庭节点, 它也只能在一段时间内忽略那些诚实节点创建的区块. 最多 n 个提案之后, BFT 协议的领导者会被替换, 新的领导者节点会确认之前被故意忽略的区块. 只要满足委员会选举间隔 $T \gg n$, 那么被拜占庭领导者节点忽略的区块都能在下一轮委员会选举之前被确认, 从而解决这一安全问题.

4.3 委员会内部共识机制

委员会规模 s 是混合共识协议的重要参数, 它的大小在很大程度上决定了混合共识协议的安全性和性能. 本节首先分析了 s 的大小对混合共识协议安全性的影响, 随后讨论了如何在混合共识协议中实现安全性-性能平衡.

(1) 缩小委员会规模

在混合共识协议中, 委员会选举可以视为有替换的随机抽样. 委员会中被攻击者控制的成员数量服从如下二项分布:

$$X \sim B(s, p) \tag{1}$$

其中, X 是随机变量, 表示被攻击者控制的成员数量, p 为一个成员被攻击者控制的概率. 攻击者正好控制 k 个委员会成员的概率可以用公式 (2) 计算:

$$P\{X = k\} = \binom{s}{k} p^k (1-p)^{s-k} \tag{2}$$

攻击者控制的成员数量大于等于 K 的概率可以使用公式 (3) 计算:

$$P\{X \geqslant K\} = \sum_{x=K}^{s} \binom{s}{x} p^x (1-p)^{s-x} \tag{3}$$

全节点被选中的概率与其算力成正比. 对于攻击者, 根据第 2.1 节的威胁模型, $P \leqslant 25\%$, 其中 P 为攻击者控制的算力占系统总算力的比例, 因此有 $p \leqslant 0.25$.

根据公式 (3), 可以发现很难灵活地实现安全性-性能平衡.

1) 从性能的角度考虑, 委员会规模应该尽可能小. 以委员会内部运行 PBFT 协议为例, 随着 s 的增加, 混合共识协议的性能呈指数级下降, 这是因为 PBFT 协议的通信复杂度为 $O(s^2)$. 因此, 为了保证 BFT 协议的性能, 通常有 $s \leqslant 100$, 如在 Haootia 中, $s=46$.

2) 从安全性的角度考虑, 委员会规模应该尽可能大, 以防止攻击者控制委员会. 以 $P=25\%$ 为例, 委员会在 $s=46$ 和 $s=100$ 时的故障概率分别为 8.96% 和 2.76%. 为了达到银行系统级别的安全性, 即故障概率低于 10^{-6}, 需要令 $s \geqslant 646$.

研究指出, 如果可以阻止拜占庭节点向不同节点发送冲突消息, 那么 BFT 协议只需要 $n=2f+1$ 个节点即可容忍 f 个拜占庭节点[34]. 作为对比, 如果拜占庭节点可以向其他节点发送冲突消息, 那么 BFT 协议至少需要 $n=3f+1$ 个节点才能容忍 f 个拜占庭节点. 图 8 显示了 $n=2f+1$ 和 $n=3f+1$ 两种情况下, 使委员会故障概率低于 10^{-6} 所需的

委员会规模. 可以看到, $n=2f+1$ 情况下所需的委员会规模远小于 $n=3f+1$ 的情况, 即使是在 $P=25\%$ 的情况下, 也只需要 79 个节点.

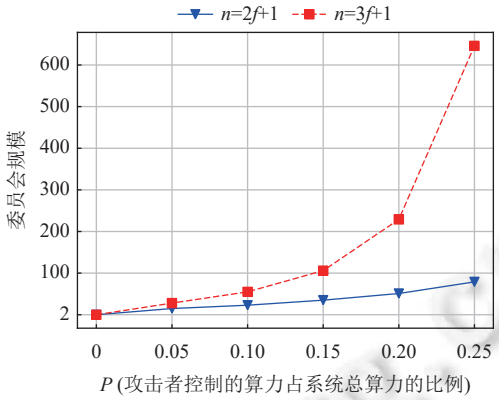


图 8 不同攻击者算力和不同 BFT 协议下使系统故障概率低于 10^{-6} 所需的委员会规模

目前阻止拜占庭节点向其他节点发送冲突消息的主流方法是可信日志方案[30]: 每个节点在本地配置一个由 TEE 实现的可信日志, 该日志会记录节点发出的每一条消息. 如果拜占庭节点向不同节点发送互相冲突的消息, 那么事后可以通过审查日志发现和惩罚这种恶意行为. 然而, 可信日志方案并不适用于 ElasticDAG, 这是因为 ElasticDAG 是一个公有链系统, 所有节点都是匿名的, 不可能知道一个节点的真实身份和真实位置, 更不可能定期审查所有节点的本地日志.

(2) 冲突消息检测机制

如前所述, 提高 BFT 协议容错能力的关键是阻止拜占庭节点向不同节点发送冲突消息的恶意行为. 为此, MinBFT 协议[35]提出了基于 USIG (unique sequential identifier generator) 的冲突消息检测机制. USIG 相当于一个不可篡改的可信顺序计数器, 会为收到的每一条消息分配一个唯一、连续且递增的序号. MinBFT 协议要求每个节点在本地配置一个 USIG, 且每一条合法消息必须附带 USIG 分配的序号和签名. 如果一个节点是诚实的, 那么该节点发送的每一条消息都有唯一、连续且递增的序号. 由于每个节点都会记录其他节点的前一条消息的序号, 因此如果一个恶意节点尝试向不同节点发送互相冲突的消息, 例如两个序号相同但内容不同的提案, 那么一定会有诚实节点检测到其中一条消息的序号不是连续的.

相较于可信日志方案, 基于 USIG 的冲突消息检测机制具有以下优势.

1) 可信日志方案实质上无法阻止拜占庭节点向不同节点发送冲突消息, 只能在本地的日志中记录这种恶意行为. 因此, 只能通过定期审查各个节点的本地日志来发现和惩罚恶意行为. 显然, 如果拜占庭节点是非理智的, 那么可信日志方案无法阻止拜占庭节点的恶意行为. 与之相反, 基于 USIG 的冲突消息检测机制可以直接阻止拜占庭节点向不同节点发送冲突消息, 因此具有更高的安全性.

2) 由于需要定期审查各个节点的本地日志, 可信日志方案只适用于身份和位置公开的联盟链系统. 而基于 USIG 的冲突消息检测机制只要求节点公开公钥, 因此可适用于节点匿名的公有链系统.

(3) TrustBFT 协议

虽然 MinBFT 协议只需要 $n=2f+1$ 个节点就能容忍 f 个拜占庭节点, 但 MinBFT 协议仍存在着以下问题, 导致其难以用于 DAG 区块链混合共识协议.

1) 在 MinBFT 协议中, 每个节点都要广播一条投票消息, 因此其通信开销为 $O(n^2)$(n 为节点数量), 导致其扩展性较差.

2) MinBFT 协议采用一个节点一票的投票规则, 对于区块链中的高算力节点不公平.

3) MinBFT 协议没有考虑委员会轮换.

针对上述问题, 本节提出了 TrustBFT 协议. 算法 4 和图 9 描述了 TrustBFT 协议的具体设计.

算法 4. TrustBFT 协议 (针对 $node_r$).

1. **for** *view* ← 1, 2, 3,… **do**
2. L ← Leader(*view*)
   //准备阶段
3. **if** $r == L$ **then** //$node_r$ 是当前视图的领导者
   //s_r 和 UI_r 分别是 $node_r$ 的身份和本地 USIG 分配的序号, C_i 是本轮共识的提案
4. broadcast message <Prepare, *view*, s_r, C_i, UI_r> and its signature
5. **else** //$node_r$ 不是当前视图的领导者
6. wait for message: m ← <Prepare, *view*, s_L, C_i, UI_L> from $node_L$
7. **if** $m.v == view$ and $C_i.refs$ are all in DAG blockchain and $m.UI$ is valid **then**
8. send message <PreCommit, *view*, s_r, $H(C_i)$, UI_r> and its signature to $node_L$
9. **end**
10. **end**
    //提交阶段
11. **if** $r == L$ **then**
12. wait for $f+1$ messages: M ← {m | matching message < PreCommit, *view*, s, $H(C_i)$, UI>}
13. qc ← QC(M)//将 f+1 条消息中的签名聚合为单个单个签名
14. broadcast message <Commit, *view*, $H(C_i)$, qc>//广播聚合签名
15. commit C_i
16. **else**
17. wait for message: m ← <Commit, *view*, $H(C_i)$, qc>
18. **if** $m.qc$ is valid **then**
19. commit C_i
20. **end**
21. **end**
22. **end**

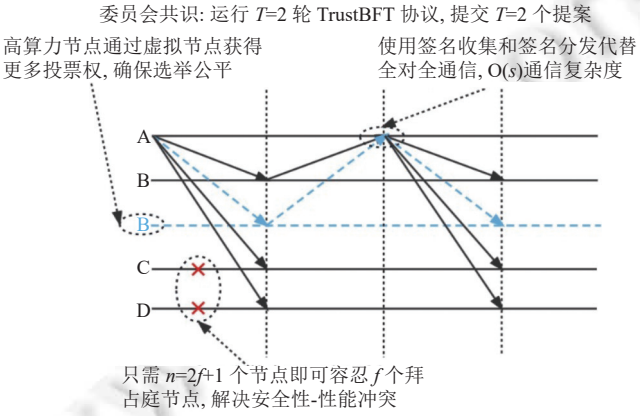


图 9 TrustBFT 委员会共识协议运行流程 (以 s=5 为例)

TrustBFT 协议分为准备阶段和提交阶段. 在准备阶段, 领导者节点广播包含新提案的 Prepare 消息. 其他节点收到 Prepare 消息后, 缓存 Prepare 消息和其中的提案, 然后向领导者节点发送一条签名的 PreCommit 消息作为投

票. 在提交阶段, 领导者节点等待 f+1 条 PreCommit 消息, 然后基于门限签名技术[36]将 f+1 条 PreCommit 消息的签名聚合为单个签名, 并广播包含聚合后的签名的 Commit 消息. 其他节点收到 Commit 消息后, 验证其中的聚合签名, 如果验证通过, 则提交提案. 从图 9 中可以发现, 通过将提交阶段的投票广播拆分为投票收集和签名广播两个步骤, TrustBFT 协议将通信复杂度从 O(n^2) 降低为 O(n), 有效提高了可扩展性.

如果检测到领导者故障, TrustBFT 协议通过视图切换协议更换领导者. TrustBFT 协议的视图切换协议与 MinBFT 协议相同.

此外, 为了提高公平性, TrustBFT 协议时还引入了虚拟节点的概念. 具体而言, 如果 n 个 (n>1) 被选中的候选区块具有相同的创建者, 那么它们的创建者在 TrustBFT 协议中相当于 n 个节点. 这 n 个节点中包含 1 个真实节点和 n–1 个虚拟节点, 真实节点的投票被看作是它自己和 n–1 个虚拟节点共同的投票, 因此被记作 n 票. 以图 7 和图 9 为例, 008a 和 009b 的创建者都是 B, 在运行 TrustBFT 协议时, B 被视作 2 个节点, 它的投票被记作 2 票.

对于任意诚实的委员会成员 $node_r$, 如果 $node_r$ 提交了提案 q_i, 那么 $node_r$ 一定收到过 q_i 的 Commit 消息. Commit 消息中包含不可伪造的聚合签名, 因此 Commit 消息可作为提交 q_i 的证明. $node_r$ 会将 q_i 添加进共识队列, 并广播 q_i 和 q_i 的 Commit 消息. 其他节点收到后, 可确定 q_i 一定被 BFT 协议提交. 因此, 所有诚实节点最终都能对被提交提案达成一致, 从而对共识队列达成一致. 对于任意确认区块 b, 因为 b 是确认区块, 所以共识队列中一定存在直接或间接引用过 b 的提案 q_i. 由于 q_i 被提交, 因此至少有 f+1 个委员会成员对 q_i 投票, 其中至少包含 1 个诚实的委员会成员. 诚实的委员会成员在对 q_i 投票前, 会检查本地 DAG 区块链副本中是否包含 q_i 引用的所有区块. 因此, 一定有诚实的委员会成员创建或收到过 q_i 引用的所有区块. 由于诚实节点会转发自己创建或收到的区块, 因此最终所有节点都能收到 q_i 引用的所有区块. 因此, 最终所有诚实节点都能对确认区块集合达成一致. 由于确认区块分区排序算法是一个确定性算法, 在输入 (共识队列和确认区块集合) 相同的情况下会产生相同的输出, 因此最终所有节点都会得到相同的确认区块排序结果, 从而实现顺序一致性.

在当前委员会 (旧委员会) 提出 T 个提案之后, 其中的诚实节点会停止产生新提案或对新提案投票. 设提案和 Commit 消息的广播延迟上限为 Δ, 委员会选举机制产生的新委员会首先等待 Δ 时间, 从而确保新委员会中的节点一定能够收到旧委员会提交的提案和相应的 Commit 消息. 然后, 新委员会中的节点广播包含本地共识队列的 Committee 消息 (通过引入检查点机制, 实际上只需要广播检查点之后的共识队列元素), 新委员会的领导者节点收到 f+1 条 Committee 消息后, 广播包含 f+1 条 Committee 消息的 NewCommittee 消息. 广播或收到 NewCommittee 消息后, 提交其中包含的提案, 然后新委员会开始运行. 由于 NewCommittee 消息包含 f+1 条 Committee 消息, 因此其中至少有一条 Committee 消息来自诚实节点. 通过等待 Δ 时间, 诚实节点一定收到了旧委员会提交的所有提案, 因此 NewCommittee 消息一定包含旧委员会提交的所有提案, 从而保证了委员会轮换具有一致性.

ElasticDAG 的共识机制允许节点自由地加入或离开, 下面予以分析. 委员会之外的节点不会直接参与共识, 因此它们的加入和离开不会影响共识机制的运行. 委员会内部运行 BFT 协议, 本身具有一定的容错能力, 因此少量节点的离开不会影响委员会的运行. 如果出现大量节点离开委员会, 导致委员会无法运行的小概率情况, 那么可重新组建委员会. 具体而言, 可设置一个较长的超时时间, 如果委员会一直不能提出新提案, 那么可基于虚拟链重新组建委员会, 如选择虚拟链的第 s+1, s+2,…, 2s 个区块的创建者作为新的委员会.

5 安全性分析

区块链系统必须满足安全性 (safety) 和活性 (liveness) 两个基本性质. 前者确保区块链中的诚实节点能够达成一致, 后者确保合法的交易最终能被区块链记录和确认. 安全性和活性的定义如下.

定义 1. 安全性. 本文使用 SMR 算法处理交易, 其对安全性的要求如下: 1) 顺序一致性: 所有诚实节点以相同的顺序提交相同的交易; 2) 有效性 (validity): 如果一个诚实的节点提交了一个交易, 那么这个交易肯定由某个诚实的节点提出.

定义 2. 活性. 节点发出的交易能在有限时间内被提交.

定理 1. 顺序一致性. 如果诚实 (正确) 节点 $node_i$ 以序号 h 确认 (提交) 了交易 tx, 那么所有诚实节点都会以相

同的序号 h 确认 (提交) tx.

证明: 如果交易 tx 被 $node_i$ 确认, 那么 tx 所在的区块一定被共识队列中的某个元素 q_j 直接或间接引用. 由于 q_j 是委员会通过 BFT 协议产生的, 因此只要委员会满足安全假设, 那么委员会内部一定已经对 q_j 达成一致, 且拜占庭节点无法伪造 q_j. 由于所有诚实节点在收到 tx 和 q_j 后都会转发 tx 和 q_j (包括 $node_i$), 因此最终所有诚实节点都会收到 tx 和 q_j. 由于 q_j 直接或间接引用了 tx 所在的区块, 因此 tx 一定会被所有诚实节点确认.

因为共识队列中的元素无法被伪造, 所以最终所有诚实节点都会对共识队列和共识队列所引用的区块达成一致. 由于算法 2 是确定性算法, 在输入 (共识队列和确认区块) 相同的情况下一定会产生相同的输出, 因此所有诚实节点都能对确认区块的顺序达成一致, 并为其中的交易分配相同的序号.

定理 2. 有效性. 如果诚实 (正确) 节点 $node_i$ 确认 (提交) 了交易 tx, 那么 tx 一定由某个诚实节点提出.

证明: 在拜占庭环境下, 有效性依靠诸如密码学哈希、数字签名等攻击者无法打破的标准密码学假设保证. 每个交易包含发起交易发起者的签名, 因此攻击者无法伪造或篡改交易.

定理 3. 活性. 诚实节点发出的交易最终会被确认.

证明: 根据第 2 节的网络模型, 轻节点广播的交易可以在有限时间内传播到所有全节点, 即使拜占庭节点故意忽略一笔交易, 也会有诚实的全节点将这笔交易打包进新区块. 根据网络模型, 区块可以在有限时间内传播到所有诚实的全节点, 因此合法的交易/区块一定能在有限时间内加入 DAG 区块链.

如果委员会中的 BFT 协议领导者是诚实的, 那么会周期性地广播提案和发起共识. 即使领导者发生故障, 也可以在有限时间内通过视图切换协议更换领导者. 因此, 在有限的时间内, 一定会有新的元素被添加到共识队列.

对于最长链上的未确认区块, 它一定会在有限时间内被未来新产生的共识队列元素确认. 对于分叉链上的未确认区块, 每轮共识都会随机确认最多 $N_{\text{chain}}\times R$ 条分叉链, 只要 $R>dF$, 那么所有分叉链都能在有限时间内得到确认. 其中, R 为预先设置的系统参数, d 为 BFT 协议每轮共识的平均间隔, F 为产生分叉链的平均频率.

6 实验分析

本节通过实验验证 ElasticDAG 的性能. 首先测试了不同存储模型设计下的吞吐量, 证明了 ElasticDAG 的存储模型设计可有效提高吞吐量; 然后测试了不同 SGX 操作的时间开销; 接下来测试并分析了 TrustBFT 协议的性能, 证明了 ElasticDAG 实现了安全且高效的委员会内部共识; 最后测试了 ElasticDAG 的确认延迟和吞吐量, 证明 ElasticDAG 实现了高吞吐和低延迟.

6.1 实验设置

本文的实验在 Windows Server 2019 操作系统中进行. ElasticDAG 协议的主要部分使用 Golang 开发, USIG 部分使用 Intel SGX 和 OpenEnclave 开发, 聚合签名为 kyber 提供的 bls 签名. 本文假设 USIG 会在内部产生一对 RSA 密钥对, 并通过 Intel 提供的远程认证服务安全的交换公钥. 此外, SGX 需要结合其他手段才能抵御回滚攻击[37]. 在 SGX 相关的实验上, 本文的实验设置类似于 AHL[30], 即首先在一台支持 SGX 的 CPU 上测试每个 SGX 操作的延迟, 然后在系统测试中通过人工加入延迟的方式模拟 SGX 操作的开销. 本文在 Google Cloud 平台的 16 个虚拟机上测试 TrustBFT 协议和 ElasticDAG 原型系统的性能, 每个虚拟机包含 8 个 vCPU 和 32 GB 内存, 每台虚拟机最多运行 10 个 ElasticDAG 全节点. 为了模拟真实网络环境, 所有虚拟机被部署到世界范围内的 16 个不同区域. 虚拟机之间通过 ping 命令测得的平均网络延迟为 49.8–365.6 ms. 每个 ElasticDAG 节点拥有 24 Mb/s 的上行和下行带宽.

在第 6.5 节和第 6.6 节的实验中, 区块大小固定为 300 KB, 由随机产生的平均大小为 0.5 KB 的支付交易填充, 整个系统以平均每秒 5 个区块的速度产生新区块. 为了方便, 每个节点同时充当全节点和轻节点.

6.2 存储模型对比分析

本节对比了不同存储模型设计对性能的影响, 结果如图 10 所示. 在本节的实验中, 区块大小固定为 100 KB, 系统平均每秒产生 10 个区块. 为了方便区分, 本节中将已确认但未排序的区块称为部分确认区块, 将已确认且已排序的区块称为确认区块.

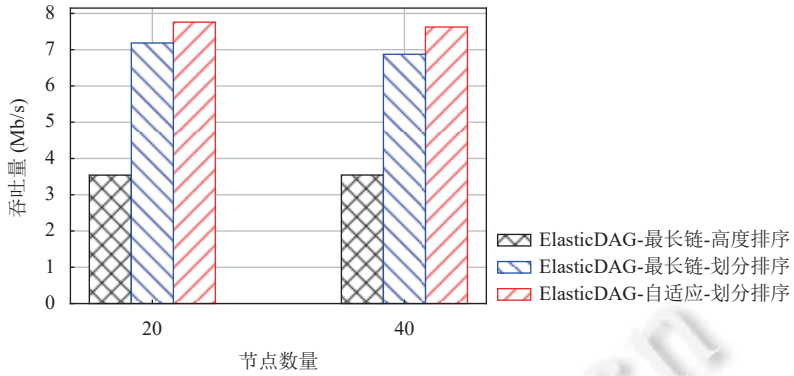


图 10 不同存储模型设计在不同节点数量下的吞吐量

本节对比了 3 种存储模型设置. 其中, ElasticDAG-最长链-高度排序设置表示仅确认最长链区块, 同时基于区块高度和区块链实例编号对确认区块排序: 对于高度为 h 的部分确认区块 b, 如果所有区块链实例中均存在高度为 h 的部分确认区块, 那么可以对 b 进行排序, 排序后的 b 成为确认区块. 排序的规则是首先根据高度排序, 如果高度相同, 则根据区块所属的区块链实例编号排序. ElasticDAG-最长链-划分排序设置表示仅确认最长链区块, 同时使用第 3 节设计的基于划分的确认区块排序算法. ElasticDAG-自适应-划分排序设置表示使用第 3 节设计的自适应区块确认策略和基于划分的确认区块排序算法.

ElasticDAG-最长链-高度排序设置的吞吐量仅为理论吞吐量 (1 000 KB/s 或 7.8 Mb/s) 的 45%. 这主要是因为这一设置下的区块排序算法难以适应区块链实例长度不平衡的情况. 具体而言, 假设在第 i 个区块链实例中, 部分确认区块的最高高度为 L_i, 取 $L=\min\{L_i \mid i=0, 1,\ldots, N_{\text{chain}}\}$, 那么该设置下的确认区块排序算法此时只能对所有高度等于或低于 L 的部分确认区块排序, 其余高度高于 L 的部分确认区块不能被排序, 只能继续等待. 由于区块创建过程中的随机性, 不同区块链实例的长度是不平衡的, 这导致大量部分确认区块不能被排序, 进而严重影响吞吐量.

ElasticDAG-最长链-划分排序设置在 20 个节点和 40 节点下分别实现了 92% 和 88% 的理论吞吐量, 大约是 ElasticDAG-最长链-高度排序设置的两倍. 这是因为算法 2 能够在区块链实例长度不平衡的情况下直接对所有部分确认区块排序, 因此有效提高了吞吐量.

ElasticDAG-自适应-划分排序设置将分叉区块也被纳入共识, 因此进一步提高了吞吐量, 在 20 个节点和 40 节点下分别实现了 99.3% 和 97.6% 的理论吞吐量.

后文的实验将主要集中于对比共识协议设计对性能的影响, 除特殊说明之外, 存储模型将默认采用 ElasticDAG-自适应-划分排序设置.

6.3 USIG 开销分析

本节在 Intel XEON E-2288G CPU 上测量了 SGX 模式下, SHA256 算法, RSA 签名、RSA 签名验证和 USIG 的运行时开销. 表 3 详细说明了每个操作的运行时开销. 对于每个操作, 本文测试了其 1 000 次运行的平均时间. 可以看到, SHA256 的开销很低, RSA 签名和签名验证的开销相对较高, USIG 的主要开销为 RSA 签名, 所有操作的时间开销均小于 1 ms. 在后文的实验中, 本文通过插入测量到的时间延迟来模拟 SGX 操作.

6.4 TrustBFT 协议对比分析

本节对比了 TrustBFT 协议和线性 PBFT (linear PBFT) 协议[38]在不同节点数量下, 对一个提案达成共识的平均时间开销, 结果如图 11 所示. 线性 PBFT 协议是 PBFT 协议的改进版本, 它通过聚合签名技术将通信开销从 $O(s^2)$ 降低到 $O(s)$. 根据第 4.3 节的计算, 如果使用 TrustBFT 协议作为委员会内部的共识协议, 那么委员会只需要包含 79 个节点即可确保故障概率低于 10^{-6}. 因此, 本节在最多 79 个节点下进行了测试. 在第 6.5 节和第 6.6 节的实验中, 当全节点数量小于或等于 79 时, 所有全节点均加入委员会; 当全节点数量大于 79 时, 委员会的规模固定为 79.

实验结果显示, TrustBFT 协议的延迟明显低于线性 PBFT 协议, 这是因为: 1) TrustBFT 协议只需要等待超过

一半节点的回复, 而线性 PBFT 需要等待超过 2/3 节点的回复; 2) TrustBFT 协议只有 1 轮投票, 而线性 PBFT 协议有 2 轮投票.

表 3 SGX 操作的运行时开销

操作	平均时间 (μs)
SHA256	11.4
RSA签名	485.9
RSA签名验证	17.9
USIG	510.1

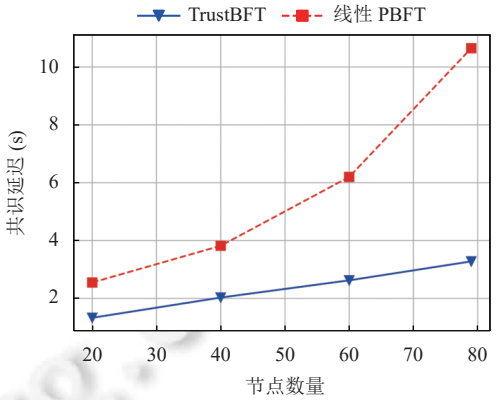


图 11 TrustBFT 协议和线性 PBFT 协议在不同节点数量下的延迟

在节点数量为 79 时, 线性 PBFT 协议和 TrustBFT 协议的平均延迟分别为 10.6 s 和 3.3 s. 此外, 由于 TrustBFT 协议只需 $n=2f+1$ 个节点就可以容忍 f 个故障节点, 因此 TrustBFT 协议具备更强的容错能力. 在委员会规模为 79 时, 采用线性 PBFT 协议 (以及其他要求 $n=3f+1$ 的 BFT 协议) 的故障概率为 4.3%, 采用 TrustBFT 的故障概率为 0.000 086%. 可以看到, 在相同节点数量下, TrustBFT 协议不仅具有更低的延迟, 还具有更强的容错能力.

与 Haootia 相比, ElasticDAG 的委员会内部共识更加高效和安全. Haootia 的委员会规模最大为 46, 使用 PBFT 协议作为委员会内部的共识协议. 从性能的角度考虑, PBFT 协议在 46 个节点时的延迟约为 4 s, 略高于 TrustBFT 协议在 79 个节点时的延迟. 考虑到 TrustBFT 协议和 PBFT 协议的通信复杂度分别为 $O(s)$ 和 $O(s^2)$, 因此 ElasticDAG 在委员会内部共识上更加高效. 从安全性的角度考虑, ElasticDAG 的委员会被攻击者控制的概率可以忽略不计, 而 Haootia 的委员会被攻击者控制的概率很大. 例如当攻击者控制着 25% 的算力时, ElasticDAG 和 Haootia 的委员会被攻击者控制的概率分别为 0.000 086% 和 8.96%.

6.5 确认延迟分析

图 12 对比了 ElasticDAG 原型系统在不同共识协议和不同节点数量下的确认延迟. 其中, ElasticDAG-线性 PBFT 表示采用混合共识协议, 且委员会内部通过线性 PBFT 协议达成共识; ElasticDAG-TrustBFT 表示采用混合共识协议, 且委员会内部通过 TrustBFT 协议达成共识; ElasticDAG-PoW 表示节点通过 PoW 和最长链法则达成一致, 相当于在 DAG 区块链之外额外运行一条 PoW 区块链, 为了方便, 后文中将这条区块链称为共识链. 在 ElasticDAG-PoW 中, 节点通过 PoW, 以 10 s 的平均时间间隔向共识链添加区块, 每个共识链区块在创建时引用区块确认策略返回的区块, 当 1 个共识链区块得到 k 个确认时, 该区块成为确认区块, 并被加入共识队列. 为了获得足够低的确认错误概率, 本节将 k 设置为 30.

作为对比, 图 12 还显示了 OHIE 的确认延迟, 其设置如下: 区块大小 20 KB, 运行 750 个区块链实例, 单个实例中平均 10 s 产生一个区块, 一个区块在得到 30 个确认之后成为部分确认区块.

对于 ElasticDAG 原型系统, 随着节点数量的增加, 确认延迟有所增加, 但总体上呈收敛趋势. 由于概率性共识需要等待足够长的时间才能保证确认错误概率足够低, 因此在确认延迟方面, 使用概率性共识协议的 DAG 区块链明显高于使用确定性共识协议的 DAG 区块链. 例如当节点数量为 160 时, ElasticDAG-PoW 和 OHIE 的确认延迟分别为 347 s 和 560 s, 而对于混合共识协议, ElasticDAG-TrustBFT 和 ElasticDAG-线性 PBFT 的延迟分别为 32 s 和 43 s.

尽管都是等待 30 个确认, 但 OHIE 的确认延迟仍然比 ElasticDAG-PoW 高 200 s 以上. 这是因为 OHIE 中的区

块在得到 30 个确认后只能成为部分确认区块, 只有当其他区块链实例中都存在虚拟高度相同或更高的部分确认区块时, 该区块才能得到最终的确认和排序.

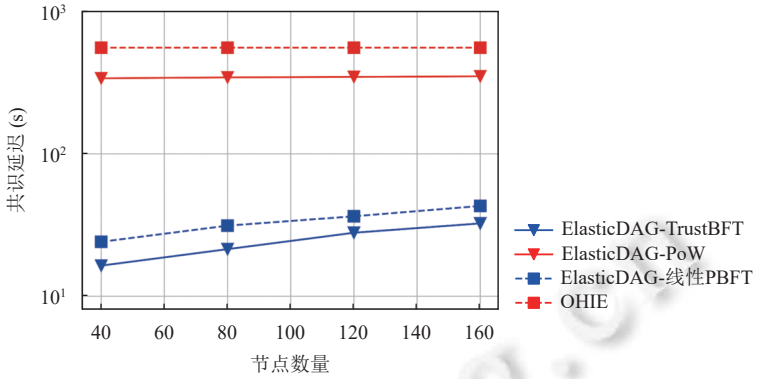


图 12 不同节点数量下的确认延迟对比

在使用混合共识协议时, 委员会内部采用 TrustBFT 协议可以将确认延迟降低 8–10 s, 这是因为 TrustBFT 协议可以提高委员会内部共识的效率 (第 6.4 节).

6.6 吞吐量分析

图 13 显示了 ElasticDAG 原型系统在不同设置和节点数量下的吞吐量, 作为对比, 图 13 还显示了 OHIE 的吞吐量. 从实验中可以发现, 在 ElasticDAG 系统中, 虽然共识协议对确认延迟的影响很大, 但共识协议对吞吐量却几乎没有任何影响, 例如, 当节点数量为 120 时, ElasticDAG-TrustBFT、ElasticDAG-线性 PBFT 和 ElasticDAG-PoW 的吞吐量分别为 11.21 Mb/s、11.16 Mb/s 和 11.23 Mb/s. 造成这一现象的原因是 ElasticDAG 对 DAG 区块链的存储模型和共识协议进行了解耦, 共识协议的设计只会影响向共识队列添加元素的速度, 而不会对存储模型产生任何影响, 也不会影响吞吐量.

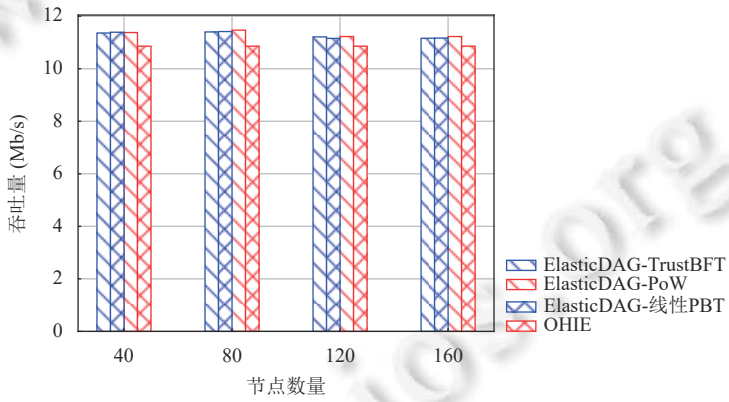


图 13 不同节点数量下的吞吐量对比

随着节点数量的不断增加, ElasticDAG 原型系统和 OHIE 的吞吐量基本保持不变, 这证明了 ElasticDAG 具有良好的可扩展性. 此外, 在实验过程中, 本文发现分叉区块大约占总区块数量的 10%–15%, 因此如果不确认分叉区块, 那么采用平行链 DAG 拓扑的系统会损失掉这部分吞吐量.

在节点数量为 160 时, 按照从左到右的顺序, 图 13 中的吞吐量分别为 11.16 Mb/s、11.17 Mb/s、11.22 Mb/s 和 10.86 Mb/s. 可以看到, ElasticDAG 在确认延迟具有明显优势的同时, 取得了和 OHIE 相当的吞吐量. 假设每笔交易的平均大小和比特币交易相同, 即 0.5 KB, 那么 ElasticDAG-TrustBFT、ElasticDAG-线性 PBFT 和 ElasticDAG-PoW 的交易吞吐量分别为 2 857 tps、2 859 tps 和 2 874 tps, 这已经足以满足 VISA 等应用的需求, 并与其他 DAG

区块链工作中报告的吞吐量相当[10,12].

6.7 委员会轮换分析

后文图 14 报告了 ElasticDAG 原型系统在不同委员会规模下的委员会轮换开销. 实验设置如下: 委员会内部运行 TrustBFT 协议, 设置委员会选举间隔 T=20, 对于每个委员会规模, 测试了 10 次委员会轮换的平均时间开销. 从实验中可以发现, 随着委员会规模的增加, 委员会轮换的时间开销迅速上升, 这主要是因为在委员会轮换过程中, 每个节点都要广播 Committee 消息, 从而导致 $O(s^2)$ 的通信复杂度.

本文在委员会选举间隔的设置上参考了 OmniLedger[39], 大约每 3–6 h 执行一次委员会轮换. 虽然 ElasticDAG 在委员会轮换期间必须停止确认区块和交易, 但由于委员会轮换在实际运行过程中的频率很低, 因此对系统性能的影响很小. 例如, 当委员会规模为 79 时, 委员会轮换的时间开销为 46.2 s, 仅占总运行时间的 0.43%–0.21%.

6.8 交易执行分析

图 15 报告了 ElasticDAG 原型系统在不同 CPU 数量下的最大交易执行速率. 实验设置如下: 1) 所有交易均为比特币支付交易, 并包含完整的密码学操作; 2) 为了保证结果的一致性, 所有修改账户的操作均为原子操作; 3) 不断提高交易执行速率, 直至系统极限; 4) 对于每种 CPU 数量, 进行了 10 min 的测试, 并取其平均值作为最终结果.

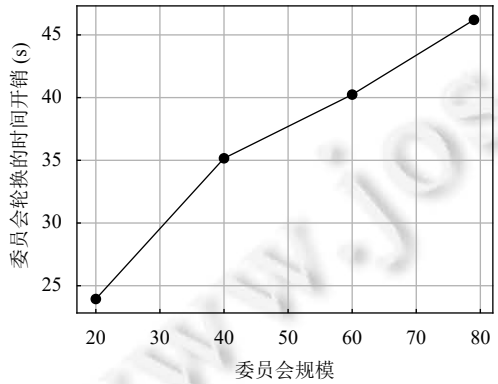


图 14 不同节点数量下的委员会轮换开销

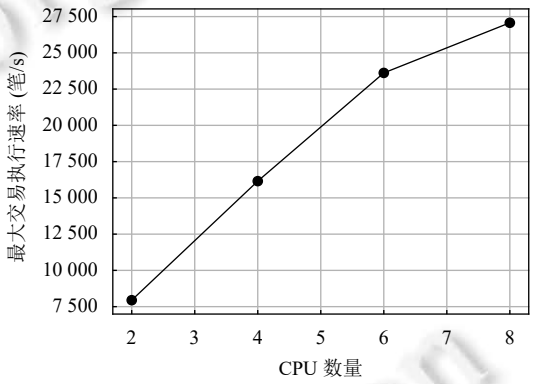


图 15 不同 CPU 数量下的最大交易执行速率

从实验中可以发现, 随着 CPU 数量的增加, 最大交易执行速率不断提高. 这主要是因为交易执行的主要开销为密码学操作 (如验证签名), 而密码学操作可以并行执行, 因此可以通过提高并行度来提高系统的交易执行速率. 此外, 还可以发现在目前的情况下, 交易执行不会成为系统性能瓶颈, 即使当 CPU 数量为 2 时, 每秒也能执行约 7900 笔交易, 而当 CPU 数量为 8 时, 每秒可执行约 27000 笔交易. 综上所述, 网络和共识协议才是区块链目前的主要性能瓶颈, 且即使交易执行存在性能瓶颈, 也可通过增强硬件性能、提高程序并行度等方式进行解决.

7 总 结

本文提出了可灵活扩展的弹性 DAG 区块链系统 ElasticDAG. 在 ElasticDAG 中, 通过将存储模型和共识机制进行解耦, 两者以并行方式运行, 提高了系统面向不同应用需求的性能和灵活性. 针对平行链 DAG 拓扑中抛弃分叉区块带来的吞吐量损失和活性问题, 本文设计了自适应区块确认策略和基于划分的确认区块排序算法. 接下来, 为了降低确认延迟, 本文提出了一种低延迟 DAG 区块链混合共识协议. 针对委员会选举, 本文提出了一种基于虚拟链的委员会选举机制, 解决了因算力稀释导致的安全性问题. 针对委员会内部共识, 本文提出了容错能力更强的 TrustBFT 协议作为委员会内部共识协议, 在不影响安全性的前提下缩小委员会规模, 从而实现安全性-性能平衡. 最后, 本文实现了一个 DAG 区块链系统原型, 实验结果显示, ElasticDAG 在为交易提供全序和确定性共识的基础上实现了高吞吐和低延迟, 能够满足多元化应用需求.

目前 ElasticDAG 处于原型开发阶段, 未来将进一步基于 ElasticDAG, 针对故障处理机制和对通信机制展开研

究. 此外, ElasticDAG 在提高交易确认效率的同时可能导致交易执行性能难以匹配, 成为制约整体系统性能的瓶颈问题, 未来研究将通过并行化提高交易执行效率.

References:

[1] Nakamoto S. Bitcoin: A peer-to-peer electronic cash system. 2008. https://bitcoin.org/bitcoin.pdf

[2] Jin H, Xiao J. Towards trustworthy blockchain systems in the era of “Internet of value”: Development, challenges, and future trends. Science China Information Sciences, 2022, 65(5): 153101. [doi: 10.1007/s11432-020-3183-0]

[3] Xia Q, Dou WS, Guo KW, Liang G, Zuo C, Zhang FJ. Survey on blockchain consensus protocol. Ruan Jian Xue Bao/Journal of Software, 2021, 32(2): 277–299 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/6150.htm [doi: 10.13328/j.cnki.jos.006150]

[4] 2020 mobile Internet blue book: Blockchain infrastructure has become the focus of industrial development (in Chinese). 2020. http://blockchain.people.com.cn/n1/2020/0715/c417685-31784959.html

[5] Gao ZF, Zheng JL, Tang SY, Long Y, Liu ZQ, Liu Z, Gu DW. State-of-the-art survey of consensus mechanisms on DAG-based distributed ledger. Ruan Jian Xue Bao/Journal of Software, 2020, 31(4): 1124–1142 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/5982.htm [doi: 10.13328/j.cnki.jos.005982]

[6] Ethereum whitepaper. 2020. https://github.com/ethereum/wiki/wiki/White-Paper

[7] Uniswap v3 Core. 2021. https://uniswap.org/whitepaper-v3.pdf

[8] The tangle. 2017. http://cryptoverze.s3.us-east-2.amazonaws.com/wp-content/uploads/2018/11/10012054/IOTA-MIOTA-Whitepaper.pdf

[9] Sompolinsky Y, Lewenberg Y, Zohar A. Spectre: A fast and scalable cryptocurrency protocol. IACR Cryptology ePrint: 1159, 2016.

[10] Li CX, Li PL, Zhou D, Yang Z, Wu M, Yang G, Xu W, Long F, Yao ACC. A decentralized blockchain with high throughput and fast confirmation. In: Proc. of the 2020 USENIX Annual Technical Conf. USENIX Association, 2020. 515−528.

[11] Bagaria V, Kannan S, Tse D, Fanti G, Viswanath P. Prism: Deconstructing the blockchain to approach physical limits. In: Proc. of the 2019 ACM SIGSAC Conf. on Computer and Communications Security. London: Association for Computing Machinery, 2019. 585−602. [doi: 10.1145/3319535.3363213]

[12] Yu HF, Nikolić I, Hou RM, Saxena P. OHIE: Blockchain scaling made simple. In: Proc. of the 2020 IEEE Symp. on Security and Privacy. San Francisco: IEEE, 2020. 99−105. [doi: 10.1109/sp40000.2020.00008]

[13] Keidar I, Kokoris-Kogias E, Naor O, Spiegelman A. All you need is DAG. In: Proc. of the 2021 ACM Symp. on Principles of Distributed Computing. Salerno: Association for Computing Machinery, 2021. 165–175. [doi: 10.1145/3465084.3467905]

[14] Danezis G, Kokoris-Kogias L, Sonnino A, Spiegelman A. Narwhal and Tusk: A DAG-based mempool and efficient BFT consensus. In: Proc. of the 17th European Conf. on Computer Systems. Rennes: Association for Computing Machinery, 2022. 34–50. [doi: 10.1145/3492321.3519594]

[15] Byteball: A decentralized system for storage and transfer of value. 2016. https://obyte.org/Byteball.pdf

[16] Bracing a transaction DAG with a backbone chain. 2020. https://eprint.iacr.org/eprint-bin/getfile.pl?entry=2020/472&version=20200629:173052&file=472.pdf

[17] Speed-security tradeoffs in blockchain protocols. 2015. https://eprint.iacr.org/2015/1019.pdf

[18] Miller A, Xia Y, Croman K, Shi E, Song D. The honey badger of BFT protocols. In: Proc. of the 2016 ACM SIGSAC Conf. on Computer and Communications Security. London: Association for Computing Machinery, 2016. 31−42. [doi: 10.1145/2976749.2978399]

[19] Kokoris-Kogias E, Jovanovic P, Gailly N, Khoffi I, Gasser L, Ford B. Enhancing bitcoin security and performance with strong consistency via collective signing. In: Proc. of the 25th USENIX Conf. on Security Symp. Austin: USENIX Association, 2016. 279−296.

[20] Castro M, Liskov B. Practical Byzantine fault tolerance. In: Proc. of the 3rd Symp. on Operating Systems Design and Implementation. New Orleans: USENIX Association, 1999. 173–186.

[21] Amiri MJ, Agrawal D, El Abbadi A. SharPer: Sharding permissioned blockchains over network clusters. In: Proc. of the 2021 Int’l Conf. on Management of Data. Xi’an: Association for Computing Machinery, 2021. 76–88. [doi: 10.1145/3448016.3452807]

[22] Chatzopoulos D, Gujar S, Faltings B, Hui P. Mneme: A mobile distributed ledger. In: Proc. of the 2020 IEEE Conf. on Computer Communications. Toronto: IEEE, 2020. 1897–1906. [doi: 10.1109/infocom41043.2020.9155497]

[23] Yin MF, Malkhi D, Reiter MK, Gueta GG, Abraham I. HotStuff: BFT consensus with linearity and responsiveness. In: Proc. of the 2019 ACM Symp. on Principles of Distributed Computing. Toronto: Association for Computing Machinery, 2019. 347–356. [doi: 10.1145/3293611.3331591]

[24] Wang RH, Zhang LF, Xu QQ, Zhou H. Byzantine fault tolerance algorithm for consortium blockchain. Application Research of Computers, 2020, 37(11): 3382–3386 (in Chinese with English abstract). [doi: 10.19734/j.issn.1001-3695.2019.07.0268]

[25] van Renesse R, Dumitriu D, Gough V, Thomas C. Efficient reconciliation and flow control for anti-entropy protocols. In: Proc. of the 2nd Workshop on Large-Scale Distributed Systems and Middleware. New York: Association for Computing Machinery, 2008. 6. [doi: 10.1145/1529974.1529983]

[26] Wang J, Fan CY, Cheng YQ, Zhao B, Wei T, Yan F, Zhang HG, Ma J. Analysis and research on SGX technology. Ruan Jian Xue Bao/Journal of Software, 2018, 29(9): 2778–2798 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/5594.htm [doi: 10.13328/j.cnki.jos.005594]

[27] Douceur JR. The sybil attack. In: Proc. of the 1st Int'l Workshop on Peer-to-peer Systems. Cambridge: Springer, 2002. 251−260. [doi: 10.1007/3-540-45748-8_24]

[28] Sapirshtein A, Sompolinsky Y, Zohar A. Optimal selfish mining strategies in bitcoin. In: Proc. of the 20th Int'l Conf. on Financial Cryptography and Data Security. Christ Church: Springer, 2017. 515−532. [doi: 10.1007/978-3-662-54970-4_30]

[29] Eyal I, Gencer AE, Sirer EG, van Renesse R. Bitcoin-NG: A scalable blockchain protocol. In: Proc. of the 13th USENIX Conf. on Networked Systems Design and Implementation. Santa Clara: USENIX Association, 2016. 45−59.

[30] Dang H, Dinh TTA, Loghin D, Chang EC, Lin Q, Ooi BC. Towards scaling blockchain systems via sharding. In: Proc. of the 2019 Int'l Conf. on Management of Data. Amsterdam: Association for Computing Machinery, 2019. 123–140. [doi: 10.1145/3299869.3319889]

[31] Kiayias A, Russell A, David B, Oliynykov R. Ouroboros: A provably secure proof-of-stake blockchain protocol. In: Proc. of the 37th Annual Int'l Cryptology Conf. Santa Barbara: Springer, 2017. 357−388. [doi: 10.1007/978-3-319-63688-7_12]

[32] Zhou T, Li XF, Zhao H. DLattice: A permission-less blockchain based on DPoS-BA-DAG consensus for data tokenization. IEEE Access, 2019, 7: 39273–39287. [doi: 10.1109/ACCESS.2019.2906637]

[33] Merkle RC. A digital signature based on a conventional encryption function. In: Proc. of the 1988 CRYPTO. Berlin, Heidelberg: Springer, 1988. 369–378. [doi: 10.1007/3-540-48184-2_32]

[34] Correia M, Neves NF, Verissimo P. How to tolerate half less one Byzantine nodes in practical distributed systems. In: Proc. of the 23rd IEEE Int'l Symp. on Reliable Distributed Systems. Florianopolis: IEEE, 2004. 174–183. [doi: 10.1109/RELDIS.2004.1353018]

[35] Veronese GS, Correia M, Bessani AN, Lung LC, Verissimo P. Efficient Byzantine fault-tolerance. IEEE Trans. on Computers, 2013, 62(1): 16–30. [doi: 10.1109/TC.2011.221]

[36] Boneh D, Lynn B, Shacham H. Short signatures from the Weil pairing. In: Proc. of the 7th Int'l Conf. on the Theory and Application of Cryptology and Information Security. Gold Coast: Springer, 2001. 514–532. [doi: 10.1007/3-540-45682-1_30]

[37] Matetic S, Ahmed M, Kostiainen K, Dhar A, Sommer DM, Gervais A, Juels A, Capkun S. ROTE: Rollback protection for trusted execution. In: Proc. of the 26th USENIX Security Symp. Vancouver: USENIX Association, 2017. 1289–1306.

[38] Gueta GG, Abraham I, Grossman S, Malkhi D, Pinkas B, Reiter M, Seredinschi DA, Tamir O, Tomescu A. SBFT: A scalable and decentralized trust infrastructure. In: Proc. of the 49th Annual IEEE/IFIP Int'l Conf. on Dependable Systems and Networks. Portland: IEEE, 2019. 568–580. [doi: 10.1109/DSN.2019.00063]

[39] Kokoris-Kogias E, Jovanovic P, Gasser L, Gailly N, Syta E, Ford B. OmniLedger: A secure, scale-out, decentralized ledger via sharding. In: Proc. of the 2018 IEEE Symp. on Security and Privacy. San Francisco: IEEE, 2018. 583–598. [doi: 10.1109/SP.2018.000-5]

附中文参考文献:

[3] 夏清, 窦文生, 郭凯文, 梁赓, 左春, 张凤军. 区块链共识协议综述. 软件学报, 2021, 32(2): 277–299. http://www.jos.org.cn/1000-9825/6150.htm [doi: 10.13328/j.cnki.jos.006150]

[4] 2020 移动互联网蓝皮书: 区块链基础设施已成为产业发展重点. 2020. http://blockchain.people.com.cn/n1/2020/0715/c417685-31784959.html

[5] 高政风, 郑继来, 汤舒扬, 龙宇, 刘志强, 刘振, 谷大武. 基于 DAG 的分布式账本共识机制研究. 软件学报, 2020, 31(4): 1124–1142. http://www.jos.org.cn/1000-9825/5982.htm [doi: 10.13328/j.cnki.jos.005982]

[24] 王日宏, 张立锋, 徐泉清, 周航. 可应用于联盟链的拜占庭容错共识算法. 计算机应用研究, 2020, 37(11): 3382–3386. [doi: 10.19734/j.issn.1001-3695.2019.07.0268]

[26] 王鹃, 樊成阳, 程越强, 赵波, 韦韬, 严飞, 张焕国, 马婧. SGX 技术的分析和研究. 软件学报, 2018, 29(9): 2778–2798. http://www.jos.org.cn/1000-9825/5594.htm [doi: 10.13328/j.cnki.jos.005594]

岳镜涛(1996－), 男, 硕士, 主要研究领域为区块链共识, 拜占庭容错协议.

程凤(1999－), 女, 硕士生, 主要研究领域为区块链可扩展性, 区块链事务处理.

肖江(1988－), 女, 博士, 副教授, 博士生导师, CCF 高级会员, 主要研究领域为区块链, 分布式计算.

陈汉华(1978－), 男, 博士, 教授, 博士生导师, CCF 专业会员, 主要研究领域为大数据处理系统, 分布式计算机系统.

张世桀(1994－), 男, 博士生, CCF 学生会员, 主要研究领域为区块链可扩展性, 区块链事务处理.

金海(1965－), 男, 博士, 教授, 博士生导师, CCF 会士, 主要研究领域为分布式系统.