

区块链智能合约交易并行执行模型综述*

施建锋^{1,2}, 吴恒², 高赫然^{1,2}, 张文博^{2,3}

¹(中国科学院大学, 北京 100049)

²(中国科学院软件研究所 软件工程技术研究开发中心, 北京 100190)

³(计算机科学国家重点实验室(中国科学院软件研究所), 北京 100190)

通信作者: 张文博, E-mail: zhangwenbo@otcaix.iscas.ac.cn



摘要: 以太坊等区块链采用串行方式执行区块中的智能合约交易, 虽能严格保障执行后节点间区块链状态的一致性, 但这已成为目前制约区块链吞吐率的一个重要瓶颈. 因此, 采用并行方法优化智能合约交易的执行逐渐成为工业界和学术界关注的重点. 总结了区块链智能合约并行执行方法的研究进展, 提出了一个研究框架, 该框架以智能合约并行执行的阶段为视角, 凝练出 4 种智能合约并行执行模型, 即基于静态分析的并行执行模型、基于动态分析的并行执行模型、节点间并行执行模型和分治并行执行模型, 然后描述了每种模型下典型的并行执行方法. 最后, 对交易依赖图和并发控制策略等影响并行执行的因素进行了讨论, 并提出了未来可研究的方向.

关键词: 区块链; 智能合约; 并行执行; 事务; 吞吐率

中图法分类号: TP311

中文引用格式: 施建锋, 吴恒, 高赫然, 张文博. 区块链智能合约交易并行执行模型综述. 软件学报, 2022, 33(11): 4084-4106. <http://www.jos.org.cn/1000-9825/6528.htm>

英文引用格式: Shi JF, Wu H, Gao HR, Zhang WB. Overview on Parallel Execution Models of Smart Contract Transactions in Blockchains. Ruan Jian Xue Bao/Journal of Software, 2022, 33(11): 4084-4106 (in Chinese). <http://www.jos.org.cn/1000-9825/6528.htm>

Overview on Parallel Execution Models of Smart Contract Transactions in Blockchains

SHI Jian-Feng^{1,2}, WU Heng², GAO He-Ran^{1,2}, ZHANG Wen-Bo^{2,3}

¹(University of Chinese Academy of Sciences, Beijing 100049, China)

²(Technology Center of Software Engineering, Institute of Software, Chinese Academy of Sciences, Beijing 100190, China)

³(State Key Laboratory of Computer Science (Institute of Software, Chinese Academy of Sciences), Beijing 100190, China)

Abstract: Blockchains such as Ethereum serially execute smart contract transactions in a block, which can strictly guarantee the consistency of the blockchain state between nodes after execution, but it has become a serious bottleneck restricting the throughput of these blockchains. Therefore, the use of parallel methods to optimize the execution of smart contract transactions has gradually become the focus of industry and academia. This study summarizes the research progresses of the parallel execution methods of smart contracts in blockchains, and proposes a research framework. From the perspective of the phases of parallel execution of smart contracts, the framework condenses four parallel execution models of smart contracts, namely the parallel execution model based on static analysis, the parallel execution model based on dynamic analysis, the parallel execution model between nodes and the divide-and-conquer parallel execution model, and describes the typical parallel execution methods under each model. Finally, this study discusses the factors affecting parallel execution such as the transaction dependency graph and concurrency control strategies, and proposes future research directions.

Key words: blockchain; smart contract; parallel execution; transaction; throughput

区块链是一种去中心化的分布式数据库账本技术. 凭借其去中心化、透明性、可溯源和防篡改等特性^[1], 其应用已不再局限于比特币 (Bitcoin)^[2]、以太坊 (Ethereum)^[3]等传统加密数字货币领域, 在金融^[4]、供应链^[5]、农

* 基金项目: 国家重点研发计划 (2018YFB1402803); 国家自然科学基金 (61872344); 山东省重点研发计划 (2021CXGC010101)

收稿时间: 2021-05-08; 修改时间: 2021-06-22, 2021-08-31; 采用时间: 2021-11-05; jos 在线出版时间: 2021-12-24

业^[6,7]、数字版权^[8,9]、物联网^[10,11]、云存储^[12]等领域也有着广泛的应用前景. 然而, 目前区块链存在吞吐率(即每秒执行完成的交易数, TPS, transactions per second)低的问题^[13], 严重制约了区块链的进一步普及^[14]. 比如, 作为非许可链代表的以太坊的 TPS 约为 15–20^[15,16], 作为许可链代表的 Hyperledger Fabric 的 TPS 约为 3 000^[17], 而支付宝自主研发的分布式数据库 OceanBase 的 TPS 峰值则可以达到 6 100 万^[18].

区块链的吞吐率主要受制于两方面, 一是共识机制的执行, 二是智能合约的执行. 共识机制的改进^[19–22]在一定程度上提高了区块链系统的吞吐率, 比如张振峰等人提出的小飞象拜占庭容错 (DumboBFT) 算法^[23], 其 TPS 将近 1.8 万. 然而, 智能合约的执行正在成为另一个关键的瓶颈^[24], 尤其是串行执行智能合约交易的机制极大限制了区块链系统的吞吐率^[25].

近年来, 充分利用现代处理器的多核处理能力, 采用并行方法优化智能合约的执行已逐渐引起产业界和学术界的重点关注, 成为当前区块链性能研究中的一个重要方向^[26–28].

相对于已有文献关注区块链的发展现状^[29]、可拓展性方案^[15]、数据管理技术^[30]和智能合约技术^[31]等, 本文以智能合约并行的阶段(节点内并行, 节点间并行和子链间并行)为视角(如图 1 所示), 概括出了 4 种智能合约的并行执行模型, 即基于静态分析的并行执行模型、基于动态分析的并行执行模型、节点间并行执行模型和分治并行执行模型. 然后, 针对不同的并行执行模型, 分别梳理了典型的智能合约并行执行方法.

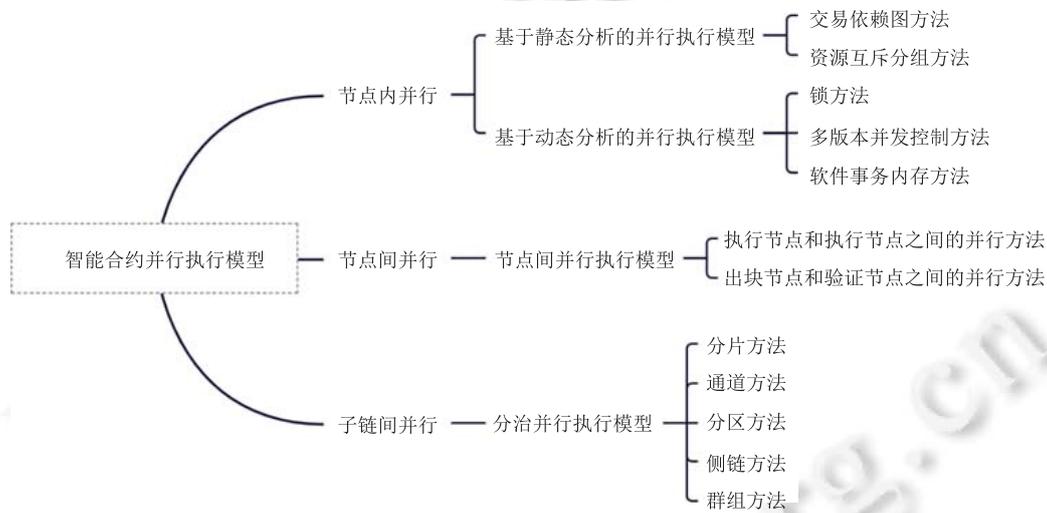


图 1 智能合约并行执行模型分类

本文第 1 节介绍了智能合约的执行特点和实现智能合约并行执行的关键任务. 第 2 节阐述了本文提出的关于智能合约并行执行方法的研究框架. 第 3 节详细描述了本文概括的 4 种智能合约并行执行模型. 第 4 节对现有的智能合约并行执行方法的研究工作进行了总结和对比, 并对一些影响智能合约并行执行的因素进行了讨论. 第 5 节建议了未来值得研究的方向. 第 6 节总结了本文的工作.

1 智能合约和并行执行

1.1 智能合约和并行冲突

1.1.1 区块链

区块链 (blockchain) 可以看成是一种新型的分布式事务处理系统^[32], 其与传统的分布式事务处理系统主要有 2 点不同.

(1) 区块链节点是允许互相不信任的^[33], 故每个节点需独立维护着一份与其他节点一致的账本, 即一串前后相连的区块 (block).

(2) 对于新区块中的智能合约交易, 区块链中的每个节点都将在各自的环境中执行一遍。

1.1.2 智能合约

智能合约 (smart contract)^[34]是由用户定义并部署在区块链上的代码^[35], 类似于编程语言中的类 (class)^[36], 包含了一批共享数据对象^[37]和函数, 可以通过函数操作数据对象, 以向用户提供一些复杂功能. 这些数据代表了当前智能合约的状态, 而所有智能合约的状态则构成了当前节点的区块链状态。

1.1.3 交易

交易 (transaction) 又称为事务, 是用户从区块链外部调用智能合约的指令^[3], 每个交易都可以指定其要调用的智能合约函数和本次调用输入的参数值。

1.1.4 并行冲突

如果两个并行执行的智能合约交易访问了同一个共享数据对象, 并且至少其中一个执行了写操作, 那么这两个合约交易就会产生冲突^[27,37,38]. 在这种情况下, 必须等待前一个合约交易执行完毕, 后一个合约交易才能执行。

1.1.5 依赖关系

如果两个智能合约交易是冲突的, 那么它们之间存在依赖关系^[38]. 对于一个智能合约交易 Tx , $\omega(Tx)$ 代表一组写操作, $\rho(Tx)$ 代表一组读操作, $\text{timestamp}(Tx)$ 代表创建该交易的时间戳. 如果两个交易 Tx_i 和 Tx_j 存在依赖关系 $Tx_i \rightarrow Tx_j$, 当且仅当 $\text{timestamp}(Tx_i) < \text{timestamp}(Tx_j)$ 和下面的至少一个条件同时成立^[39]:

$$\rho(Tx_i) \cap \omega(Tx_j) \neq \emptyset \tag{1}$$

$$\omega(Tx_i) \cap \rho(Tx_j) \neq \emptyset \tag{2}$$

$$\omega(Tx_i) \cap \omega(Tx_j) \neq \emptyset \tag{3}$$

因此, 判断同一时刻两个合约交易能否被并行执行, 就是判断这两个智能合约交易的读写集是否存在交集, 交集为空的合约交易可以被并行执行。

1.2 智能合约的执行特点

本文将以太坊等区块链的智能合约执行模型抽象为图 2, 其可以分为 2 个阶段。

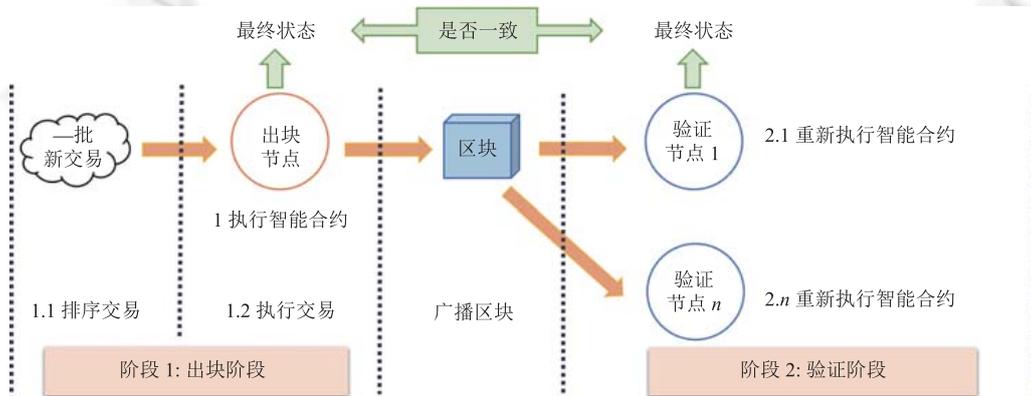


图 2 智能合约的串行执行模型

阶段 1 是出块阶段, 出块节点选取一批新的智能合约交易, 然后串行执行这批交易调用的智能合约以得到区块链的最终状态 (final state), 最终生成一个包含这批交易和区块链最终状态的新区块。

阶段 2 是验证阶段, 验证节点接收到新区块时, 重新串行执行这批智能合约交易, 并将自己节点生成的最终状态和出块节点生成的最终状态进行比对, 如果一致, 则接受该区块, 如果不一致, 则丢弃该区块^[27,40]。

以太坊等区块链执行智能合约的特点可归结如下。

- (1) 一个区块包含多个智能合约交易, 类似于批处理;
- (2) 一个区块中的交易是按照顺序串行执行的, 类似于状态机^[41];

(3) 执行过程中包含对共享账本的读写操作^[42];

(4) 每个节点都需要完全执行一遍区块中的智能合约交易. 出块节点必须执行这批智能合约交易, 才能生成一个完整的新区块. 验证节点必须重新执行新区块中的智能合约交易, 才能验证该区块的有效性^[43];

(5) 区块中的智能合约交易被执行完毕时, 每个节点的最终状态应该一致.

1.3 并行执行可能引发的问题

1.3.1 并行冲突导致的执行结果错误

在一个区块中, 不同的交易可能调用了同一个智能合约. 如果直接并行执行这些交易, 则可能对智能合约的共享数据对象进行存在冲突的读写, 从而出现丢失更新、脏读等问题, 导致最终的执行结果错误.

如图 3 所示, 智能合约交易 Tx_1 和 Tx_2 被并行执行, 其中, Tx_1 表示将用户 x 的账户增加 10 元, Tx_2 表示将用户 x 的账户增加 20 元. 在 Time 1 时刻, Tx_1 和 Tx_2 读取到的 x 的值均为 100. 在 Time 2 时刻, Tx_2 执行了写入操作, x 的值被更新为 120. 在 Time 3 时刻, Tx_1 执行了写入操作, x 的值被更新为 110. 显然, 并行执行 Tx_1 和 Tx_2 生成的结果是错误的.

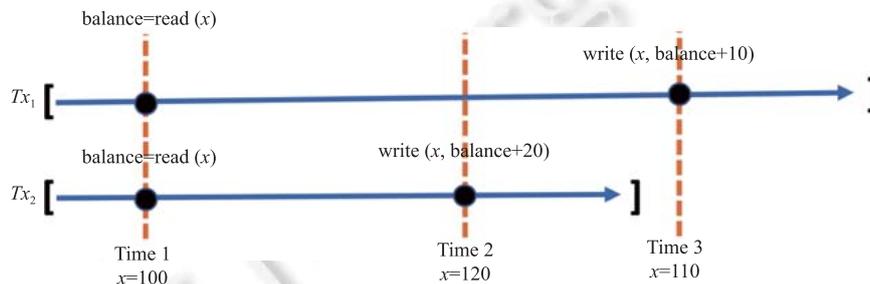


图 3 丢失更新

1.3.2 不同的串行化执行顺序导致最终状态的不一致

验证一个区块的有效性需要确定性地执行智能合约交易^[44], 验证节点才能产生和出块节点一样的最终状态. 以不同的串行化顺序执行存在冲突的交易, 这将可能导致不同节点得到不一致的区块链最终状态.

如图 4 所示, 出块节点对并行执行的串行化执行顺序是先执行 Tx_1 , 再执行 Tx_2 , 共享数据对象 a 最终的值是 20. 而验证节点对并行执行的串行化执行顺序是先执行 Tx_2 , 再执行 Tx_1 , 共享数据对象 a 最终的值是 10. 显然, 验证节点与出块节点分别得到了不一致的区块链最终状态, 因此验证节点会丢弃该区块.

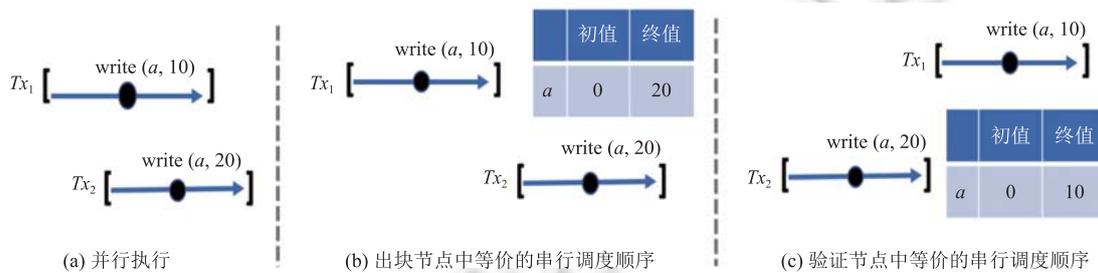


图 4 不同的等价串行调度^[36]

1.4 并行执行的关键任务

为解决第 1.3 节中并行执行智能合约交易可能引发的问题, 实现智能合约的并行执行存在如下 2 项关键任务.

1.4.1 识别具有依赖关系的冲突交易

智能合约交易能否被正确地并行执行的关键在于识别出交易集中具有依赖关系的冲突交易. 区块链中执行智能合约类似于多线程操作共享内存中的对象^[35], 并行执行的多个智能合约交易可能操作了同一个共享数据对象,

从而产生数据竞争,出现丢失更新和脏读等问题.同时,因为用来编写智能合约的 Solidity^[45]等语言是图灵完备的,直接识别出冲突的合约交易是不太可能的^[36,43,46,47].

1.4.2 生成等价的执行顺序

区块链中每个节点的最终状态必须保持一致,因此要求所有节点都能以等价的执行顺序正确地并行执行同一批智能合约交易.生成等价的执行顺序的关键在于对冲突的交易拥有相同的串行化执行顺序,这确保了各个节点并行执行智能合约交易时能够将其中冲突的交易按某种相同的顺序串行执行^[43,48].如果验证节点只是简单地重新并行执行新区块中的智能合约交易,则可能会产生不同的串行化执行顺序和不同的区块链最终状态,从而导致新区块验证失败.因此,所有节点都应该确保其并行执行的串行化顺序是等价的.

2 研究框架

通过对近年来相关研究工作的分析和总结,本文以智能合约并行的阶段(节点内并行、节点间并行和子链间并行)为视角,提出了如图 5 所示的智能合约并行执行的研究框架,其中包含 4 种智能合约并行执行模型(基于静态分析的并行执行模型、基于动态分析的并行执行模型、节点间并行执行模型和分治并行执行模型).

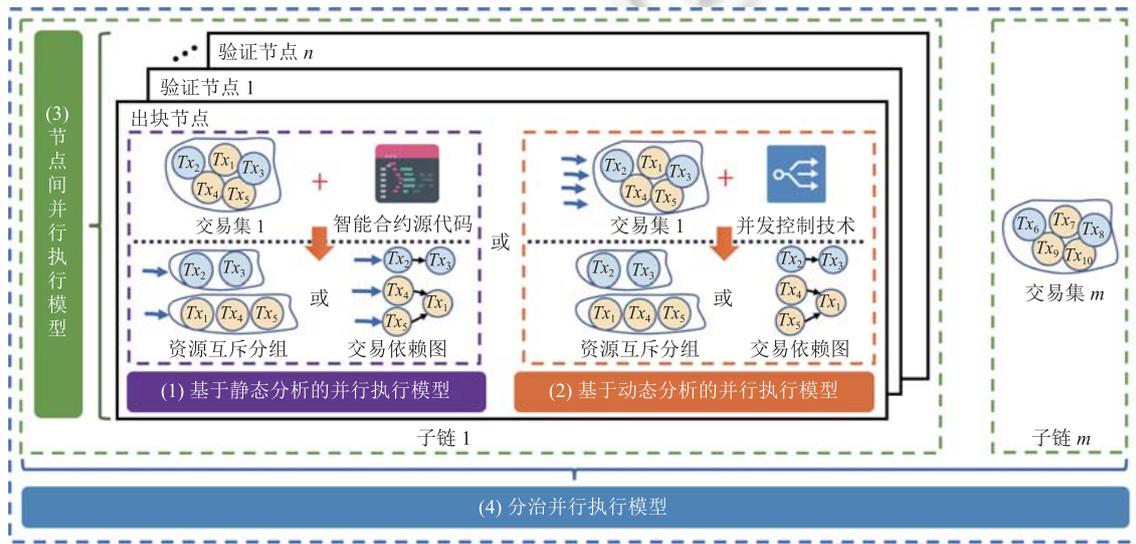


图 5 研究框架

2.1 基于静态分析的并行执行模型

区块链节点在执行智能合约交易之前,将提前分析新区块中智能合约交易之间的依赖关系,以便不同的线程能够同时执行不存在依赖关系的智能合约交易以实现并行,本文将这种并行方式概括为基于静态分析的并行执行模型.显然,这种并行模型是具有确定性的和不需要锁的,虽然会耗费一定的时间,但保证了不会发生丢失更新和脏读等并发冲突.

如图 5 所示,基于静态分析的并行执行模型^[49]的核心思路是在智能合约执行之前,找到智能合约交易之间的依赖关系,通过分析智能合约源代码里定义的共享数据对象提取该交易将占用的资源集合,然后依据这些资源的占用情况,以交易依赖图或资源互斥分组的方法记录交易间的依赖关系,最后允许占用不同资源的交易被并行执行.无论是出块节点还是验证节点,都可以独自分析智能合约交易间的依赖关系,因此无需在区块中记录交易依赖图等额外的信息.本文第 3.1 节将详细阐述该并行执行模型.

2.2 基于动态分析的并行执行模型

区块链节点无需提前知道新区块中智能合约交易间的依赖关系,而是将所有智能合约交易直接并行执行,根

据执行过程中发生的冲突, 动态地记录下智能合约交易之间的依赖关系, 本文将这种并行方式概括为基于动态分析的并行执行模型。显然, 这种并行方式是具有不确定性的, 因为冲突的交易在不同节点的执行顺序具有随机性。虽然不存在提前分析交易间依赖关系的额外耗时, 但需要利用并发控制技术解决脏读、丢失更新等并行冲突。

如图 5 所示, 基于动态分析的并行执行模型的思路是出块节点利用数据库领域中成熟的解决方案试探地并行执行所有的智能合约交易, 然后在并行执行的过程中通过回滚和延后执行冲突的交易以解决冲突。同时, 可以选择使用交易依赖图等方法记录各交易的依赖关系, 并将其保存到区块中。验证节点接收到该区块时, 可以根据交易依赖图确定性地并行执行区块中的智能合约交易。本文第 3.2 节将详细阐述该并行执行模型。

2.3 节点间并行执行模型

大多数的区块链采用如图 2 所示的交易处理架构“排序-执行-验证-提交”, 这种架构是串行的。一方面, 出块节点挑选一批交易进行排序和执行, 可以完成执行的交易数量受限于当前出块节点的最大执行能力; 另一方面, 验证阶段位于出块阶段之后, 需要重新执行一遍新区块中的智能合约交易。

在出块阶段, 将执行交易从出块节点中分离, 并将“执行交易”置于“排序交易”之前, 即构成一种新的交易处理架构“执行-排序-验证-提交”, 这种架构允许不同的节点之间并行执行来自不同客户端的交易, 从而提高区块链系统的吞吐率。

在许可链中, 所有节点都是互相信任的, 可以为出块节点和验证节点提前确定一批合约交易及其执行顺序, 然后让出块节点和验证节点同时并行执行这一批合约交易, 这将能极大缩短验证新区块的时间, 从而提高区块链系统的吞吐率。

本文将这类通过让不同节点并行以改进“执行”和“验证”过程的方法概括为节点间并行执行模型, 并将在第 3.3 节详细阐述该并行执行模型。

2.4 分治并行执行模型

一个区块链系统可以创建多条并行的子链, 每条子链可以独立执行和验证自己内部的智能合约交易。如图 5 所示, 可以将交易集按照某种规则划分到不同的子链上同时执行以实现并行, 本文将这种并行方式概括为分治并行执行模型。

分治并行执行模型的核心思想是分而治之, 通过将工作负载分配到不同的子链并行执行, 以提高整个区块链系统的吞吐率。工作负载在不同的区块链中有着不同的划分方法, 比如根据用户的账户地址进行划分或者根据子链承载的不同的分布式应用的功能进行划分等。子链在不同的区块链中可能有着不同的表述, 本文将其用来代指分片 (shards)^[15]、通道 (channels)^[50]、分区 (zones)^[51]、侧链 (sidechains)^[32]和群组 (groups)^[41]等。本文第 3.4 节将详细阐述该并行执行模型。

3 智能合约的并行执行模型

3.1 基于静态分析的并行执行模型

基于静态分析的并行执行模型主要分为 2 个模块, 资源占用隔离模块和并行执行模块。如图 6 所示, 资源占用隔离模块采用静态分析方法, 根据智能合约源代码 (或其他辅助信息) 中对共享数据对象的定义, 提取每个交易将占用的资源集合。然后通过交易依赖图方法或者资源互斥分组方法记录交易间的依赖关系。并行执行模块可以根据机器的 CPU 核数初始化对应数量的线程, 不同的线程并行执行交易依赖图中不存在依赖关系的交易 (或不同的分组), 同时无需担心在执行过程中会发生任何冲突。

3.1.1 交易依赖图方法

交易依赖图通常使用有向无环图 (DAG) 表示, 其记录了一个区块内所有交易的依赖关系, 使用顶点代表交易, 使用有向边代表 2 个交易间的依赖关系, 即执行顺序。这种图在不同文献中可能称呼不同, 比如 happen-before 图^[43]和区块图^[37]等, 本文统称为交易依赖图。

企业级联盟链底层平台 FISCO BCOS^[41]的 2.0 版本采用了交易依赖图方法以支持无锁地并行。开发者在编写

智能合约时,可以自定义需要并行的智能合约函数和需要互斥的参数,在合约交易中,这些互斥参数的值会被当做共享数据对象.如函数 1 所示,通过 registerParallelFunction 函数将转账函数 transfer 设置为允许并行,同时将该函数的前 2 位参数 (from 和 to) 设置为互斥参数.

函数 1. 注册可以并行的合约函数.

```
function enableParallel() public
{
    // 将转账函数 transfer(from, to, money) 的前 2 位参数定义为互斥的
    registerParallelFunction("transfer(string, string, uint256)", 2);
}
```

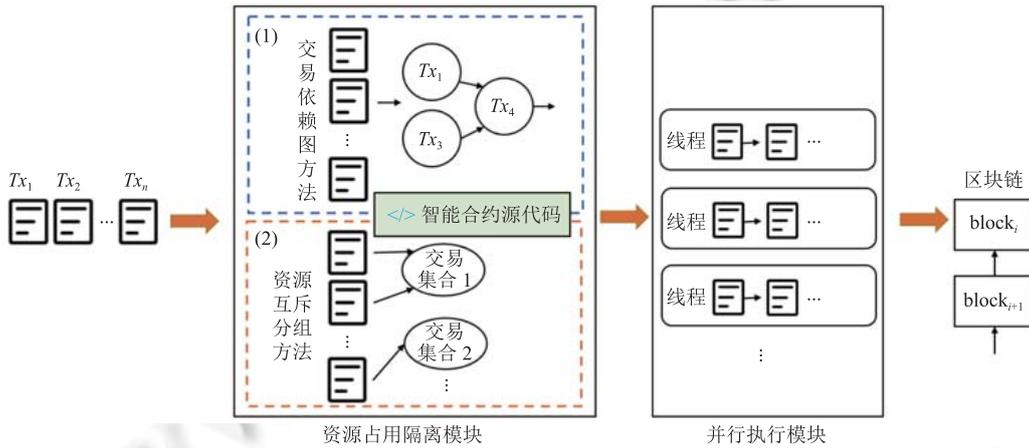


图 6 基于静态分析的并行执行模型

当该智能合约被部署后,区块链节点在执行智能合约交易前,根据交易中互斥参数的值自动构建如图 7 所示的交易依赖图,其中顶点代表合约交易,有向边代表交易间的依赖关系.

基于交易依赖图,区块链节点可以并行执行那些无依赖关系的合约交易.(1) 区块链节点首先根据机器的 CPU 核心数初始化一个相应大小的线程池.(2) 入度为 0 的合约交易允许被不同的线程直接并行执行,如图 7 中的 Tx₁、Tx₂ 和 Tx₃,这个过程是安全的,因为它们没有任何依赖的前驱交易.(3) 执行完毕后,如果被执行的合约交易所在顶点的出度不为 0,则消除以该顶点为起点的有向边,以该有向边为终点的顶点的入度同时减一.(4) 重复第 (2) 步和第 (3) 步直到区块中全部合约交易都被执行完毕.

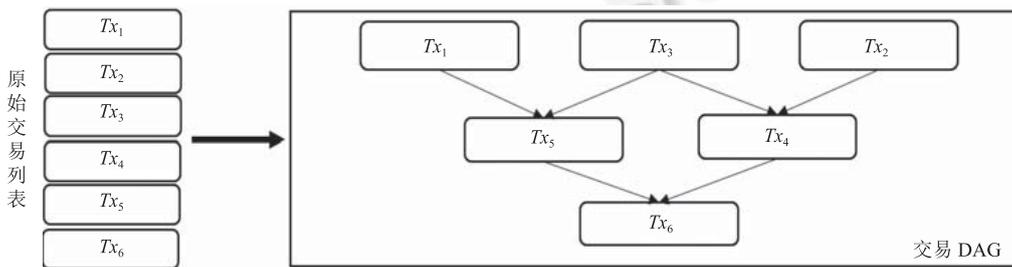


图 7 构建 DAG^[41]

在 FISCO BCOS 的性能测试中,使用 Solidity 转账合约在 4 核 CPU 的机器进行测试,串行执行实现的 TPS 为 4735,并行执行则将 TPS 提升到 12755^[41].

3.1.2 资源互斥分组方法

资源互斥分组方法根据合约交易访问的共享资源集合,发现共享资源的占用关系,从而将合约交易按照共享资源的占用关系划分到不同的组中.访问相同共享资源的合约交易将被划分到同一个分组中,即每个分组的内部都是存在冲突的合约交易.同时,每个分组之间的交易都是不冲突的,因为它们之间不存在任何共享资源的交集.例如 Yu 等人^[42]和 aelf^[32,49,52]都采用了分组方法.

智能合约中的共享变量(也称状态变量)被认为是一种共享资源.如图8所示, Tx₁和Tx₂存在共享变量x1, Tx₂和Tx₃存在共享变量x3,所以Tx₁、Tx₂和Tx₃被分为一个组^[42].Tx₄不与任何交易存在共享变量,所以独自作为一组.Tx₅和Tx₆存在共享变量x4,所以Tx₅和Tx₆被分为一组.

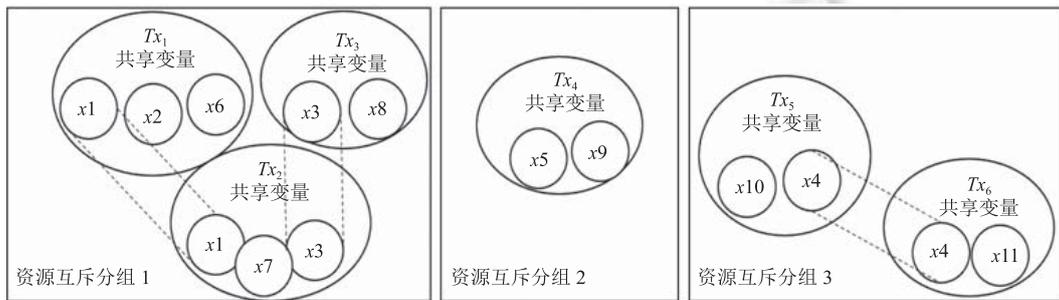


图8 资源互斥分组方法^[42]

可以通过生成一个资源依赖图(无向图)求得所有分组.如图9所示,每个顶点代表一个共享资源,若一个交易占用了2个不同的共享资源,这两个资源的顶点之间将会生成一条无向边.互相冲突的交易将会组成一个连通分量,每一个连通分量代表了一个独立的分组,因此可以使用并查集,求得所有互斥的分组^[49,52].

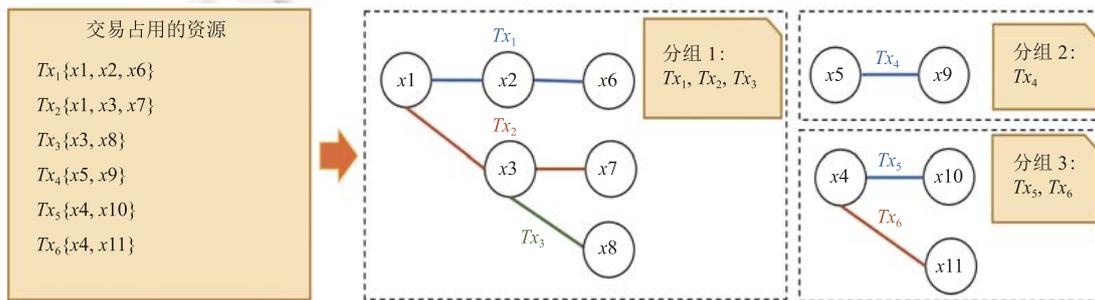


图9 资源依赖图^[49]

并行处理模块中,每个线程负责执行一个独立的分组,不同分组的合约交易不会访问同一个资源,因此这些组可以被并行执行.在 Yu 等人^[42]的实验中,每个区块包含 2000 个交易,交易的相关度是 10%,串行执行的耗时为 5809 ms,并行执行的耗时为 3625 ms.

3.2 基于动态分析的并行执行模型

智能合约交易类似于传统数据库系统中的事务^[53],同时数据库领域中的并发控制技术已经有了非常广泛的研究.因此,如图10所示,区块链节点可以采用数据库中的并发控制技术试探地并行执行所有合约交易,以此来保证并行执行的正确性.

出块节点可以采用锁机制(locking)^[54]、多版本并发控制(multi-version concurrency control, MVCC)^[55]和软件事务内存(software transactional memory, STM)^[56]等技术试探地并行执行所有智能合约交易,以解决读-写冲突导致的脏读和不可重复读问题,解决写-写冲突导致的丢失更新问题.同时,因冲突而终止的合约交易需要进行回滚

和重新执行. 最终根据执行过程中发现的冲突关系生成一个可序列化的交易依赖图, 并和区块链的最终状态一起保存到区块中.

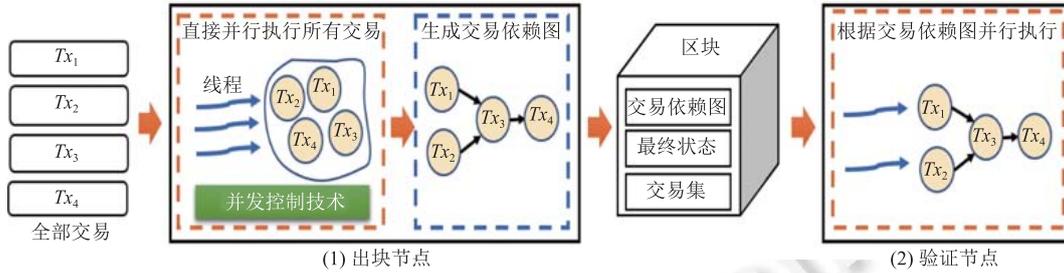


图 10 基于动态分析的并行执行模型

验证节点可以根据新区块中的交易依赖图正确地并行执行合约交易. 如果在交易依赖图中合约交易 Tx_i 到 Tx_j 有一条路径, 则必须先执行 Tx_i 再执行 Tx_j . 对于那些入度为 0 的合约交易, 验证节点可以并行地执行它们. 在执行智能合约交易的过程中, 交易依赖图会逐渐缩小, 依赖关系也会发生变化. 这种方法不仅提高了验证节点的执行效率, 也确保了验证节点和出块节点生成的区块链最终状态是相同的, 从而完成对新区块的有效性验证.

3.2.1 锁方法

锁 (locking) 是一种悲观的并发控制方法, 智能合约交易执行时对某个共享数据对象进行操作 (读操作或写操作), 都会对其请求加锁, 只要加锁成功, 就拥有了对该共享数据对象的控制权. 若是排它锁, 则在其释放前, 其他智能合约交易不能再对该共享数据对象进行读取和修改; 若是共享锁, 则在其释放前, 其他智能合约交易只能对该共享数据对象进行读取, 但不能修改.

Saraph 等人^[46]提出了一种基于读写锁 (read-write lock) 的两阶段试探执行方法. 在其方法中, 区块链节点并行执行智能合约可以分为两个阶段.

在第 1 阶段, 节点并行执行所有合约交易. 追踪每个合约交易的读集和写集 (读取和写入的内存位置), 通过对每一个存储位置关联一个读写锁检测冲突. 每一个读操作会请求一个对应位置的读锁, 每一个写操作会请求一个对应位置的写锁. 一个合约交易如果请求一个已经被其他合约交易持有、会产生冲突的锁, 那么这个合约交易就会被回滚, 并推迟到下一个阶段执行. 也就是说, 如果两个合约交易访问了同一个内存位置, 并且至少其中一个是写操作, 那么这两个合约交易就会被认为是冲突的, 其中一个交易将被提交, 另外一个交易将被放弃. 在当前阶段结束之前, 不会释放任何锁, 即使是被终止的合约交易持有的锁也不会释放.

在第 2 阶段, 节点按照顺序串行执行那些因为冲突而被回滚的合约交易.

在其实验中, 重新执行以太坊 2016 年的合约交易可以实现 8 倍的加速. 当冲突程度较低时, 试探执行的方法效果很好, 而冲突程度较高时, 因为存在大量的回滚和重新执行的操作, 整体效果不佳. 该方法没有使用交易依赖图, 出块节点和验证节点都使用相同的方法并行执行, 虽然不会占用额外存储空间, 但也未能将并行性发挥到最大^[37].

3.2.2 多版本并发控制方法

多版本并发控制 (MVCC) 是一种可以解决读写冲突的无锁并发控制策略, 即使存在读写冲突, 也可以实现不加锁的非阻塞并发读. 每个智能合约交易在执行写操作时不会直接覆盖数据项, 而是会保留数据项的每个版本. 每个读操作都可以读取到其想要读取版本的数据, 因此可以避免因为读太迟而导致的交易冲突. 总之, 读操作不用阻塞写操作, 写操作不用阻塞读操作, 多版本并发控制有效减少了交易冲突.

Zhang 等人^[44]提出了基于多版本交易排序 (multi-version transaction ordering, MVTO) 的并行方案. 出块节点可以使用任何并发控制技术生成一个序列化了的冲突交易的调度. 在执行过程中, 出块节点记录每个合约交易的写集, 并将这个写集保存到区块中. 然后出块节点重新调整这些交易在区块中的顺序以匹配刚才生成的调度. 验证节点基于新区块中的交易顺序和写集和构建一个关于冲突数据对象的“写链”, 其预先确定了这些交易的竞争关系和

优先级, 然后验证节点通过使用“写链”在运行时解决冲突以产生确定性的结果, 从而完成对该区块的有效性验证. 其在“写链”上使用了多版本并发控制, 使得执行过程中进一步减少冲突.

其实验表明, 该方法使用具有 3 个线程的线程池, 在验证区块时, 可以实现大约 2.5 倍的加速.

3.2.3 软件事务内存方法

软件事务内存 (STM) 作为一种新型的并行编程模式, 相较锁机制存在代码复杂度高和易死锁等问题, STM 允许开发者将一组需要访问共享内存的操作封装成一个事务, 然后以原子操作的方式运行^[57]. 该事务满足原子性和独立性, 原子性要求这组操作要么都执行成功, 要么都不执行. 独立性意味着这组操作所做的更新仅在成功提交时才对共享内存可见, 因此, 多个事务可以并行执行, 具有乐观性质.

通过表 1 所示的函数^[27,37], STM 提供了一个简单的并行控制机制来管理多个事务对共享内存的访问, 而无需担心一致性问题.

表 1 STM 的函数

函数	功能说明
STM.begin()	开启一个事务, 为其分配一个唯一的时间戳
STM.write(x, v)	在本事务的本地内存中, 更新数据对象x的值为v. 未提交前, 对其他事务不可见
STM.read(x, v)	尝试读取数据对象x的值, 以变量v返回. 若其他事务修改了该值, 会被通知失效
STM.tryC()	尝试提交本事务, 若成功, 该事务所做的所有修改都将更新到共享内存
STM.tryA()	终止本事务, 该事务对共享内存数据的修改都会被取消

如代码段 1 展示的部分代码^[37], STM 会为每个智能合约交易创建一个 STM 事务, 通过分解智能合约的每一步操作指令, 使用 STM.read(x, v) 替代智能合约中的读操作, 使用 STM.write(x, v) 替代智能合约中的写操作, 以此来控制并行的多个合约交易对共享内存中变量的访问. 如果一个合约交易终止了, 则 STM 会重新为这个智能合约交易创建一个新事务, 直到成功提交该事务^[37].

代码 1. STM 执行合约的部分代码.

```

 $T_i \leftarrow \text{STM.begin}();$  // 创建一个新的事务  $T_i$ 
while(curAU.steps.hasNext()) do // 执行当前智能合约代码的每一步
  curStep = curAU.steps.next();
  switch(curStep) do // 判断当前操作的类型
    case read(x): // 使用 STM.readi(x) 替代读操作 read(x)
      v ← STM.readi(x);
      if(v == abort) then
        goto Line 1;
      end if
    case write(x, v): // 使用 STM.writei(x, v) 替代写操作 write(x, v)
      STM.writei(x, v);
    case default: // 其他不在内存中进行读写的操作
      execute curStep;
  end while
v ← STM.tryCi(); // 尝试提交该事务
if (v == abort) then
  goto Line 1; // 如果提交失败, 意味着存在冲突, 重新开始一个新事务
end if

```

基于乐观软件事务内存系统 (optimistic STM), Anjana 等人提出了一种高效的无锁并行执行框架^[37]. 如图 11 所示, 出块节点基于 STM 多线程并发地执行所有合约交易, 最终可以得到一个有冲突的交易集合 (使用交易依赖图表示) 和一个无冲突的交易集合 (被称为并行箱).

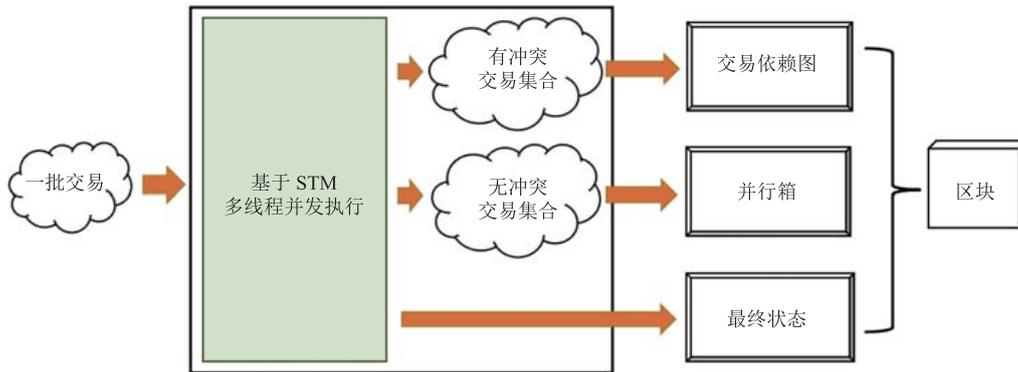


图 11 基于 Bin 和乐观 RWSTM, 出块节点并发执行合约交易

验证节点接收到新区块之后, 分两个阶段确定性地并行执行新区块中的合约交易. 第 1 阶段, 所有线程从并行箱中取智能合约交易, 然后可以直接并行执行, 因为并行箱中的交易不存在任何冲突. 第 2 阶段, 并行箱中的智能合约交易执行完之后, 开始根据交易依赖图执行那些具有冲突的交易. 最后将本节点此时产生的最终状态和区块中记录的最终状态进行对比, 如果一致, 则接受该区块, 如果不一致, 则丢弃.

在其实验中, 出块节点和验证节点都基于多版本时间排序的 STM 实现智能合约的并行执行, 相较串行执行分别得到了 4.55 倍和 7.84 倍的速度提升.

3.3 节点间并行执行模型

节点间并行执行模型是指通过让区块链节点在交易处理架构“排序-执行-验证-提交”的某个过程并行以提高区块链系统的吞吐率, 其主要包含 2 种并行方法, 一是让将“执行交易”置于“排序交易”之前, 通过让不同的区块链节点并行执行来自不同客户端的交易, 以提高区块链系统的处理能力. 二是针对全网节点在某一时刻具有确定的待执行交易集的区块链, 让出块节点和验证节点在智能合约交易的执行阶段并行, 从而提高区块链系统的吞吐率. 本文将“执行节点和执行节点之间并行”和“出块节点和验证节点之间并行”等方法均抽象为节点间并行执行模型.

3.3.1 执行节点和执行节点之间的并行方法

如图 2 所示的交易处理架构“排序-执行-验证-提交”, 合约交易的排序和执行通常均在一个节点上完成. 显然, 在这种架构下, 区块链系统的吞吐率受限于出块节点的处理能力, 系统的性能并不会因为节点数的增加而得到提升^[28]. 为了获得更大的交易处理能力, 受启发于数据库中乐观并发控制机制, Hyperledger Fabric^[50]提出了一种新的交易处理架构“执行-排序-验证-提交”. 这种新架构中, 合约交易的排序和执行是分开的, 并且将“执行交易”置于“排序交易”之前. 本文将可以执行交易的节点称为执行节点 (如 Hyperledger Fabric 中的背书节点和 XuperChain^[58]中的全节点), 负责排序出块的节点仍然称为出块节点 (如 Hyperledger Fabric 中的排序服务和 XuperChain 中的矿工节点).

XuperChain 同样采用了“执行-排序-验证-提交”架构. 如图 12 所示, 客户端首先向一个执行节点提交包含智能合约执行参数的预执行请求 (Hyperledger Fabric 中称为模拟执行); 该执行节点基于当前区块链状态进行模拟执行, 此过程并不修改该执行节点的区块链账本状态, 然后将读写集返回给客户端; 客户端组装一个包含该读写集的交易, 发送给区块链网络; 出块节点收集和排序一批交易以生成一个新区块, 并将其广播给所有执行节点; 执行节点验证新区块的有效性, 并更新本地的区块链账本状态.

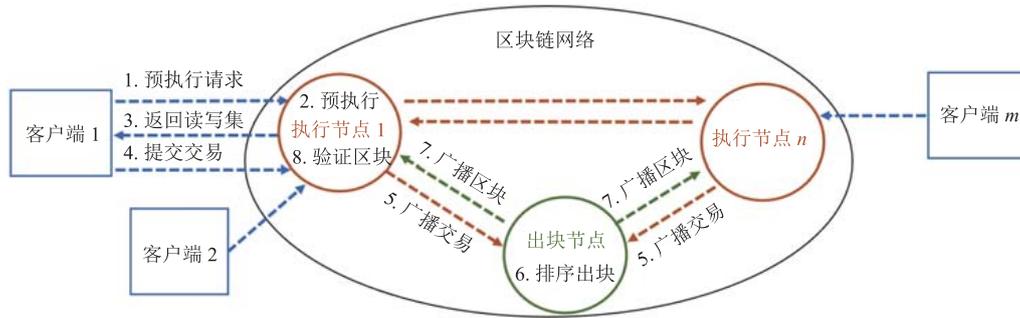


图 12 XuperChain 的交易流程

显然, 一个执行节点除了自身可以并行处理不同的预执行请求, 还能与其他执行节点并行处理更多的来自不同客户端的预执行请求, 这是一种并行能力的放大.

3.3.2 出块节点和验证节点之间的并行方法

采用 PoW 共识机制的区块链, 所有节点都在争夺出块权, 并不能确定哪个节点是此时的出块节点, 每个节点打包的交易集是不同的^[40], 具有随机性. 因此, 验证节点必须在出块节点生成新区块之后才能知道需要执行的交易集, 即验证阶段的交易执行一定位于出块阶段的交易执行之后. 但对于许可链, 节点间是互相信任的, 可以由专门的节点 (可以是出块节点) 选择一批新交易进行排序, 再广播给其他节点执行.

如图 13 所示, 因为交易集和出块节点在此刻是确定的, 那么可以让所有节点 (包括出块节点和验证节点) 同时执行交易集, 等待出块节点执行完所有智能合约交易之后, 出块节点将包含执行结果的新区块广播给其他节点, 其他节点通过比较自己的区块链最终状态和出块节点的区块链最终状态是否一致以判断出块节点生成的新区块的有效性^[59].

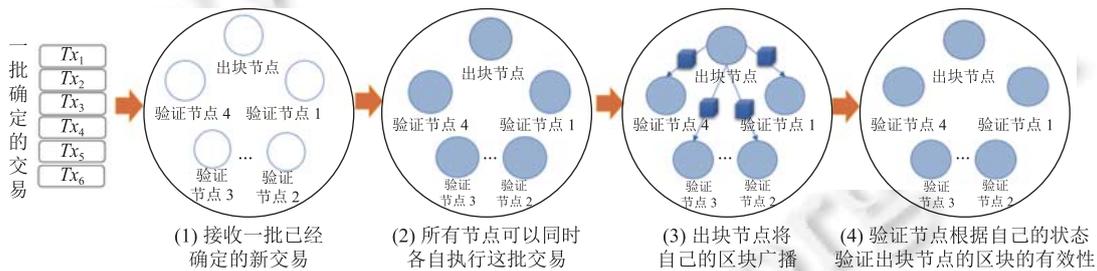


图 13 出块节点和验证节点之间的并行方法^[59]

如图 14 所示, 这种出块节点和验证节点并行的方式, 让验证节点不必等待出块节点将智能合约交易执行完毕就可以开始执行这批智能合约交易, 这极大地缩短了验证阶段的耗时.

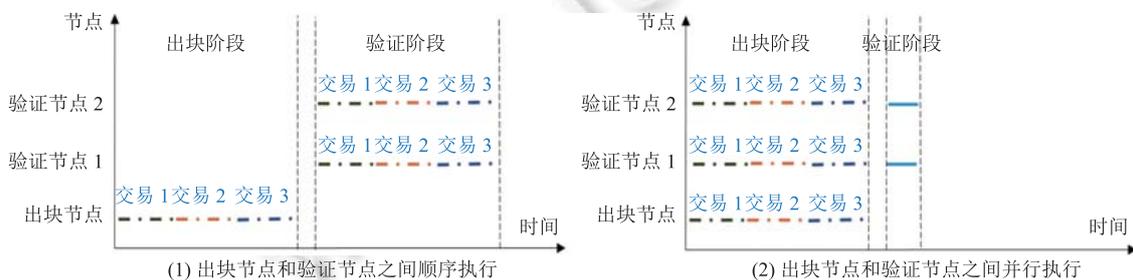


图 14 出块节点和验证节点之间的并行方法的效果图^[59]

3.4 分治并行执行模型

分治并行执行模型如图 15 所示, 负载划分模块根据一定的规则, 将全部的合约交易划分为不同的子集, 并分配到不同的子链进行执行. 因为子链之间是并行的, 从而提高了整个区块链系统的吞吐率. 本文将以以太坊中的分片^[15]、Hyperledger Fabric 中的通道^[50]、Monoxide 中的分区^[51]、aelf 中的侧链^[32]、FISCO BCOS 中的群组^[41]等并行方法均抽象为分治并行执行模型.

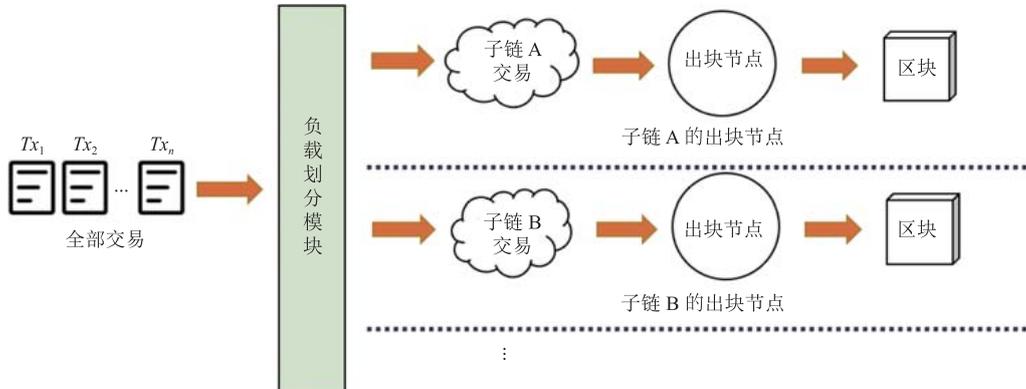


图 15 分治并行执行模型

3.4.1 分片方法

分片方法 (sharding) 是一种被给予厚望的并行方案, 其能够对网络中的计算资源实现更有效的管理^[15,60-62]. 分片方法将整个区块链根据一定规则划分成多个子集, 每个子集被称为一个分片 (shard), 每个分片具有独立的状态和账本. 区块链网络中所有新交易将按照规则分配到不同的分片中, 分片间并行执行这些交易. 由于每个分片只需要处理一部分交易, 因此增加了整个区块链系统的吞吐量. 在比特币、以太坊等传统区块链中, 增加节点的数量只会增加系统的安全性, 但不会增加系统的吞吐率. 与此不同的是, 采用分片方法的区块链因为分片数量的增加, 其交易处理的并行度也将线性增加.

以太坊开发团队计划通过分片方法提升以太坊平台的吞吐率, 其新架构主要由两部分组成:“信标链 (beacon chain)”是负责主要的共识机制, 而“分片链 (shard chains)”是存储账户数据和交易的多个子链^[15]. 分片方法在区块链领域中的热度越来越高, 许多采用分片方法的区块链研究和项目不断涌现, 比如 Zilliqa^[63]、RapidChain^[64]、Quarkchain^[65]、Chainspace^[66]、Omniledger^[67]和 Elastico^[68]等. 但是, 分片方法分散了整个网络的节点, 相比整个链更容易遭到破坏, 从而可能威胁到区块链系统的安全性^[15], 同时也需要研发相应的机制处理分片间的通信.

3.4.2 通道方法

通道方法 (channel)^[69]被开源的企业级许可链平台 Hyperledger Fabric^[50]采用, 具有极佳的保密性和可伸缩性. 比如在一个供应链金融的区块链网络中, 参与方包括核心企业、供应商、银行和物流企业等. 不同的参与方各自运行一个区块链节点. 显然, 该区块链提供的金融服务不需要物流企业的节点参与共识和保存账本数据. 同时, 该区块链提供的物流服务也不需要金融机构的节点参与共识和保存账本数据. 通道方法允许需要进行私有交易的成员与业务竞争者或其他受限制的成员在同一个区块链网络中共存.

如图 16 所示, 通道相当于某几个网络成员之间进行通信的私有“子网”^[70], 每个通道都维护着自己的账本数据. 在模拟执行阶段, 每个背书节点只接收和执行在其通道内有权限的客户端发起的交易提案. 在排序阶段, 排序服务为每个通道创建单独的区块. 在验证阶段, 节点只接收在同一个通道上的区块进行验证. 在提交更新阶段, 验证通过的区块只会追加到在同一个通道的节点的账本. 可以发现, Hyperledger Fabric 的通道隔离模型类似于其他区块链的分链模型, 通道为交易处理的各个方面都带来固有的并行性^[70].

3.4.3 分区方法

分区方法被 Wang 等人^[51]提出的 Monoxide 区块链采用. 如图 17 所示, 系统会被分为多个区域 (zones), 每个分区被称为一个异步共识区. 每个异步共识区都拥有自己的区块链账本、状态数据库等, 能够在各自的分区内完成交易的处理和区块的共识. 不同的合约交易会根据规则映射到某个分区进行处理, 不同分区的合约交易因此可以被并行执行, 从而提高了整个区块链系统的吞吐量. 分治并行执行模型都会存在因为算力稀释导致子链安全性下降的问题^[15], 这是因为每个子链只是由系统中的部分节点运行. 对于区块链来说, 越多的节点参与共识, 黑客聚集大于 51% 算力的难度就越大, 系统就越安全. 与其他分治并行执行模型不同的是, Wang 等人^[51]提出了“连弩挖矿”方法以应对算力稀释导致的安全性下降问题. 其实验表明, 在 1200 台虚拟机上运行 48000 个节点, 实现了超过以太坊 1000 倍的吞吐量. 同时, 随着分区数量的增加, 吞吐量可以实现线性增长.

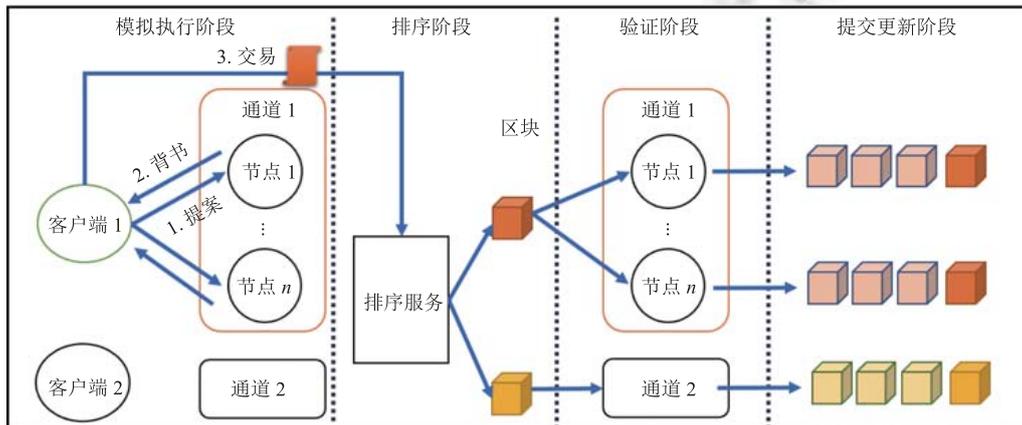


图 16 Hyperledger Fabric 的交易处理流程

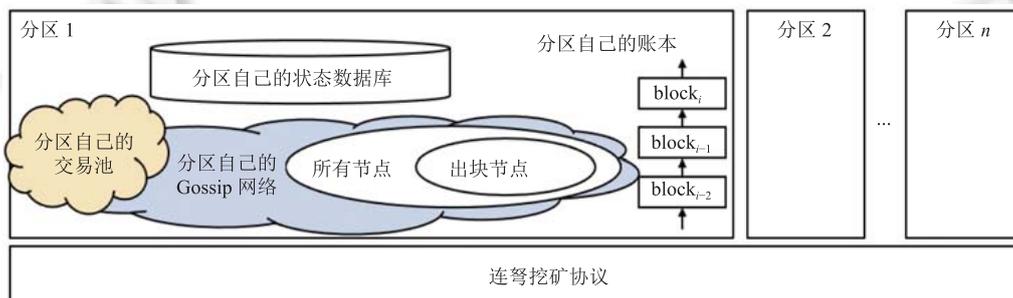


图 17 异步共识区^[51]

3.4.4 侧链方法

侧链方法同样被许多区块链采用以实现智能合约交易并行执行. 多级侧链结构如图 18. aelf 是一个基于多级侧链的并行化区块链框架^[32], 其主链负责统一规划和索引, 侧链独立运行, 每条侧链专门负责一项功能服务. 不同功能服务的合约交易可以在所属的侧链上并行执行, 因此得以提高整个区块链系统的合约交易执行效率. 同时, 侧链可以通过主链的验证以实现跨链交互. 此外, 每个子链中采用了资源互斥分组方法, 交易集会根据冲突关系被划分为多个分组, 同组内的合约交易串行执行, 不同组的合约交易并行执行. aelf 区块链的并行化方案的 TPS 达到了近 1.5 万^[49].

3.4.5 群组方法

群组方法被 FISCO BCOS^[41]用来实现其分治并行执行模型. 每个区块链节点都可以根据实际业务关系, 自由

组成多个群组. 如图 19 所示, 每个群组拥有一个属于自己的独立账本, 其交易处理、数据存储和区块共识都是与其他群组相互隔离的. 这种架构在保障隐私性的同时, 允许不同群组间的交易合约可以被并行执行, 从而提高了整个区块链系统的处理能力.

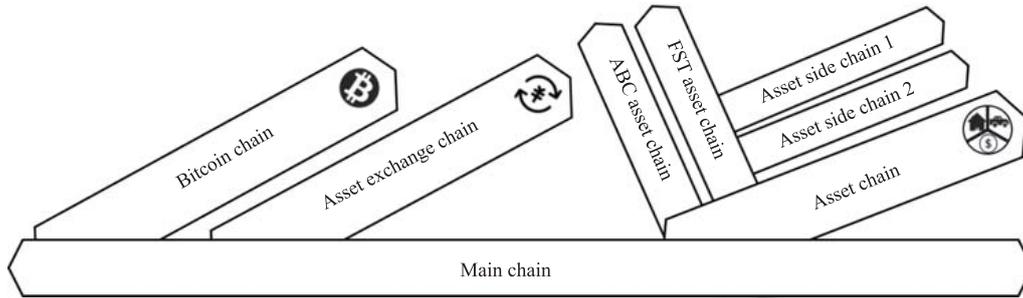


图 18 多级侧链结构 [32]

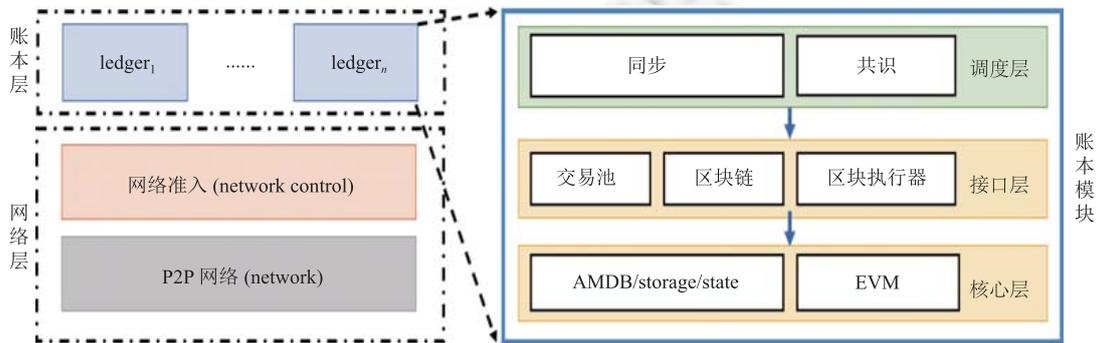


图 19 群组架构 [41]

4 现有工作总结和相关讨论

4.1 现有工作总结

表 2 是本文对现有的智能合约并行执行的相关研究工作的总结和对比. 其中的维度包括并行的阶段、所属的并行执行模型、出块节点的并行执行方法、是否需要锁、区块中是否包含交易依赖图和验证节点的并行执行方法.

4.2 区块中携带交易依赖图

区块中如果携带交易依赖图, 直接的影响是增大了区块的大小, 进而增加通信和存储开销. 如果区块的大小继续保持不变, 则会压缩区块能够存储的交易数量, 使得区块的有效载荷变小. 此外, 恶意节点也可能通过广播包含精心编制过的交易依赖图的新区块实现双花攻击 [41]. 但好处在于验证节点不仅可以较快地执行该区块中的智能合约交易, 而且可以保证验证节点并行执行冲突交易的过程中, 保持和出块节点一样的执行顺序, 从而能够快速验证该区块的有效性.

区块中如果不携带交易依赖图, 虽然减轻了存储和通信的开销, 但验证节点会因为不知道区块内哪些智能合约交易是冲突的而增加了执行时间, 同时也可能因为对于冲突交易的执行顺序和出块节点不一样, 导致错误地拒绝了本来有效的区块, 进而降低了区块链的效率.

比如, 以太坊的每个区块的平均大小为 20.98 KB, 平均每个区块包含的交易数量约为 100 [38,71]. 在 Anjana 等人 [37] 的实验中, 默认的单版本时间排序 (Def-BTO) 的交易依赖图占用一个区块的存储比例平均为 34.96%, 经过优化的单版本时间排序 (Opt-BTO) 的交易依赖图占一个区块的存储比例平均为 17.71%. 由此可见, 一方面, 交易依赖图的存储大小不容忽视; 另一方面, 对交易依赖图的存储结构进行优化具有重要的意义.

表2 现有工作总结

工作	并行阶段	并行执行模型	出块节点并行执行方法	锁	交易依赖图	验证节点并行执行方法
Yu等人 ^[42]	节点内	基于静态分析	根据交易中的共享变量的交集进行分组, 然后多线程并行执行	×	×	同出块节点
DiPETrans ^[40]	节点内	基于静态分析	每个节点由一个集群构成, leader机器使用静态分析将交易分成独立的组, 分配给follower机器并行执行	×	×	同出块节点
Dickerson等人 ^[43]	节点内	基于动态分析	基于抽象锁的消极软件事务内存	√	√	根据交易依赖图并行执行
Zhang等人 ^[44]	节点内	基于动态分析	不限	—	写集	多版本交易排序
Saraph等人 ^[46]	节点内	基于动态分析	基于锁的两阶段试探执行	√	×	同出块节点
Anjana等人 ^[27,36]	节点内	基于动态分析	乐观的基于读写的软件事务内存	×	√	根据交易依赖图去中心化地多线程并行
Anjana等人 ^[38]	节点内	基于动态分析	乐观的基于对象的软件事务内存	×	√	根据交易依赖图去中心化地多线程并行
Anjana等人 ^[37]	节点内	基于动态分析	乐观的基于读写的软件事务内存	×	√(只保存有依赖关系的交易)	根据并行箱和交易依赖图去中心化地多线程并行
Pang等人 ^[47]	节点内	基于动态分析	结合批处理特性的OCC (optimistic concurrency control)变体	×	中粒度并发调度日志	确定性OCC
ChainMaker ^[72]	节点内	基于动态分析	OCC并行执行, 交易执行结束时检测读集是否被其他交易修改	×	√	根据交易依赖图并行执行
Yu等人 ^[59]	节点间	节点间	—	—	—	与出块节点部分并行
Amiri等人 ^[39]	节点间	节点间	不同的节点并行执行只属于自己承载应用的智能合约交易	×	√	—
FISCO BCOS ^[41,73]	子链间+节点内	分治+基于静态分析	不同的群组并行执行+根据交易依赖图多线程并行执行	×	×	同出块节点
aelf ^[32,49]	子链间+节点内	分治+基于静态分析	不同的侧链并行执行+并行执行资源互斥的分组	×	×	—
Monoxide ^[51]	子链间	分治	不同的异步共识区并行执行	—	—	—
Hyperledger Fabric ^[50]	子链间+节点间+节点内	分治+节点间+节点内	不同的通道并行执行+各背书节点并行执行+背书节点自身并行执行	×	读写集	根据读写集验证
XuperChain ^[58]	子链间+节点间+节点内	分治+节点间+节点内	不同的平行链并行执行+各全节点并行执行+全节点自身并行执行	×	读写集	根据读写集并行验证

注: ×代表不需要, √代表需要

4.3 并行执行的中心化程度

区块链节点实现并行的方式按照中心化程度可以分为2类, 中心化的并行执行和去中心化的并行执行。

中心化的并行执行是指区块链节点中存在一个主线程识别出那些在交易依赖图中不存在任何依赖的合约交易, 并将它们分配给不同的子线程执行^[36], 即 Dickerson 等人采用的 fork-join 方法^[43]。

去中心化的并行执行是指区块链节点中不存在一个负责分配的线程, 每个线程都会独立识别那些在交易依赖图中入度为0的顶点, 线程会占据该顶点并执行对应的智能合约交易。Anjana 等人表明, 去中心化的方法产生了超过 fork-join 显著的性能收益^[36,37]。

4.4 不同的并发控制策略

在基于动态分析的并行执行模型中, 出块节点采用并发控制策略协调对共享数据对象的并发访问, 以确保智能合约交易作为事务的原子性和隔离性。

(1) 悲观并发控制和乐观并发控制

悲观并发控制 (pessimistic concurrency control) 在改变共享数据对象之前会将该对象锁住, 直到提交了所做的

更改之后才会释放锁. 乐观并发控制 (optimistic concurrency control) 假设数据竞争的程度较低^[35], 读取或改变共享数据对象时并不加锁, 只在提交操作时检查是否存在冲突. 悲观并发控制使用简单, 读取和修改前加锁, 遇到冲突则等待, 避免了回滚, 但存在死锁和延迟问题, 并发量大的场景性能较差. 乐观并发控制可以提高并行执行的效率, 但是如果出现了冲突只能回滚, 然后重新执行.

通常, 乐观并发控制比悲观并发控制可以实现更高的执行效率, 如 Dickerson 等人^[43]提出的基于抽象锁的消极软件事务内存的并行方法, 验证节点相较串行执行可以取得 1.59 倍的加速. 而 Anjana 等人^[37]提出的无锁乐观软件事务内存的并行方法, 验证节点可以取得 7.05 倍的加速. 同样, 长安链 (ChainMaker) 采用了乐观并发控制, 其 TPS 超过了 10 万^[72].

区块链选择使用哪一种并发控制策略取决于数据竞争的程度和解决冲突的代价. 数据竞争的程度是指一个区块中的交易所调用的相同的智能合约的数量分别占总数的比例, 如果粒度更细, 则可指智能合约中的某个函数或某个共享数据对象. 如果一个区块中的大部分交易都是调用某个智能合约, 则称数据竞争的程度较高. 解决冲突的代价是指并行执行的智能合约交易产生冲突时, 回滚和重新执行被终止的合约交易的代价. 如果某个智能合约函数实现的是耗时较长的复杂任务, 那么回滚并重新执行的代价就非常高, 则称解决冲突的代价较高.

因此, 智能合约的并行执行选择使用何种并发控制策略需要根据应用场景的特征决定. 悲观并发控制适合数据竞争的程度高和解决冲突的代价高的场景. 而乐观并发控制适合数据竞争的程度低和解决冲突的代价低的场景.

(2) 单版本并发控制和多版本并发控制

单版本并发控制 (SVCC) 里每个共享数据对象只有一个版本, 合约交易会因为读写冲突而被终止和回滚. 多版本并发控制 (MVCC) 在执行写操作时没有直接覆盖数据, 而是保留了写操作的每个版本, 每个读操作都知道它应该读取哪个版本的数据, 所以能够减少读写冲突.

通常, 多版本并发控制比单版本并发控制可以实现更少的回滚次数和更高的吞吐率, 如 Anjana 等人^[37]的实验中, 使用单版本并发控制的 STM 和多版本并发控制的 STM, 出块节点并行执行相较串行执行分别得到了 4.10 倍和 4.55 倍的提升, 验证节点并行执行相较串行执行分别得到了 7.05 倍和 7.84 倍的提升.

4.5 静态分析和动态分析智能合约交易的依赖关系

构建交易依赖图的关键是发现区块中所有合约交易的依赖关系, 可以通过分析合约交易的读写集以获得合约交易间的依赖关系. 在基于动态分析的并行执行模型中, 通过试探执行 (如锁机制和软件事务内存等) 获取每个交易的读写集, 从而动态发现交易间的依赖关系. 静态分析的方法要求在生成交易依赖图的时候, 需要对合约交易的读写集有先验知识, 即无需通过执行合约交易就可以提前知道该合约交易的读写集. 合约交易的读集和写集可以预先声明, 或者可以通过静态分析从合约交易中获取. 在文献 [39] 中, 生成交易依赖图之前, 要求将读集和写集预先声明在客户端发送的请求中. 在文献 [41] 中, 开发者在编写智能合约时, 需要提前定义好哪个函数的哪几个参数是互斥变量.

动态分析因为需要回滚和重新执行因冲突而终止的交易, 冲突率较高的场景下比较耗时, 但对于合约开发者的要求较低, 无需在编写智能合约代码的时候, 定义和管理冲突. 静态分析要求开发者在编写智能合约代码的时候提前定义好, 对合约开发者要求较高, 但是在智能合约并行执行阶段可以取得更优的性能.

4.6 性能评估

在表 2 中, 不同的研究工作实现的性能提升不具有强可比较性, 不能片面地根据各文献中的表述来评价优劣. 主要原因如下.

- (1) 使用的智能合约编程语言和虚拟机可能不一样;
- (2) 区块链系统的实现程度可能不一样, 比如成熟的区块链框架拥有许多组件以实现安全等功能, 这些组件的应用自然在一定程度上限制了智能合约执行的性能;
- (3) 区块链的类型可能不一样, 一方面, 非许可链由于可能存在恶意的节点, 每个出块节点都需要自己对交易

进行排序, 而许可链则可以由专门的节点来充当排序节点, 性能自然不一样. 另一方面, 由于共识机制的关系, 非许可链的性能几乎和节点数量没有关系, 而基于 PBFT 的许可链则会因为节点数的增加而导致通信开销的暴增, 因为每个背书节点都必须和其他的节点一一进行通信^[15];

(4) 测试基准可能不一样, 比如区块大小、测试数据的竞争程度、测试环境等都可能存在不同.

4.7 区块扩容、缩小区块产生的时间间隔和 DAG 区块链

目前拥有许多提升区块链系统 TPS 的方法, 比如区块扩容、缩小区块产生的时间间隔和采用 DAG 架构等, 但是这些方法都存在较大的局限性. 可以通过调高区块的大小限制以容纳更多的智能合约交易, 从而提升区块链系统的吞吐量, 但是更大的区块可能增加了区块链的共识时间^[15], 这对于金融等注重时效性的应用场景来说是不可接受的. 以太坊等非许可链的节点众多且地理分布广, 缩小区块产生的时间间隔可能导致新区块在区块链网络中无法得到充分的广播, 进而导致网络安全问题^[52]. DAG 区块链采用了一种全新的架构, 能够并行验证交易, 但增加了实现的复杂性^[15].

4.8 并行执行智能合约的动机

(1) 提高吞吐率. 吞吐率是区块链系统最重要的性能指标之一, 多个不存在依赖关系的合约交易如果能够被并行执行, 这将会能大幅提高区块链系统的吞吐率.

(2) 提高计算机资源利用率. 串行执行智能合约交易时, 多核 CPU 里其他的核可能是完全空闲的, 这不仅限制了系统的吞吐率, 更是对计算资源的极大浪费.

(3) 增加伸缩性. 单个节点即使增加 CPU 核心或者直接增加机器, 因为串行执行的特点, 也不能使区块链的吞吐率得到提升. 并行执行合约交易将可以通过提高机器的配置来提升交易执行的效率, 从而支持区块链的所有者不断地扩大业务规模.

(4) 多方受益. 更快地执行区块中交易调用的所有智能合约, 区块链的所有参与者都将受益. 比如采用 PoW 共识机制的区块链, 出块节点(矿工节点)若能快速完成合约交易的执行, 其将能尽早进入到求解 PoW 难题的阶段, 以便最终获得奖励. 其他节点作为验证节点, 也希望能够对接收到的新区块进行快速验证, 以便尽快投入到下一个区块打包权的争夺. 许可链通常由某个组织运营, 若能根据业务的发展情况, 动态地调整运行区块链节点的计算机资源以提升整个系统处理合约交易的能力, 这将会能有效控制经济成本.

5 未来可研究方向

5.1 利用集群提升并行执行智能合约交易的能力

如今的大部分研究都是关注如何利用单机多核处理器提高并行执行智能合约交易的能力. 根据互联网和分布式系统的发展经验, 区块链的节点将发展为运行在以多机器组成的集群之上. 如何利用集群的可扩展性提高节点并行执行智能合约交易的能力将会成为一个重要的研究方向^[40].

5.2 支持嵌套的智能合约并行执行

许多研究都是假设不存在嵌套的智能合约调用, 但是实际上, 在智能合约的编写中, 嵌套是非常普遍的. 能够在并行框架中支持嵌套的智能合约将会非常有用^[19]. 比如在 FISCO BCOS 中, 声明可并行的合约接口必须满足两个条件^[41]: (1) 无调用外部合约; (2) 无调用其他函数接口. Dickerson 等人^[43]提出了使用嵌套的试探操作 (nested speculative actions) 应对一个智能合约调用另外一个智能合约的情况, 其中, 嵌套的试探操作会继承其父级操作的抽象锁, 同时创建自己的回滚日志.

5.3 构造更优的交易依赖图

Anjana 等人^[37]的交易依赖图只保存具有依赖关系的合约交易, 有效减少了交易依赖图的大小. 在其实验中, 经过优化后的交易依赖图在存储效率方面是传统默认的 1.97 倍, 可见交易依赖图的构造和规模对存储开销具有较大的影响. 交易依赖图的顶点和边越多, 区块链需要的存储空间和通信开销就越大, 这将成为不可忽视的负担.

因此,如何构造一种经过优化的交易依赖图具有一定的研究意义^[38].

5.4 设计支持多线程的智能合约虚拟机

大多数的区块链都支持以太坊虚拟机 (EVM) 作为智能合约引擎来执行智能合约,但是 EVM 是不支持多线程的^[74],无法并行执行智能合约交易,这在一定程度上影响到了智能合约的并行性.而支持多线程的 Java 虚拟机 (JVM) 等是为通用场景设计的,不能充分适应和利用区块链的环境特点,所以设计专用于智能合约且支持多线程的虚拟机将具有较大的前景^[38].例如,2021 年 3 月,微众银行推出的全新的智能合约编程语言 Liquid 对 WASM 执行引擎进行了深度优化,其中就支持合约交易并行化^[75].

5.5 提高区块链节点的并行效率

无论是采用何种并行执行模型,虽然一定程度提高了区块链节点执行智能合约交易的效率,但在目前的研究工作中,节点的并行效率仍然存在很大的提升空间,比如出块节点如何在打包交易时让可以并行执行的交易打包到同一个区块以提高并行性^[30],在采用“执行-排序-验证-提交”的区块链中如何通过交易重排序减少交易终止率^[28,76],如何高效分片以最大化并行性^[77]以及如何减少跨分片通信^[78]等.

5.6 混合的并行执行模型

尽管分片方法通过并行执行交易在一定程度上提高了吞吐量,但每个分片本质上是一个规模较小的区块链,如果其内部的节点仍然串行执行合约交易,则整个区块链系统的吞吐量并没有得到显著提升^[79].因此,可以将第 3 节中描述的多种并行执行模型在 3 个不同的并行层级(节点内、节点间和子链间)上进行融合,以实现更高的并行性.如图 20 所示,aeif^[32]将基于静态分析的并行执行模型和分治并行执行模型进行融合,在不同的层级上实现并行,极大地提高了吞吐量.XuperChain 结合平行链、侧链和 DAG^[80]等并行方法可以达到 20 万 TPS^[58].

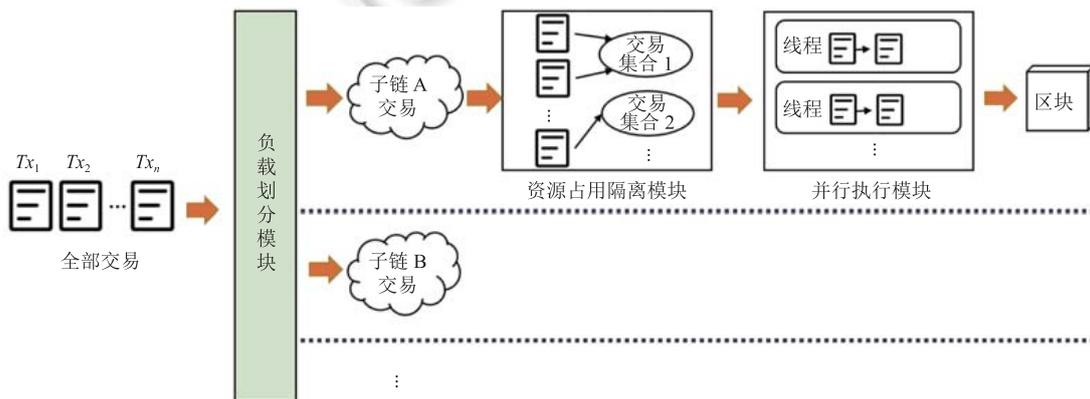


图 20 混合的并行执行模型

6 总结

大多数的研究工作是改进区块链的共识层以提高吞吐量,而本文关注的是通过改进区块链的执行层以提高吞吐量.尽管串行执行智能合约交易可以防止执行过程中任何可能的不一致,但其限制了区块链的吞吐率和可拓展性.通过充分利用现代机器提供的多核处理能力,并行执行智能合约交易将能显著提高区块链系统的吞吐量.正确地并行执行智能合约交易的关键在于识别出具有依赖关系的冲突交易和生成等价的执行顺序.本文通过对已有的研究工作进行较为系统的总结,以智能合约并行的阶段(节点内并行、节点间并行和子链间并行)为视角,提出了一个区块链智能合约并行执行模型的研究框架,详细阐述了基于静态分析的并行执行模型、基于动态分析的并行执行模型、节点间并行执行模型和分治并行执行模型.最后,从多个方面对比现有的各种智能合约并行执行方法,讨论了影响智能合约并行性的因素,并介绍了未来可研究的方向.希望本综述能为研究者对区块链的性能研究带来一个新的角度.

References:

- [1] Yuan Y, Wang FY. Blockchain: The state of the art and future trends. *Acta Automatica Sinica*, 2016, 42(4): 481–494 (in Chinese with English abstract).
- [2] Nakamoto S. Bitcoin: A peer-to-peer electronic cash system. *Decentralized Business Review*, 2008: 21260.
- [3] Ethereum. Yellow paper. <https://ethereum.github.io/yellowpaper/paper.pdf>
- [4] Singh S, Singh N. Blockchain: Future of financial and cyber security. In: *Proc. of the 2nd Int'l Conf. on Contemporary Computing and Informatics (IC3I)*. Greater Noida: IEEE, 2016. 463–467. [doi: 10.1109/IC3I.2016.7918009]
- [5] Korpela K, Hallikas J, Dahlberg T. Digital supply chain transformation toward blockchain integration. In: *Proc. of the 50th Hawaii Int'l Conf. on System Sciences*. Hawaii, 2017.
- [6] Bermeo-Almeida O, Cardenas-Rodriguez M, Samaniego-Cobo T, Ferruzola-Gómez E, Cabezas-Cabezas R, Bazán-Vera W. Blockchain in agriculture: A systematic literature review. In: *Proc. of the Int'l Conf. on Technologies and Innovation*. Guayaquil: Springer, 2018. 44–56. [doi: 10.1007/978-3-030-00940-3_4]
- [7] Ferrag MA, Shu L, Yang X, Derhab A, Maglaras L. Security and privacy for green iot-based agriculture: Review, blockchain solutions, and challenges. *IEEE Access*, 2020, 8: 32031–32053. [doi: 10.1109/ACCESS.2020.2973178]
- [8] Yang WT, He P, Yang Z, Yi XC, Chen C. Digital copyright depository system enhanced by blockchain. In: *Proc. of the 2020 Int'l Conf. on Culture-oriented Science & Technology (ICCST)*. Beijing: IEEE, 2020. 198–202. [doi: 10.1109/ICCST50977.2020.00044]
- [9] Han PB, Sui AN, Jiang T, Gu CN. Copyright certificate storage and trading system based on blockchain. In: *Proc. of the 2020 IEEE Int'l Conf. on Advances in Electrical Engineering and Computer Applications (AEECA)*. Dalian: IEEE, 2020. 611–615. [doi: 10.1109/AEECA49918.2020.9213631]
- [10] Samaniego M, Jamsrandorj U, Deters R. Blockchain as a service for IOT. In: *Proc. of the 2016 IEEE Int'l Conf. on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*. Chengdu: IEEE, 2016. 433–436. [doi: 10.1109/iThings-GreenCom-CPSCom-SmartData.2016.102]
- [11] Huang JQ, Kong LH, Chen GH, Cheng L, Wu KS, Liu X. B-IoT: Blockchain driven Internet of Things with credit-based consensus mechanism. In: *Proc. of the 39th IEEE Int'l Conf. on Distributed Computing Systems (ICDCS)*. Dallas: IEEE, 2019. 1348–1357. [doi: 10.1109/ICDCS.2019.00135]
- [12] Zhang Y, Lin XD, Xu CX. Blockchain-based secure data provenance for cloud storage. In: *Proc. of the 20th Int'l Conf. on Information and Communications Security*. Lille: Springer, 2018. 3–19. [doi: 10.1007/978-3-030-01950-1]
- [13] Qian WN, Shao QF, Zhu YC, Jin CQ, Zhou AY. Research problems and methods in blockchain and trusted data management. *Ruan Jian Xue Bao/Journal of Software*, 2018, 29(1): 150–159 (in Chinese with English abstract). <http://www.jos.org.cn/1000-9825/5434.htm> [doi: 10.13328/j.cnki.jos.005434]
- [14] Lu T, Peng L. BPU: A blockchain processing unit for accelerated smart contract execution. In: *Proc. of the 57th ACM/IEEE Design Automation Conf. (DAC)*. San Francisco: IEEE, 2020. 1–6. [doi: 10.1109/DAC18072.2020.9218512]
- [15] Meneghetti A, Parise T, Sala M, Taufer D. A survey on efficient parallelization of blockchain-based smart contracts. *Annals of Emerging Technologies in Computing (AETiC)*, 2019, 3(5): 9–16. [doi: 10.33166/AETiC.2019.05.002]
- [16] Hazari SS, Mahmoud QH. A parallel proof of work to improve transaction speed and scalability in blockchain systems. In: *Proc. of the 9th IEEE Annual Computing and Communication Workshop and Conf. (CCWC)*. Las Vegas: IEEE, 2019. 916–921. [doi: 10.1109/CCWC.2019.8666535]
- [17] Gorenflo C, Lee S, Golab L, Keshav S. FastFabric: Scaling hyperledger fabric to 20000 transactions per second. *Int'l Journal of Network Management*, 2020, 30(5): e2099. [doi: 10.1002/nem.2099]
- [18] Zhao H. Alipay's self-developed database reaches 61 million transactions per second. 2019. <http://www.techweb.com.cn/internet/2019-11-11/2763343.shtml> (in Chinese).
- [19] David B, Gaži P, Kiayias A, Russell A. Ouroboros praos: An adaptively-secure, semi-synchronous proof-of-stake blockchain. In: *Proc. of the 37th Annual Int'l Conf. on the Theory and Applications of Cryptographic Techniques*. Tel Aviv: Springer, 2018. 66–98. [doi: 10.1007/978-3-319-78375-8_3]
- [20] Zou J, Ye B, Qu L, Wang Y, Orgun MA, Li L. A proof-of-trust consensus protocol for enhancing accountability in crowdsourcing services. *IEEE Trans. on Services Computing*, 2019, 12(3): 429–445. [doi: 10.1109/TSC.2018.2823705]
- [21] Wang Q, Li RJ. A weak consensus algorithm and its application to high-performance blockchain. In: *Proc. of the 2021 IEEE Conf. on Computer Communications*. Vancouver: IEEE, 2021. 1–10. [doi: 10.1109/INFOCOM42981.2021.9488725]
- [22] Li PL, Wang GS, Chen XQ, Long F, Xu W. Gosig: A scalable and high-performance byzantine consensus for consortium blockchains. In: *Proc. of the 11th ACM Symp. on Cloud Computing*. New York: Association for Computing Machinery, 2020. 223–237. [doi: 10.1145/

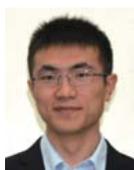
- 3419111.3421272]
- [23] Guo BY, Lu ZL, Tang Q, Xu J, Zhang ZF. Dumbo: Faster asynchronous BFT protocols. In: Proc. of the 2020 ACM SIGSAC Conf. on Computer and Communications Security. ACM, 2020. 803–818. [doi: 10.1145/3372297.3417262]
- [24] Liu J, Li PL, Cheng R, Asokan N, Song D. Parallel and asynchronous smart contract execution. IEEE Trans. on Parallel and Distributed Systems, 2022, 33(5): 1097–1108. [doi: 10.1109/TPDS.2021.3095234]
- [25] Zheng ZB, Xie SA, Dai HN, Chen WL, Chen XP, Weng J, Imran M. An overview on smart contracts: Challenges, advances and platforms. Future Generation Computer Systems, 2020, 105: 475–491. [doi: 10.1016/j.future.2019.12.019]
- [26] Bartoletti M, Galletta L, Murgia M. A true concurrent model of smart contracts executions. In: Proc. of the 22nd IFIP WG 6.1 Int'l Conf. on Coordination Models and Languages, COORDINATION 2020, Held as Part of the 15th Int'l Federated Conf. on Distributed Computing Techniques. Valletta: Springer, 2020, 12134: 243–260. [doi: 10.1007/978-3-030-50029-0_16]
- [27] Anjana PS, Kumari S, Peri S, Rathor S, Somani A. Entitling concurrency to smart contracts using optimistic transactional memory. In: Proc. of the 20th Int'l Conf. on Distributed Computing and Networking. Bangalore: ACM, 2019. 508–508. [doi: 10.1145/3288599.3299723]
- [28] Sharma A, Schuhknecht FM, Agrawal D, Dittrich J. Blurring the lines between blockchains and database systems: The case of hyperledger fabric. In: Proc. of the 2019 Int'l Conf. on Management of Data. Amsterdam: ACM, 2019. 105–122. [doi: 10.1145/3299869.3319883]
- [29] Huang HW, Kong W, Zhou SC, Zheng ZB, Guo S. A survey of state-of-the-art on blockchains: Theories, modelings, and tools. ACM Computing Surveys (CSUR), 2022, 54(2): 44. [doi: 10.1145/3441692]
- [30] Zhang ZW, Wang GR, Xu JL, Du XY. Survey on data management in blockchain systems. Ruan Jian Xue Bao/Journal of Software, 2020, 31(9): 2903–2925 (in Chinese with English abstract). <http://www.jos.org.cn/1000-9825/6091.htm> [doi: 10.13328/j.cnki.jos.006091]
- [31] Fan JL, Li XH, Nie TZ, Yu G. Survey on smart contract based on blockchain system. Computer Science, 2019, 46(11): 1–10 (in Chinese with English abstract). [doi: 10.11896/j.sjcx.190300013]
- [32] aelf. Aelf-A multi-chain parallel computing blockchain framework. 2018. https://aelf.io/gridcn/aelf_whitepaper_EN.pdf
- [33] Dinh TTA, Wang J, Chen G, Liu R, Ooi BC, Tan KL. BLOCKBENCH: A framework for analyzing private blockchains. In: Proc. of the 2017 ACM Int'l Conf. on Management of Data. Chicago: ACM, 2017. 1085–1100. [doi: 10.1145/3035918.3064033]
- [34] Szabo N. Formalizing and securing relationships on public networks. First Monday, 1997, 2(9). <https://journals.uic.edu/ojs/index.php/fm/article/view/548> [doi: 10.5210/fm.v2i9.548]
- [35] Sergey I, Hobor A. A concurrent perspective on smart contracts. In: Proc. of the 2017 Int'l Conf. on Financial Cryptography and Data Security. Sliema: Springer, 2017. 478–493. [doi: 10.1007/978-3-319-70278-0_30]
- [36] Anjana PS, Kumari S, Peri S, Rathor S, Somani A. An efficient framework for optimistic concurrent execution of smart contracts. In: Proc. of the 27th Euromicro Int'l Conf. on Parallel, Distributed and Network-based Processing (PDP). Pavia: IEEE, 2019. 83–92. [doi: 10.1109/EMPDP.2019.8671637]
- [37] Anjana PS, Kumari S, Peri S, Rathor S, Somani A. OptSmart: A space efficient optimistic concurrent execution of smart contracts. arXiv:2102.04875, 2021.
- [38] Anjana PS, Attiya H, Kumari S, Peri S, Somani A. Efficient concurrent execution of smart contracts in blockchains using object-based transactional memory. In: Proc. of the 8th Int'l Conf. on Networked Systems. Marrakech: Springer, 2020. 77–93. [doi: 10.1007/978-3-030-67087-0_6]
- [39] Amiri MJ, Agrawal D, El Abbadi A. Parblockchain: Leveraging transaction parallelism in permissioned blockchain systems. In: Proc. of the 39th IEEE Int'l Conf. on Distributed Computing Systems (ICDCS). Dallas: IEEE, 2019. 1337–1347. [doi: 10.1109/ICDCS.2019.00134]
- [40] Shrey B, Singh AP, Sathya P, Yogesh S. DiPETrans: A framework for distributed parallel execution of transactions of blocks in blockchain. arXiv:1906.11721v1, 2019.
- [41] FISCOBCOS. <https://fisco-bcos-documentation.readthedocs.io/en/latest/docs/introduction.html>
- [42] Yu W, Luo K, Ding Y, You G, Hu K. A parallel smart contract model. In: Proc. of the 2018 Int'l Conf. on Machine Learning and Machine Intelligence. Ha Noi Viet Nam: ACM, 2018. 72–77. [doi: 10.1145/3278312.3278321]
- [43] Dickerson T, Gazzillo P, Herlihy M, Koskinen E. Adding concurrency to smart contracts. Distributed Computing, 2020, 33(3): 209–225. [doi: 10.1007/s00446-019-00357-z]
- [44] Zhang A, Zhang KL. Enabling concurrency on smart contracts using multiversion ordering. In: Proc. of the 2nd Int'l Joint Conf. on Web and Big Data. Macao: Springer, 2018. 425–439. [doi: 10.1007/978-3-319-96893-3_32]
- [45] Solidity. <https://solidity.readthedocs.io>

- [46] Saraph V, Herlihy M. An empirical study of speculative concurrency in ethereum smart contracts. In: Proc. of the Int'l Conf. on Blockchain Economics, Security and Protocols (Tokenomics 2019). Paris: ACM, 2020. 4.
- [47] Pang SF, Qi XD, Zhang Z, Jin CQ, Zhou AY. Concurrency protocol aiming at high performance of execution and replay for smart contracts. arXiv:1905.07169, 2019.
- [48] Wüst K, Matetic S, Egli S, Kostianin K, Capkun S. ACE: Asynchronous and concurrent execution of complex smart contracts. In: Proc. of the 2020 ACM SIGSAC Conf. on Computer and Communications Security. ACM, 2020. 587–600. [doi: 10.1145/3372297.3417243]
- [49] Rong P. Use divide and conquer to solve the parallel transaction problem of the blockchain. 2018. <https://www.infoq.cn/article/divide-and-conquer-in-blockchain>
- [50] Androulaki E, Barger A, Bortnikov V, Cachin C, Christidis K, De Caro A, Enyeart D, Ferris C, Laventman G, Manevich Y, Muralidharan S, Murthy C, Nguyen B, Sethi M, Singh G, Smith K, Sorniotti A, Stathakopoulou C, Vukolić M, Cocco SW, Yellick J. Hyperledger fabric: A distributed operating system for permissioned blockchains. In: Proc. of the 13th EuroSys Conf. Porto: ACM, 2018. 1–15. [doi: 10.1145/3190508.3190538]
- [51] Wang JP, Wang H. Monoxide: Scale out blockchain with asynchronous consensus zones. In: Proc. of the 16th USENIX Symp. on Networked Systems Design and Implementation (NSDI 19). Boston: USENIX, 2019. 95–112. [doi: 10.13140/RG.2.2.32017.48489]
- [52] Wang ZK. Research on parallelization of grouping-based smart contract's execution [MS. Thesis]. Tianjin: Tianjin University, 2018 (in Chinese with English abstract).
- [53] Cook V, Painter Z, Peterson C, Dechev D. Read-uncommitted transactions for smart contract performance. In: Proc. of the 39th IEEE Int'l Conf. on Distributed Computing Systems (ICDCS). Dallas: IEEE, 2019. 1960–1970. [doi: 10.1109/ICDCS.2019.00194]
- [54] Eswaran KP, Gray JN, Lorie RA, Traiger IL. The notions of consistency and predicate locks in a database system. Communications of the ACM, 1976, 19(11): 624–633. [doi: 10.1145/360363.360369]
- [55] Garcia-Molina H, Ullman JD, Widom JD. Database System Implementation. Upper Saddle River: Prentice Hall, 2000. 672.
- [56] Ghosh A, Chaki R, Chaki N. A new concurrency control mechanism for multi-threaded environment using transactional memory. The Journal of Supercomputing, 2015, 71(11): 4095–4115. [doi: 10.1007/s11227-015-1507-8]
- [57] He XY. Research and comprehensive optimization of software transaction memory algorithm in parallel computing programming [MS. Thesis]. Shanghai: Fudan University, 2011 (in Chinese with English abstract).
- [58] XuperChain. <https://xuper.baidu.com/n/xuperdoc/index.html> (in Chinese).
- [59] Yu L, Tsai WT, Li GN, Yao YF, Hu CJ, Deng EY. Smart-contract execution with concurrent block building. In: Proc. of the 2017 IEEE Symp. on Service-Oriented System Engineering (SOSE). San Francisco: IEEE, 2017. 160–167. [doi: 10.1109/SOSE.2017.33]
- [60] Wang G, Shi ZJ, Nixon M, Han S. SoK: Sharding on blockchain. In: Proc. of the 1st ACM Conf. on Advances in Financial Technologies. Zurich: ACM, 2019. 41–61. [doi: 10.1145/3318041.3355457]
- [61] Yu GS, Wang X, Yu K, Ni W, Zhang JA, Liu RP. Survey: Sharding in blockchains. IEEE Access, 2020, 8: 14155–14181. [doi: 10.1109/ACCESS.2020.2965147]
- [62] Dang H, Tien TAD, Lohin 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: ACM, 2019. 123–140. [doi: 10.1145/3299869.3319889]
- [63] The ZILLIQA Team. The ZILLIQA technical whitepaper. 2017. <https://docs.zilliqa.com/whitepaper.pdf>
- [64] Zamani M, Movahedi M, Raykova M. Rapidchain: Scaling blockchain via full sharding. In: Proc. of the 2018 ACM SIGSAC Conf. on Computer and Communications Security. Toronto: ACM, 2018. 931–948. [doi: 10.1145/3243734.3243853]
- [65] QuarkChain. <https://www.quarkchain.io/wp-content/uploads/2018/11/QUARK-CHAIN-Public-Version-0.3.5.pdf>
- [66] Al-Bassam M, Sonnino A, Bano S, Hrycyszyn D, Danezis G. Chainspace: A sharded smart contracts platform. arXiv:1708.03778, 2017.
- [67] 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 (SP). San Francisco: IEEE, 2018. 583–598. [doi: 10.1109/SP.2018.000-5]
- [68] Luu L, Narayanan V, Zheng CD, Baweja K, Gilbert S, Saxena P. A secure sharding protocol for open blockchains. In: Proc. of the 2016 ACM SIGSAC Conf. on Computer and Communications Security. Vienna: ACM, 2016. 17–30. [doi: 10.1145/2976749.2978389]
- [69] Androulaki E, Cachin C, De Caro A, Kokoris-Kogias E. Channels: Horizontal scaling and confidentiality on permissioned blockchains. In: Proc. of the 23rd European Symp. on Research in Computer Security. Barcelona: Springer, 2018. 111–131. [doi: 10.1007/978-3-319-99073-6_6]
- [70] Thakkar P, Nathan S, Viswanathan B. Performance benchmarking and optimizing hyperledger fabric blockchain platform. In: Proc. of the 26th IEEE Int'l Symp. on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS). Milwaukee: IEEE, 2018. 264–276. [doi: 10.1109/MASCOTS.2018.00034]
- [71] Etherscan. <https://etherscan.io/charts>

- [72] ChainMaker. <https://docs.chainmaker.org.cn/index.html> (in Chinese).
- [73] Li HZ, Li CX, Li HX, Bai XQ, Shi X. An overview on practice of FISCO BCOS technology and application. Information and Communications Technology and Policy, 2020, 46(1): 52–60 (in Chinese with English abstract). [doi: 10.3969/j.issn.1008-9217.2020.01.011]
- [74] Zou WQ, Lo D, Kochhar PS, Le XBD, Xia X, Feng Y, Chen ZY, Xu BW. Smart contract development: Challenges and opportunities. IEEE Trans. on Software Engineering, 2021, 47(10): 2084–2106. [doi: 10.1109/TSE.2019.2942301]
- [75] Liquid. https://liquid-doc.readthedocs.io/zh_CN/latest/ (in Chinese).
- [76] Ruan PC, Loghin D, Ta QT, Zhang MH, Chen G, Ooi BC. A transactional perspective on execute-order-validate blockchains. In: Proc. of the 2020 ACM SIGMOD Int'l Conf. on Management of Data. Portland: ACM, 2020. 543–557. [doi: 10.1145/3318464.3389693]
- [77] Pirlea G, Kumar A, Sergey I. Practical smart contract sharding with ownership and commutativity analysis. In: Proc. of the 42nd ACM SIGPLAN Int'l Conf. on Programming Language Design and Implementation. ACM, 2021. 1327–1341. [doi: 10.1145/3453483.3454112]
- [78] Tao YC, Li B, Jiang JJ, Ng HC, Wang C, Li BC. On sharding open blockchains with smart contracts. In: Proc. of the 36th IEEE Int'l Conf. on Data Engineering (ICDE). Dallas: IEEE, 2020. 1357–1368. [doi: 10.1109/ICDE48307.2020.00121]
- [79] Wang Y, Li JX, Liu WS, Tan AP. Efficient concurrent execution of smart contracts in blockchain sharding. Security and Communication Networks, 2021, 2021: 6688168. [doi: 10.1155/2021/6688168]
- [80] Xiao W, Sun JY, Zheng Q, Chen u. XuperChain: A blockchain system that supports smart contracts parallelization. In: Proc. of the 2020 IEEE Int'l Conf. on Smart Internet of Things (SmartIoT). Beijing: IEEE, 2020. 309–313. [doi: 10.1109/SmartIoT49966.2020.00055]

附中文参考文献:

- [1] 袁勇, 王飞跃. 区块链技术发展现状与展望. 自动化学报, 2016, 42(4): 481–494.
- [13] 钱卫宁, 邵奇峰, 朱燕超, 金澈清, 周傲英. 区块链与可信数据管理: 问题与方法. 软件学报, 2018, 29(1): 150–159. <http://www.jos.org.cn/1000-9825/5434.htm> [doi: 10.13328/j.cnki.jos.005434]
- [18] 朝晖. 支付宝自研数据库每秒处理峰值达到6100万次. 2019. <http://www.techweb.com.cn/internet/2019-11-11/2763343.shtml>
- [30] 张志威, 王国仁, 徐建良, 杜小勇. 区块链的数据管理技术综述. 软件学报, 2020, 31(9): 2903–2925. <http://www.jos.org.cn/1000-9825/6091.htm> [doi: 10.13328/j.cnki.jos.006091]
- [31] 范吉立, 李晓华, 聂铁铮, 于戈. 区块链系统中智能合约技术综述. 计算机科学, 2019, 46(11): 1–10. [doi: 10.11896/jsjx.190300013]
- [49] 戎朋. 用分治思路解决区块链并行化交易问题. 2018. <https://www.infoq.cn/article/divide-and-conquer-in-blockchain>
- [52] 王正凯. 基于分组的智能合约并行化研究 [硕士学位论文]. 天津: 天津大学, 2018.
- [57] 和小元. 并行计算编程中的软件事务内存算法研究与综合优化 [硕士学位论文]. 上海: 复旦大学, 2011.
- [58] XuperChain. <https://xuper.baidu.com/n/xuperdoc/index.html>
- [72] ChainMaker. <https://docs.chainmaker.org.cn/index.html>
- [73] 李辉忠, 李陈希, 李昊轩, 白兴强, 石翔. FISCO BCOS技术应用实践. 信息通信技术与政策, 2020, 46(1): 52–60. [doi: 10.3969/j.issn.1008-9217.2020.01.011]
- [75] Liquid. https://liquid-doc.readthedocs.io/zh_CN/latest/



施建锋(1992—), 男, 博士生, 主要研究领域为区块链智能合约.



高赫然(1997—), 男, 博士生, 主要研究领域为存储系统.



吴恒(1983—), 男, 博士, 副研究员, CCF 专业会员, 主要研究领域为分布式系统, 云计算.



张文博(1976—), 男, 博士, 研究员, 博士生导师, CCF 专业会员, 主要研究领域为分布式系统, 云计算.