计 算 机 研 究 与 发 展
Journal of Computer Research and Development
DOI: 10.7544/issn1000-1239.202330195
61(6): 1559−1572, 2024

MC-RHotStuff: 面向多链基于信誉的 HotStuff 共识机制

路宇轩　孔兰菊　张宝晨　闵新平
（山东大学软件学院　济南　250100）

MC-RHotStuff: Multi-Chain Oriented HotStuff Consensus Mechanism Based on Reputation

Lu Yuxuan, Kong Lanju, Zhang Baochen, and Min Xinping
（*School of Software*, *Shandong University* , *Jinan* 250100）

Abstract　The existing blockchain presents a multi-chain trend. Traditional consensus algorithms do not have dynamic scalability for multi-chain, making it difficult to cope with the contradiction between open use and closed maintenance of large-scale permissioned blockchain. For this problem, a novel multi-chain consensus algorithm, MC-RHotStuff, is proposed: Nodes have different roles, including alternative nodes, candidate nodes, and consensus nodes. Each working chain has consensus nodes and alternative nodes. After the admission verification, the candidate node will become an alternative node; A consensus node has a reputation value that other nodes do not have, and a consensus node that performs the correct behavior will increase the reputation value, while a consensus node that performs the wrong behavior will deduct the reputation value, then the node with abnormal reputation value will be found through the node reputation calculation and filtering algorithm MC-Scan, and a new consensus node will be selected from the alternative nodes to exchange with the abnormal node. In addition, a dynamic node adjustment algorithm, MC-Schedule, is proposed to achieve optimization by detecting the transaction volume of each blockchain and dynamically adjusting the number of consensus nodes, which not only ensures the efficient execution of the blockchain system but also improves the speed of node filtering. MC-RHotStuff proposes a node state synchronization mechanism, MC-Syn, to ensure that the consensus operates normally when the node number change or the consensus group change. Compared with existing systems, transaction throughput and latency have been comprehensively improved by about 15%.

Key words　blockchain; multi-chain architecture; reputation value; consensus algorithm; distributed network

摘　要　现有区块链呈现出多链趋势，传统共识算法不具备面向多链的动态扩展性，难以应对大规模联盟链开放使用与封闭运维的矛盾. 对此，提出了一种新颖的多链共识方法 MC-RHotStuff: 节点划分不同的角色，分为待准入节点、备选节点、共识节点，每条工作链都拥有共识节点和备选节点，待准入节点完成准入验证后将成为备选节点；共识节点拥有其他节点不具备的信誉值，做出正确行为的共识节点将提高信誉值，做出错误行为的共识节点将扣除信誉值，通过节点信誉计算及筛选算法 MC-Scan 来寻找信誉值异常的节点，并从备选节点中选择新的共识节点与异常节点交换. 此外，还提出了节点动态调整算法 MC-

收稿日期：2023-03-27；**修回日期**：2023-07-25
基金项目：国家重点研发计划项目（2021YFB2700102）；国家社会科学基金项目（20BJY131）；山东省重大科技创新项目（2020CXGC010106, 2021CXGC010108）
This work was supported by the National Key Research and Development Program of China (2021YFB2700102), the National Social Science Fund Project of China (20BJY131), and the Major Science and Technology Innovation Project of Shandong Province (2020CXGC010106, 2021CXGC010108).
通信作者：孔兰菊 (klj@sdu.edu.cn)

Schedule, 通过检测每条区块链的交易量从而动态地调整共识节点的数量来达到最优化, 既保证区块链系统的高效执行, 又提高了节点筛选的速度. 为保证当节点数量变化或共识群组成员发生变化时, 共识机制能够正常运转, 提出了节点状态同步机制 MC-Syn.对此进行了大量的实验来验证 MC-RHotStuff 性能, 与现有系统相比, 其交易吞吐量和延迟综合提升约 15%.

关键词 区块链; 多链架构; 信誉值; 共识算法; 分布式网络

中图法分类号 TP391

传统联盟链通过 PBFT[1], RAFT[2], DIEM[3] 等拜占庭容错协议保证共识正常进行, $n \geqslant 3f+1$是保证共识协议能够正常运行的必要条件[4-5]. 若异常节点f数量较多, 采用轮换机制选举 leader 的方法会使视图更替变得频繁, 为了改善其性能, RE-PBFT[6], PROOF-OF-REPUTATION[7], WRBFT[8] 等共识算法引入了节点信誉计算机制[9] 作为节点的性能指标, 信誉越高的节点会获得更高当选 leader 的机会. 虽然这些评判机制可以避免异常节点多次当选 leader 节点, 但异常节点依然存在于共识群组当中, 若此时f因为不可控因素变为$f+1$, 节点总数$n=3f+1$, 将无法满足拜占庭容错的最低条件, 共识的进程无法得到保障. 如图 1 所示, 目前的解决方案需要人为更换异常节点, 将异常节点从共识群组中剔除, 以消除$f>\dfrac{n-1}{3}$的可能性.

如图 2 所示, 随着区块链规模增大, 出现了多链

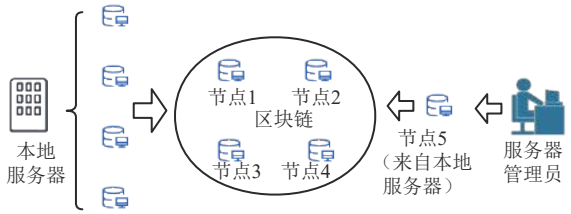


Fig. 1 Management mode of traditional blockchain

图 1 传统区块链管理模式

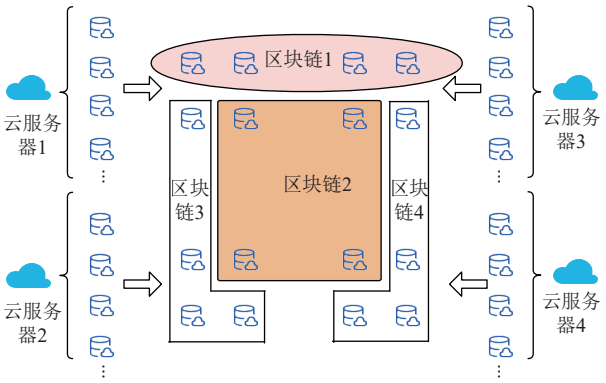


Fig. 2 Multi-chain management mode of large-scale permissioned blockchain

图 2 大规模许可区块链多链管理模式

系统[10] 以处理不同的业务[11-12], 节点规模动辄成百上千, 节点来源也从封闭式的单位局域网变成更加开放的云环境, 信誉值共识算法在面对异常节点出现的情况时, 封闭式的手动替换节点需要终止共识、同步区块状态等, 会消耗大量时间, 相比于节点可以随意加入或离开的公有链, 其开放能力还有差距[13]. 另一方面, 基于信誉的共识算法, 其共识节点数量相对比较固定, 并未考虑根据交易量进行调整, 存在极大的浪费, 难以达到最优化的区块链性能.

本文改善现有的信誉值机制, 提出了新的多链共识机制 MC-RHotStuff. 其不仅能够选择性能最佳的节点当选 leader, 还能标记异常节点, 随后用自动的节点替换机制将异常节点替换掉; 并且, 共识群组的规模是可以根据交易量变化的, 使区块链系统始终保持最佳的状态. 为了实现上述功能, 每个节点都应被赋予不同的角色, 节点根据不同的角色被划分为共识节点和备选节点, 当共识节点异常时, 备选节点可以替换异常节点成为新的共识节点, 且不干扰正常的共识进程, 不会因为节点的替换而导致系统重启.

本文的主要贡献包括 3 个方面:

1)引入多角色节点概念, 不同角色的节点负责不同区块链上的不同任务, 增强对大规模多链联盟链管理的能力.

2)引入适用于大规模多链联盟链环境下节点自动替换功能的信誉机制, 解决共识群组存在异常节点的问题, 保证节点的活性, 避免因为存在拜占庭节点而导致效率低下.

3)引入节点动态调整方法, 可以灵活地根据交易量改变共识群组的成员及节点数量, 避免计算资源的浪费.

1 相关工作

1.1 信誉值无关的共识算法

信誉值无关的共识算法多指通过轮转或其他特殊选择方法选择节点当选 leader 节点的共识算法, 主

要代表有 PBFT, Sbft[14], HotStuff[15] 等部分同步共识算法[16], 只有在 GST[17] 之后, 才能确保同步.

即使只计算数字签名或消息验证码[18], 在 PBFT 中传递一个新的区块也有 $O(n^2)$ 的通信开销, n 为工作链节点数量. 如果在决定达成之前发生视图更改, 则传输开销变为 $O(n^3)$. SBFT 引入门限签名[19] 后, 可以将视图更改的开销降为 $O(n^2)$. 尽管如此, 在多链环境下, 交易需要通过协作链分流到不同的工作链中处理[20], 此部分的通信开销为 $O(n)$, 完成交易共识后, 工作链节点会向协作链节点广播新区块, 此时的通信开销为 $O(nm)$, m 为协作链节点数量, 则完成 1 笔交易总的通信开销为 $O(n+n^2+nm)$, 若采用 PBFT 的最差情况下出现视图更替, 总体的开销将达到立方数量级.

HotStuff 引入门限签名技术, 将节点广播投票改成 leader 节点来收集投票然后再广播, 从而将通信的复杂度从 $O(n^2)$ 降低为 $O(n)$, 即使出现视图更改的情况, 其最大通信开销也仅为 $O(n^2)$. 因此, 在正常情况下, 多链共识的通信开销仅为 $O(2n+nm)$, 若能合理地控制 m 的数量, 使得 $m \ll n$, 多链共识的通信成本可控制在常量级.

Boneh 等人[21] 在 2004 年提出的 BLS12 加密算法是一种可以实现签名聚合和密钥聚合的算法, 它不需要随机数生成器, 可以将区块中的所有签名聚合成 1 个, 容易实现 $m-n$ 多重签名[22], 也可以避免签名者之间的多余通信, 从而进一步降低通信成本.

即便如此, 拜占庭节点带来的代价仍需进一步降低, 大量研究者开始考虑基于信誉值的共识算法.

1.2 信誉值共识算法

信誉值共识算法是通过信誉值来衡量一个节点性能的指标, 可以选举信誉值高的节点作为 leader 节点, 用来降低共识成本, 提升共识性能.

在 RE-PBFT 共识算法中, 节点在共识的过程中记录其他节点的行为, 共识结束后, 节点将行为记录表发送到智能合约, 智能合约根据每个节点的行为修改每个节点的信誉值, 并将该信誉值发送给每个节点, 节点根据更新后的信誉值选择信誉值最高的节点当选新的 leader 节点.

PROOF-OF-REPUTATION 的信誉值计算过程与 RE-PBFT 类似, 也是通过共识过程中节点的交互进行信誉值的计算, 但与此不同的是, leader 的选举不再以最高信誉值为准, 而是为每个节点分配 1 个权重, 该权重为该节点信誉值占整体信誉值的比重, 随后采用随机算法, 随机挑选一个节点作为 leader 节点.

WRBFT 引入了不同角色的节点, 共识群组中节点分为高信誉节点和低信誉节点, 尽管每个节点都参与共识, 但只要高信誉节点中超过阈值的节点对提议达成共识, 共识就算成功, 这使得共识群组具有可以容纳超过 f 个拜占庭节点的能力.

尽管有很多的节点信誉评判机制来衡量节点的行为, 但它们都是针对当前共识群组而设的, 异常节点还是存在于共识群组当中, 这使得异常节点仍然可以参与共识, 增加了共识失败的风险.

如何安全替换异常节点以确保共识正常运行, 是亟需解决的问题, 本文提出面向多链基于信誉的共识机制 MC-RHotStuff, 解决共识群组存在异常节点的问题, 保证节点的活性, 避免因为存在拜占庭节点而导致的效率低下. 同时支持动态调整共识节点数量: 一方面是为了在吞吐量需求大的情况下提升共识性能, 避免计算资源的浪费; 另一方面, 在吞吐量需求不大的情况下, 适当增加节点的数量, 在不影响交易正常执行的情况下, 提高筛选异常节点的速度, 使共识群组的网络环境更加安全, 不会因为存在过多异常节点而导致共识的失败.

1.3 节点多角色划分

目前已经有许多研究将区块链的节点划分为多种不同的角色, 并执行不同的任务. 例如 Hyperledger Fabric 支持多类节点角色, Orderers 节点负责提供共识服务, Peers 负责担任背书或记账服务, Comitters 负责检查交易的合法性, 最终将交易提交到区块链中. Hyperledger Fabric 的角色划分目的是为了更好地推进共识, 但是, 如果某类节点发生了较多的故障, 共识可能会因此收到很大的影响. 而本文的角色划分则更注重于解决此类问题, MC-RHotStuff 将节点划分为备选节点和共识节点, 若将 MC-RHotStuff 的思想运用到 Hyperledger Fabric 中, 将体现为处于共识的节点和处于备选的节点, MC-RHotStuff 将实时监控正在共识的节点, 一旦其性能与正常节点不匹配, 就会启用处于后备队列的节点顶替它执行共识任务.

WRBFT 也引入了备选节点的概念用于解决共识节点性能低的问题, 但与本文思路不同的是, WRBFT 的备选节点和共识节点都是存在于共识群组当中的, 一旦创建其网络结构就不会发生改变, 节点没有动态地加入和删除过程, 节点群组不会发生变化, 每轮共识根据其性能的反馈来选择部分节点作为共识节点使用, 但不论如何选择, 节点池的构成并没有发生变化. 而本文实现了节点的动态加入和删除, 一旦共识节点的性能达不到要求, 就会被备选

节点所替代，并且被替代的节点将会被踢出节点池，而备选节点组中会源源不断地有新的节点加入，这保证了共识节点不是在原有的节点池中选择相对较好的节点，而是在不断更新的节点池中选择最优性能的节点，性能提升相比 WRBFT 会更多.

2 MC-RHotStuff 系统框架

本节将详细介绍 MC-RHotStuff 的系统构成以及事务处理、节点准入、节点动态调整 MC-Schedule 流程，并阐述节点信誉值计算及筛选算法 MC-Scan、节点状态同步算法 MC-Syn 是如何实现的.

2.1 节点角色划分

如图 3 所示，本文将节点划分为待准入节点、工作链备选节点、工作链共识节点、协作链共识节点 4 类节点. 待准入节点为某个云服务器中尚未加入区块链的节点[23]，它无法参与共识；若待准入节点通过协作链共识节点审核[24]并获得相应证书后[25]，其身份会变为工作链备选节点，并隶属于某个工作链当中. 本文规定，每个工作链上的备选节点个数 $M=\frac{N-1}{3}$，N 为当前共识节点的个数，当该链上备选节点个数超过 M 时则停止为该链分配备选节点，若每个链都具备 M 个备选节点时，则停止节点的准入. 工作链共识节点为参与当前区块链共识的节点，它可以产生区块、处理交易、评判其他共识节点，当其信誉值过低时，会被备选节点替换. 协作链共识节点不具备信誉值，采用传统的共识算法进行共识.

为了更好地区分，本文将节点的身份用符号来表示：

C_i 表示当前节点为待准入节点，i 为当前第 i 待准入节点；

N_i 表示当前节点为协作链共识节点，i 为当前第 i 协作链共识节点，注意协作链不存在备选节点；

$W_j.A_i$ 表示当前第 j 工作链上第 i 备选节点；

$W_j.N_i$ 表示当前第 j 工作链上第 i 共识节点.

2.2 系统概述

如图 4 所示，本文的多链架构分为工作链和协作链的形式，协作链分流客户端发来的交易，将不同的事务划分到不同的工作链进行处理.

协作链上的共识节点负责监视每条工作链的瞬时交易量，若发现交易量较之前改变较大，就会在协作链上发起新一轮共识，这与工作链的交易共识不同，其目的是改变当前工作链上的节点数量来满足吞吐量的需求. 当有新的待准入节点申请准入时，协作链节点也会发起一轮新的共识，目的是审核待准入节点，当共识达成后，待准入节点将会获得准入证书并被分配到工作链的节点池中，协作链节点会将该准入节点加入到自身维护的节点列表中，列表记录了所有工作链节点，包括它们属于哪个工作链、目前属于什么角色，此外，协作链节点会将新节点的状态同步给相应的工作链节点，目的是当新节点加入后，保证协作链和工作链节点状态一致.

工作链共识节点将新交易打包成新区块，随后进行交易共识，当共识完成后，工作链节点首先会更新本地区块链状态，其次会发送该区块到协作链，协作链通过区块可以同步该工作链上节点和区块链的最新状态，当发生节点替换时，协作链没法通过产生的新区块获得最新的工作链节点状态，这时，工作链会采用节点状态同步算法 MC-Syn 将节点状态同步给协作链.

协作链采用基本的 HotStuff 算法运行，为了更好地提升工作链性能，区别于传统的信誉激励机制，本

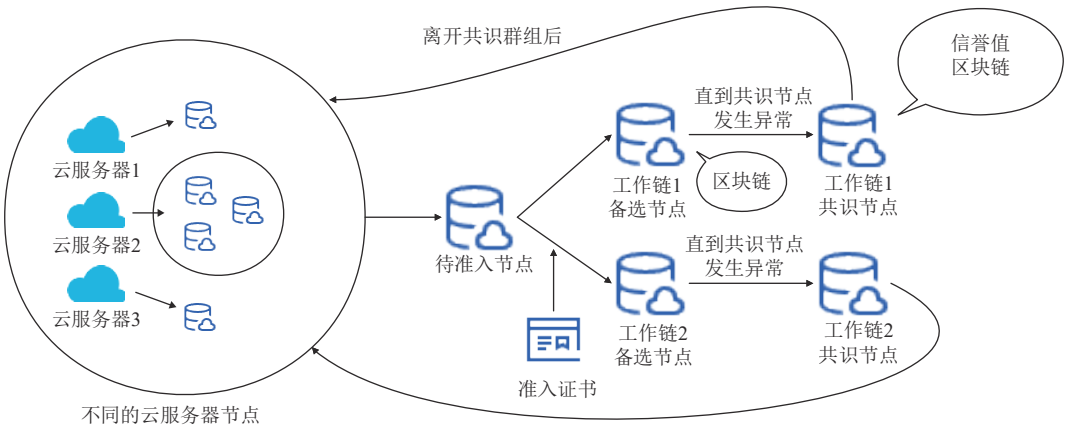


Fig. 3 Node role division

图 3 节点角色划分

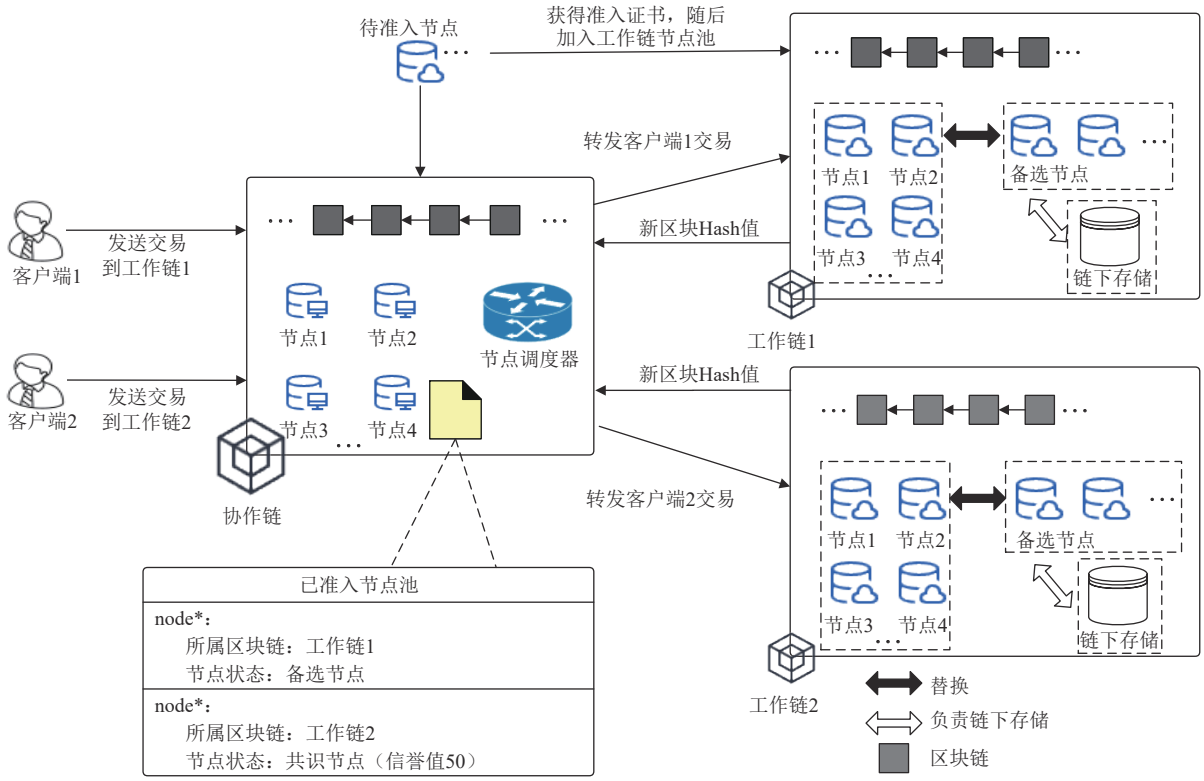


Fig. 4 MC-RHotStuff system framework

图 4 MC-RHotStuff 系统框架

文通过信誉机制的评判实现 2 个目标：一方面将信誉值最高的工作链共识节点作为 leader 发起新一轮的共识；另一方面，将信誉值作为离群因素，通过聚类算法筛选出信誉值低（离群因素过大）的工作链节点，并将其判定为异常的共识节点，然后该异常节点与备选节点交换，使备选节点成为新的共识节点，从而动态配置共识节点群.

此外，备选节点可向外部提供查询接口，返回当前工作链状态的最新值，此步骤交由备选节点完成，共识节点无需负责，可以减少共识节点的网络带宽占用. 并且，备选节点采用区块卸载机制，通过 LRU[26] 算法，将过时的区块由链上卸载到链下[27-28]，这也同样减少了共识节点的网络带宽.

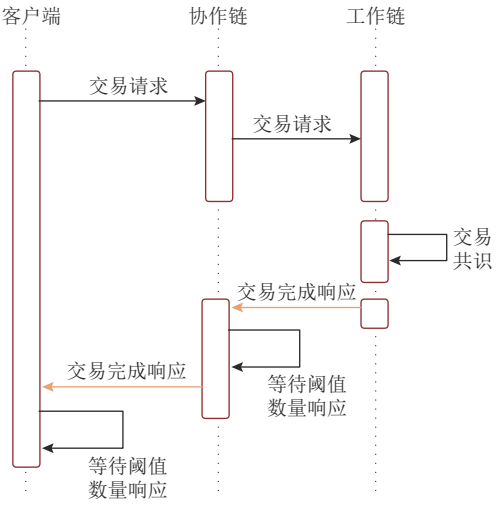


Fig. 5 Basic transaction process

图 5 基本交易事务流程

3 MC-RHotStuff 流程及算法

3.1 基本交易事务

基本交易事务由工作链负责共识. 基本交易事务流程如图 5 所示. 基本交易事务流程分为 5 个步骤：

1）客户端发送一笔交易 TX 到协作链，该交易 TX 为某个工作链应处理的交易；

2）协作链节点 N_i 收到交易 TX 后，会将该交易转播给目标工作链中的当前所有的共识节点 $W_j.N_1$，$W_j.N_2$，…，$W_j.N_n$；

3）目标工作链的节点 $W_j.N_{\text{leader}}$ 收到交易后，如果该交易不是过期交易且通过节点选举算法得知自己是主节点后，就将该笔交易放入自己构造的区块中进行共识；

4）当共识成功后，各个工作链节点 $W_j.N_1$，$W_j.N_2$，…，$W_j.N_n$，和 $W_j.A_1$，$W_j.A_2$，…，$W_j.A_m$（该工作

链所有的节点)会更新自己本地的区块链, $W_j.N_1$, $W_j.N_2$, …, $W_j.N_n$将产生的新区块广播回协作链;

5)当协作链节点收到足够阈值的消息后,就会将区块 Hash 链接到协作链保存的工作链区块头中,并会根据区块中的参与者更新自己的节点列表,例如, $W_j.N_i$为异常节点,则更新后的共识节点变为 $W_j.N_1$, $W_j.N_2$, …, $W_j.N_{i-1}$, $W_j.A_1$, $W_j.N_{i+1}$, $W_j.N_{i+2}$, …, $W_j.N_n$, 备选节点变为$W_j.A_2$, $W_j.A_3$, …, $W_j.A_m$.

3.2 节点准入

节点准入由协作链负责共识. 如图 6 所示,节点准入流程由 4 个步骤构成:

1)协作链的节点N_i收到足够数量的准入节点请求后,将这些请求封装到一个新的区块中作为新一轮共识,这些待准入节点都拥有期望加入的工作链 ID;

2)一旦共识达成,说明多数的协作链节点认可新的待准入节点,待准入节点一定是按照协作链所制定的安全规则接入网络的,其身份是可认证的,由协作链共识保证其安全性. 协作链节点N_1, N_2, …, N_n会更新自己的工作链节点列表,将新的待准入节点加入进去;

3)待准入节点C_i的身份将变成备选节点$W_j.A_i$, C_i将获得准入证书,并被放到相应的工作链节点池中;

4)协作链节点N_1, N_2, …, N_n将相应工作链W_j最新的节点信息发送给对应工作链的节点$W_j.N_1$, $W_j.N_2$, …, $W_j.N_n$和$W_j.A_1$, $W_j.A_2$, …, $W_j.A_m$, 等收取到超过阈值的协作链发送的信息后,这些工作链节点更新自己的本地节点列表,如果共识节点增加,将从备选节点中选取

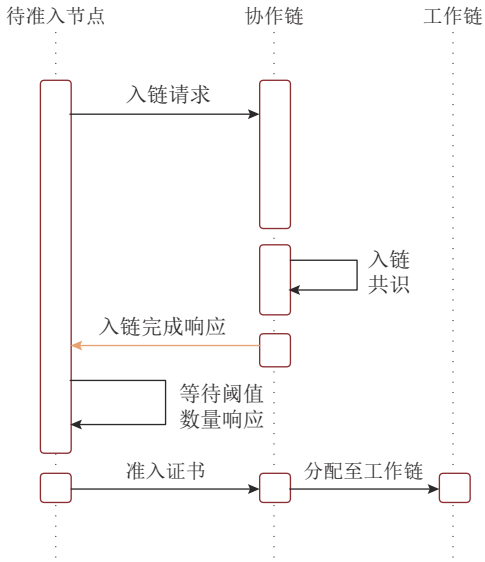


Fig. 6 Node admission process

图 6 节点准入流程

更新自己的本地节点列表,将$W_j.A_i$加入到备选节点组中.

3.3 MC-Schedule 节点调度

节点调度由协作链负责共识. 如图 7 所示,节点调度流程由 4 个步骤构成:

1)协作链节点负责监视每个工作链上的最近一个区块的平均交易量大小,如果改变幅度超过 20%时,将会根据式(1)改变当前工作链W_j共识节点的数量.

$$N' = N \times (2 - X). \tag{1}$$

注意,改变的共识节点数量应符合共识机制的数量要求,如 4,7,10 等.

2)MC-Schedule 节点调度算法的实现要根据具体情况,例如,以 C 为交易量起始值的阈值,共识节点数量为 N, 最新的交易量C'为例,计算它们的比值 $X = C'/C$.

当 X>1.2 时,说明交易量变多,要减少共识群组中的节点数量N来快速达成共识.

当 X<0.8 时,说明交易量变少,本文要增加共识群组中的节点数量N来保证更高的安全性,同时可以加速替换共识群组中的异常节点(因为有更多的正确节点,异常节点会显得更离群).

3)协作链节点N_1, N_2, …, N_n将该工作链最新的节点数量要求发送给对应的工作链节点$W_j.N_1$, $W_j.N_2$, …, $W_j.N_n$, 等收取到超过阈值的协作链节点发送的信息后,这些工作链节点更新自己的本地节点列表,如果共识节点增加,将从备选节点中选取

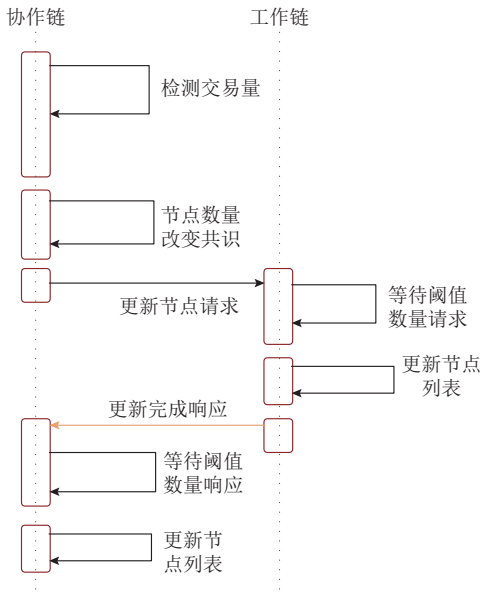


Fig. 7 MC-Schedule node scheduling process

图 7 MC-Schedule 节点调度流程

新的节点$W_j.A_1$, $W_j.A_2$, …, $W_j.A_m$(m 为需要的节点数量)加入到共识群组中；如果节点减少，优先将信誉值低的共识节点$W_j.N_{low1}$, $W_j.N_{low2}$, …, $W_j.N_{lowm}$放入到备选节点中. 最终，由新的共识节点向协作链发送回复.

4)当协作链节点收到超过阈值的回复后会更新自己的节点列表.

3.4 节点信誉评判以及节点自动替换 MC-Scan

在讲述 MC-Scan 算法前，需要注意 2 点：

1)在选举新的 leader 时，会根据最新已经提交的区块来计算最新的信誉值，然后选举最高信誉值的节点作为 leader.

2)仅仅把信誉值最高的节点作为 leader 会产生很多弊端. 例如：当该节点成为信誉值最高的节点后，即使做出异常的行为，也会因为节点信誉值高而霸占多个共识阶段，因此需要一个节点信誉值扣除机制，来保证异常的高信誉值节点能够快速地被剔除. 另外，需要防止节点的信誉值无限制的增长，以至于高信誉值节点和新加入的节点之间信誉值差距过大导致没有成为 leader 的机会.

因此，本文通过此轮共识下节点的 2 种不同表现来计算信誉值：

1)当前共识下产生了新的区块

如图 8 所示，如果当前共识下产生了新的区块，那么一定会存在一个新的 QC，这标志着其祖父区块会被确认提交，并且，根据 BLS-12 门限签名机制，可以获得该祖父区块的签名参与者，因此，在选举下一轮共识的 leader 时，就可以根据最新被提交的区块来计算每个节点的信誉值，为了防止其信誉值无限增长，需要采用特定的方法使一直诚实的签名参与者的信誉累计增加量趋近收敛，信誉值的增加幅度也应当以签名参与者累计成功参与次数、是否被选拔为 leader 等因素综合度量.

为此，节点$W_j.N_i$的第 n 次成功参与投票后的信誉值R_n^i 按照式(2)的方式增加.

$$R_n^i = R_{n-1}^i + B\left(\frac{\sum S^i_{\text{success}}}{\mathrm{e}^{R_{n-1}^i}} + a\frac{\sum R^j}{N}\right), \tag{2}$$

其中R_{n-1}^i 为上一次投票后的信誉值，S^i_{success} 是节点 i 成功参与区块构建投票的次数，$\frac{\sum R^j}{N}$为共识节点的信誉均值，N 是共识节点的个数，其不包括备选节点个数，当共识节点参与了区块的投票时，B=1，否则 B=0，a 是信誉调节因子，当节点$W_j.N_i$被选为 leader 时，a=−0.3，否则 a=0.

信誉调节因子 a 的作用在于提升共识节点群组的整体公平性和积极性，使得诚实节点被当选为 leader 的概率是相近的，同一节点不能连续被选中，从而避免了共识节点群组的中心化问题，即纵使从始至终是诚实节点也不应该一直被选中为 leader.因为 leader 节点为信誉值最高的节点，为保证其信誉值不能无限增长，并且不能长期处于最高信誉的状态，在 leader 每次当选后，就减少其信誉值，将 a 值设为 −0.3 保证了其每次削弱后不会被其他节点判定为异常节点，其信誉值处于正常共识节点信誉值的中游. 非 leader 节点且做出正常行为的节点的 a=0，这保证了其余节点的活性，意味着只要持续做出正常行为，就一定会当选为 leader 节点.

算法 1. MC-Scan 正确区块产生时信誉值的计算.

输入：第 *View* 轮共识；

输出：Null or “*View* 过旧”.

① *block*←*getCommittedBlock*(); /*获得最新提交

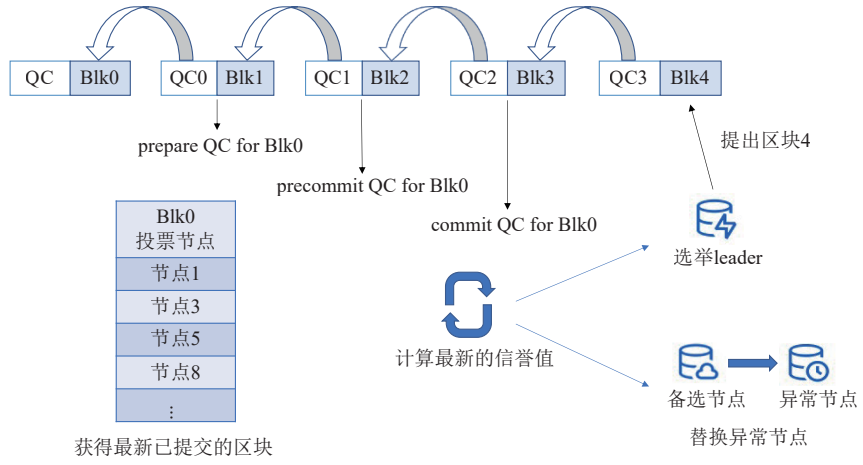


Fig. 8 A new block has been generated under the current consensus

图 8 当前共识下产生了新的区块

```
        的区块*/
② if block.view > View–chainLength()
③     return “View 过旧”;
④ end if
⑤ voters←block.QC().signature().participant;
        /*获得最新提交的区块的参与者*/
⑥ for each voters containing v
⑦    if prevCommitHeadView() < block.view
⑧       if v.id in currentConsensusNode /*这个节点
            还在共识组中*/
⑨          v.prereputation←v.reputation;
⑩          v.reputation←式(2);
⑪          v.timeout←false;
⑫       end if
⑬    end if
⑭ end for
⑮ if prevCommitHead.View() < block.view
⑯    prevCommitHead←block;
⑰ end if
⑱ updateCurrentReplicasReputation(). /*将已经标
        记为超时的节点信誉值继续减半*/
```

2）当前共识下没有产生正确的区块

如图 9 所示，如果当前共识下没有产生正确的区块，将无法通过新的 QC 来寻找其祖父区块来计算新的信誉值. 为了保证信誉值计算能够正常进行，本文将根据与 QC 产生原理相似但意义不同的 TC 作为信誉评判的标准. TC 的产生是因为当前共识下没有对 leader 提出的区块产生正确的响应，从而导致此次共识超时，根据所有节点对本次共识超时达成共识以后而产生的门限签名本文就可以成功地过渡到下一轮共识，进行新一轮共识，TC 的作用相当于在该共识中产生了一个空区块为流水线填充，从而保证流水线不会中断，若因为连续超时等原因导致多个空白区块及 TC 连接在一起，则会形成一个连续的空流水线，但最终提交到区块链中的内容都为经过 3 阶段共识成功的 QC，所以实际的区块链中并不存在 TC. 因此，本文就可以利用 TC 来对此次共识下的 leader 的行为进行评判.

当选举下一轮共识的 leader 时，会检查最新的 TC，如果 TC 是当前第 *View* 轮共识产生的话，则断定此轮共识的 leader 发生了异常. 但是，异常的可能性有很多种，其中一种可能是因为当前的交易量无法满足一个区块的打包，导致 leader 无法提出新区块造成的超时，该行为并不是因为 leader 节点自身的原因造成的，而是该节点不能因为做出了正确的行为而扣除了错误的信誉值. 因此，本文采取一个缓和措施，如式(3)所示，当共识第 1 次产生了 TC，本文不对当前节点采取信誉值扣除，而只是标注其发生了异常行为，其节点还是信誉值最高的节点，但无论其是否存在自身的问题，都不应选举该节点当选下一轮共识的 leader 节点，因此，本文顺延至信誉值排名第 2 的节点来当作新的 leader 发起共识.

当此异常 leader 再次当选 leader 并且做出正确行为时，或者参与下一轮投票时，本文会取消上一次异常行为的标注，但如果该节点还是产生异常行为，本文就会扣除该节点的信誉值，扣除的信誉值为当前所有节点信誉值平均值的一半，这对该节点的信誉积累是致命的，所有节点的平均信誉值代表大部分正常节点的信誉值，即便此时的异常节点为信誉值最高的节点，经过 1~2 轮的异常判定后，其信誉值会被减少到远离平均值，这意味着该节点很快就会被踢出共识组，这保证了节点不能因为信誉值高而不

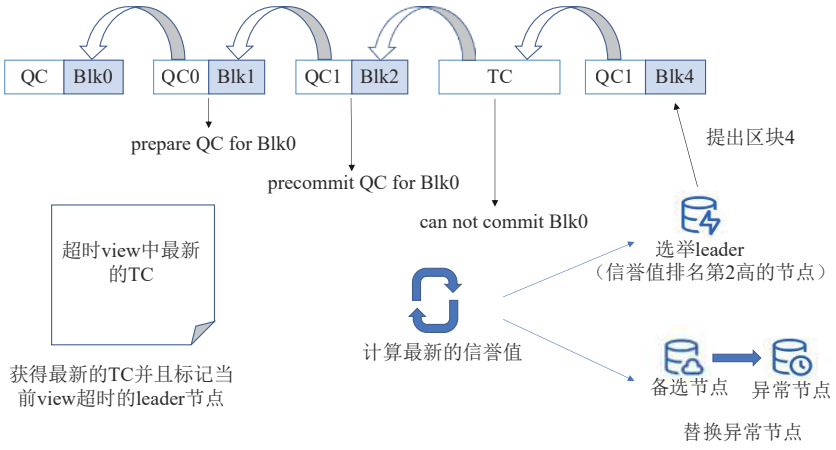


Fig. 9 The correct block has not been generated under the current consensus

图 9 当前共识下没有产生正确的区块

作为，而是等到信誉值降低到一定程度后再恢复正常行为.

$$R_n^i=\begin{cases}R_{n-1}^i, & label=\text{false};\\ R_{n-1}^i-\sum_{N}^{R^j}, & label=\text{true}.\end{cases} \tag{3}$$

算法 2. MC-Scan 没有正确区块产生时信誉值的计算.

输入：第 *View* 轮共识.

① *block*←*getCommittedBlock*(); /*获得最新提交的区块*/

② if *block.view* > *View*−*chainLength*()

③ return “*View* 过旧”;

④ end if

⑤ if *preTimeOutQC*[*View*−1]==false&&
highTCview() ==*View*−1&&*highTCview*()>
highQCview() /*此轮共识还未访问过 TC，并且产生的新的 TC 比 QC 的 view 还高*/

⑥ *leader*←*getLeader*(*View*−1);

⑦ if *leader.timeout*==false /*第 1 次仅标记*/

⑧ *leader.timeout*←true;

⑨ else /*第 2 次扣除信誉值*/

⑪ *leader.preReputation*=*leader.Reputation*;

⑫ *leader.Reputation*= 式(3);

⑬ end if

⑭ *prevTimeOutQC*[*View*−1] = true;

⑮ end if

当本轮共识的节点信誉值更新完成后，本文引入聚类算法 DBSCAN[29-30]，以节点的上一次信誉值作为 x 轴，新的信誉值作为 y 轴，平均信誉值为半径来对异常节点进行筛选，通过扫描，本文将平均信誉值范围内存在的节点数量小于阈值的节点判定为异常节点，随后与备选节点进行替换.

为了保持一致性，替换时要保证从节点 ID 最小的备选节点替换，当备选节点不足时，本文需要暂停替换，直到新的备选节点加入；当备选节点充足但替换失败时，表明备选节点存在异常节点，本文维持当前共识组，并更新备选节点，当更新完成时进行新一轮的替换.

本文使用 BLS12 加密算法来保证当节点被替换或者加入新的节点不会影响门限签名的生成，从而不影响共识的流程，解决方案是对此让每个节点都记录当前共识组节点的公钥，当节点变化时则更新这个节点的记录表.

算法 3. MC-Scan 选举 leader 节点并替换异常节点.

输入：第 *View* 轮共识；

输出：*leader Node*.

① 执行算法 1;

② 执行算法 2; /*执行算法 1 或算法 2 计算节点的信誉值*/

③ *cluster*←*dbscan*(*nodelist*,*quorumSize*(*numReplicas*),*getAverageReputation*()); /*通过 DBSCAN 聚类算法，以节点平均信誉值为半径，2/3 节点为阈值进行聚类，寻找异常节点*/

④ *alterNodes*←*currentConsensusNode*−*cluster*; /*在簇外的节点为异常节点*/

⑤ if *len*(*alterNodes*)≠0

⑥ *alterReplica*← *updateCRepl*(*alterNodes*); /*先从本地替换，返回被替换的节点(有可能会因为备选节点不足或备选节点异常导致没有节点被替换)*/

⑦ end if

⑧ *isChange*←*MC-Syn*(*alterReplicas*); /*算法 4*/

⑨ if *isChange* ==false

⑩ return “节点更新失败”;

⑪ end if

⑫ *leader*←*getHighestReputationReplicas*();

⑬ return *leader*.

MC-RHotStuff 中提出的信誉值计算和异常节点替换机制不仅适用于多链环境中，其核心思想也适用于大多数共识算法和不同类型的区块链，具有很强的通用性.

3.5 节点状态同步

节点调度流程如图 10 所示. 当一轮共识中未发生节点替换时，共识组中的节点和备选节点所保存的每个节点信誉值是一样的，而且它们都会更新自己的区块链，因此，在未替换的情况下，当前进到下一轮共识时，其状态都是同步的.

当一轮共识中发生了视图异常更替时，首先，备选节点由于身份的限制无法参与共识，所以接收不到 TC，因此无法在异常情况下更新节点的最新状态. 简言之，备选节点可以同步区块链的区块信息，而无法同步每个节点的信誉状态. 这时就需要一个同步的过程来保证全局状态的一致. 该过程只能由共识节点发起，因为大多数节点为诚实节点，因此当共识组在本轮共识中发现了需要替换的节点时，由于替换节点首选 ID 最小原则，假设 $W_j.A_1$ 为 ID 最小的备选节点，多数共识节点都会将 $W_j.A_1$ 和异常节点 $W_j.N_i$

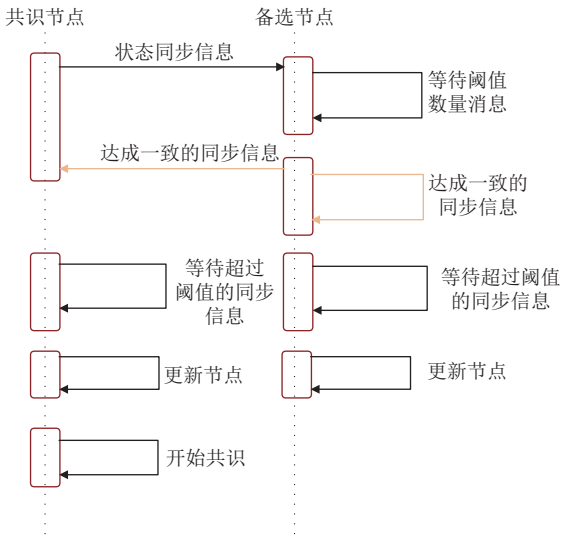


Fig. 10 Node status synchronization process

图 10 节点状态同步流程

进行替换. $W_j.N_1$, $W_j.N_2$, …, $W_j.N_{i-1}$, $W_j.N_{i+1}$, …, $W_j.N_n$会发送节点替换信息. 当所有节点收到超过共识节点阈值的消息后会进行转发; 当收到超过所有节点阈值的消息, 并且这些消息中存在$W_j.A_1$的消息, 则表明大多数节点已经对最新的节点信誉达成了共识, 并且$W_j.A_1$是活跃的节点, 这时所有节点认可$W_j.A_1$是新的共识节点, 并更新自己的列表, 将$W_j.A_1$转变为$W_j.N_{new}$. 当下一轮共识开始时, $W_j.N_{new}$将参与新的共识, 工作链共识节点会将最新同步的节点状态广播给协作链, 使协作链和工作链节点状态一致.

这个过程是在一轮共识中发生的, 由于半同步性质, 大多数的诚实节点都处于本轮共识中, 因此当发生节点替换时, 每个共识节点都延长了本轮共识的时间来保证替换的成功完成, 延长的时间由当前共识组正常推进一轮共识所需要的时间来决定, 由于每个节点都会维护一个最近几轮共识所需要的时间, 通过计算这几轮共识的方差, 并采用随机数算法添加一定的时间后来作为下一轮的共识最长时间. 每个节点的超时时间都是不同的, 但都不会超过最大超时时间, 即 GST.因此, 从理论结果表明, 当发生替换后, 延长的时间一定要超过 GST, 本文设定延长的时间为 GST 的 2 倍. 当替换完成后, 会推进下一轮共识继续进行新的共识, 从宏观角度来看, 节点的替换行为并没有影响共识的进行, 区块链的状态仍保持一致. 从算法 4, 5 和结果来看, 节点替换虽然会延长此轮共识的时间, 影响区块链的整体性能, 但与因节点故障而导致的性能下降相比可忽略不计.

算法 4. MC-Syn 实现节点状态同步.

输入: *alterReplicas*;

输出: ture or false.

① if *len*(*alterReplicas*)≠0/*真的发生了节点替换*/

② *synchronizer.restTimer*();
/*延长本轮共识的超时时间, 以完成节点同步*/

③ *synchronizer.cancelView*();
/*取消本轮共识下的相关工作*/

④ *sendNodeInfoToOtherNodes*(*alterReplicas*);

⑤ until get *quorumSize*(*allNodes*) *replys*;
/*收到了该工作链上多数节点对该消息的响应*/

⑥ if *alterReplicas.hash*==*replys.hash*

⑦ return true;

⑧ else

⑨ return false;

⑩ end if

⑪ end if

⑫ return true.

算法 5. UpdateNodeInfo 接收 MC-Syn 信息更新节点信息.

输入: *alterReplicas*.

① wait *quorumSize*(*consensusNodes*);

② alter *Replicasrequests*; /*更新节点由共识节点发起, 因此到达共识节点阈值就表明更新有效*/

③ update *CurrentReplicas*(*alterReplicasrequset*) ;

④ send *NodeInfoToOtherNodes*(*quorumrequset*).
/*转发给共识节点, 当共识节点收到超过阈值的消息后, 表明该工作链已经完成同步*/

4 性能分析

本文使用 GO 语言来编写了大约 5 000 行代码实现了实验算法, 本文在 4 台 Intel Xeon® Gold 6240C CPU @ 2.60 GHz × 72, 256 GB 运行内存的服务器上进行实验, 本文将该服务器虚拟化成 32 台可视化的 Ubuntu22 虚拟机, 每台虚拟机拥有 8 核 2.60 GHz 的 CPU 和 16 GB 的运行内存, 它们直接的网络通信延迟少于 1 ms, 平均网络带宽 10 000 Mbps.

本文算法主要和原生 HotStuff 算法进行比较, 本文设定一笔交易中拥有 128 B 的负载, 批处理大小为 400, 主要观察其吞吐量和交易延迟的差异. 另外, 本文对节点行为评判机制进行实验来观察本文算法的

表现.

1)实验 1. 该实验在多链环境下进行，将 0.16 MTPS 作为基准交易吞吐量，并布置了 10 个协作链节点，首先，该实验让每个协作链节点达到负载均衡，通过不断增加工作链条数来观察工作链平均吞吐量. 通过实验发现，当工作链条数增加到 9 时，平均吞吐量下降，这说明协作链节点条数达到了性能瓶颈. 另外，该实验改变协作链节点的负载均衡，100% 将交易平均分配给所有协作链节点，1% 使 1 个协作链节点负责处理交易，通过实验发现，当负载均衡率越低时，平均吞吐量将显著下降. 通过实验可以得出，若协作链节点数量较多且性能较好时即可发挥出每条工作链的最大性能，实验结果如图 11 所示.

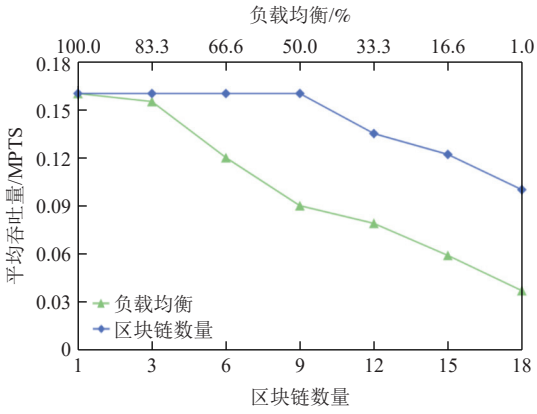


Fig. 11 Average throughput of multiple chains

图 11 多链平均吞吐量

2)实验 2. 该实验在一条工作链上进行，引入节点调度与原生算法进行比较，将 0.16 MTPS 作为基准交易吞吐量，32 个节点作为基准共识节点. 根据实验结果可以看到，在吞吐量小于基准交易吞吐量时，由于 MC-Schedule 会增加共识节点的数量，节点之间通信的开销增大，交易的延迟会小幅度增长，但随着吞吐量的增加，共识节点数量减少，其交易延迟要优于原生算法. 本文认为，为了在吞吐量高的情况下保持良好的性能，在吞吐量较少的情况下牺牲一部分的性能，来加快对异常节点的替换是值得的，因为节点数量越多，异常节点的异常行为暴露得就越快，实验结果如图 12 所示.

3)实验 3. 在该实验中，本文设置了 32 个共识节点，其中，有 10 个异常节点，本文设定每 100 轮共识时产生 1 个异常节点，随后通过 MC-Scan 算法将其替换，本文记录每 100 轮共识的平均交易延迟和吞吐量，并采用不具备节点替换机制的原生 HotStuff 算法与其比较.

可以看到，如图 13 所示在前 200 轮共识中交易时延有小幅度增长，这是因为节点替换导致共识的持续时间变长而造成的，引入节点替换机制后，会在替换初期产生较原生 HotStuff 更高的交易时延. 但在随后的共识中，随着异常节点的数量慢慢减少，时延整体是优于原生算法的. 另外，本文可以看到，随着异常节点的减少，交易吞吐量也慢慢增加，实验结果如图 14 所示.

4)实验 4. 为了检验 MC-Scan 寻找异常节点的精确度，本文设置了 2 个对比实验，在本实验中，为了方便观察，仅设置 4 个共识节点，其中，节点 1 为异常

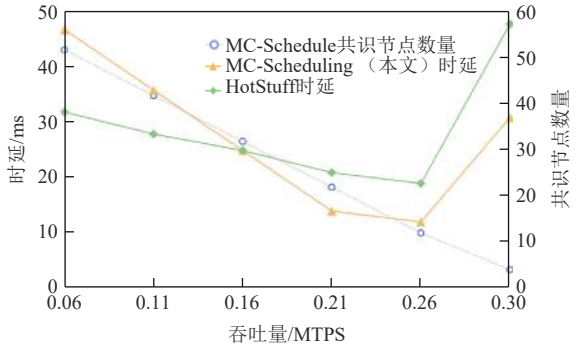


Fig. 12 Comparison between dynamic consensus node and original consensus mode

图 12 动态共识节点与原共识模式对比

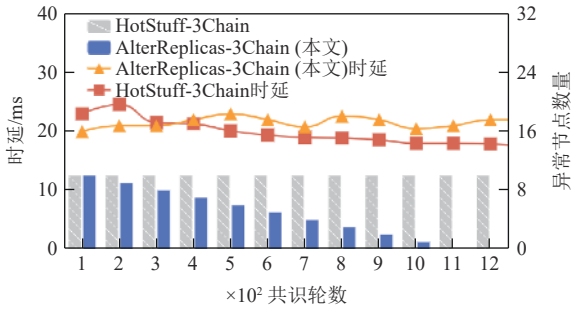


Fig. 13 Comparison of transaction throughput after introducing node replacement mechanism

图 13 引入节点替换机制后交易吞吐量对比

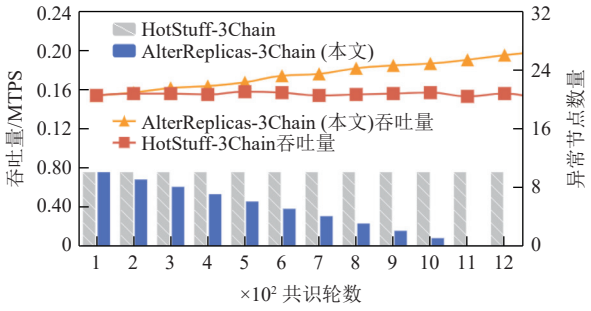
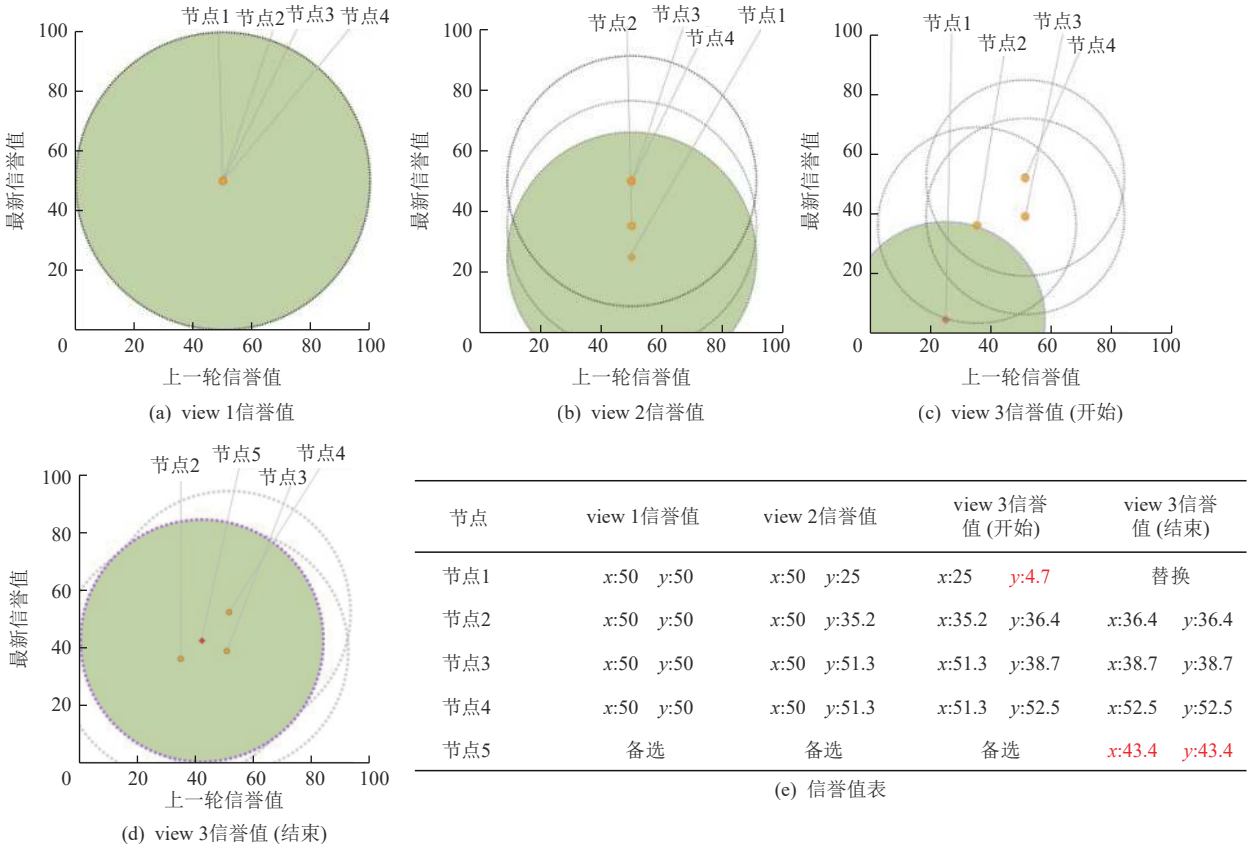


Fig. 14 Comparison of transaction latency after introducing node replacement mechanism

图 14 引入节点替换机制后交易时延对比

节点, 并且持续做出异常行为, 本文设置每个节点的初始信誉值为 50, 图 15 中的虚线部分是所有节点的平均信誉度, 而阴影部分表示节点 1 以平均信誉度为半径的扫描范围, 最新信誉值代表本轮节点信誉值, 上一轮信誉值代表上一轮节点信誉值. 可以看到, 随着共识进程的进行, 节点 1 的信誉值持续下降, 最终在 MC-Scan 算法的扫描下被判定为异常节点, 随后被替换为节点 5, 节点 5 获得其余 3 个节点的平均信誉值, 其信誉值范围为阴影部分. 与其他信誉值评判算法相比, MC-Scan 算法具有自动检测异常节点并进行替换的功能, 并且, 在大规模节点情况下该算法同样有效, 实验结果如图 15 所示.



节点	view 1信誉值	view 2信誉值	view 3信誉值 (开始)	view 3信誉值 (结束)
节点1	*x*:50 *y*:50	*x*:50 *y*:25	*x*:25 *y*:4.7	替换
节点2	*x*:50 *y*:50	*x*:50 *y*:35.2	*x*:35.2 *y*:36.4	*x*:36.4 *y*:36.4
节点3	*x*:50 *y*:50	*x*:50 *y*:51.3	*x*:51.3 *y*:38.7	*x*:38.7 *y*:38.7
节点4	*x*:50 *y*:50	*x*:50 *y*:51.3	*x*:51.3 *y*:52.5	*x*:52.5 *y*:52.5
节点5	备选	备选	备选	*x*:43.4 *y*:43.4

(e) 信誉值表

Fig. 15 Exception node persistent exception

图 15 异常节点持续异常

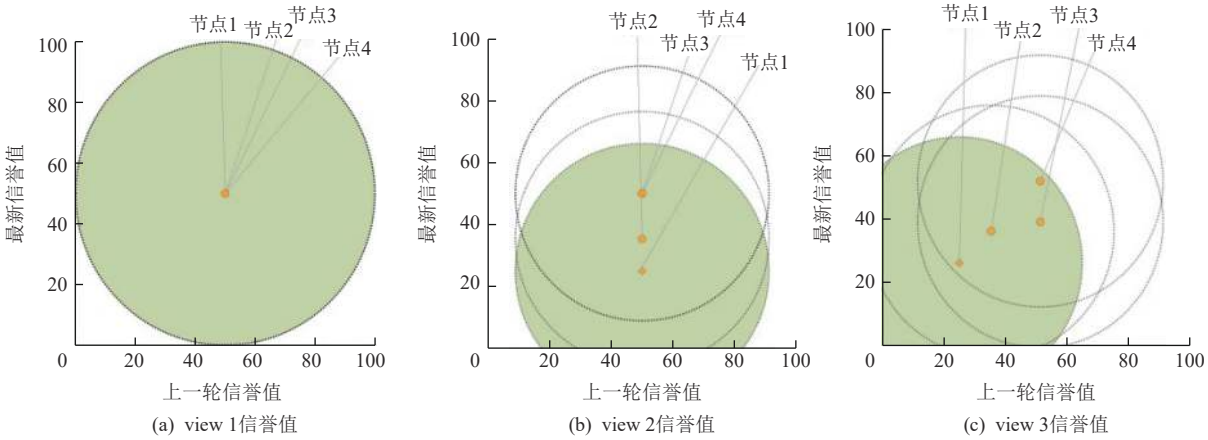
5)实验 5. 在本实验中设置 4 个共识节点, 其中, 节点 1 为异常节点, 在其做出 1 次异常行为后转为正常节点, 本文设置每个节点的初始信誉值为 50, 图中的虚线部分是所有节点的平均信誉度, 而阴影部分表示节点 1 以平均信誉度为半径的扫描范围. 可以看到, 随着共识进程的进行, 节点 1 的信誉值产生了下降, 但随后转为正常节点后, 其信誉值不断上升, 最终在收敛的节点信誉值计算下, 其信誉度逐渐恢复成和正常节点同一个水平, 并不会作为异常节点而被替换, 实验结果如图 16 所示.

5 结 论

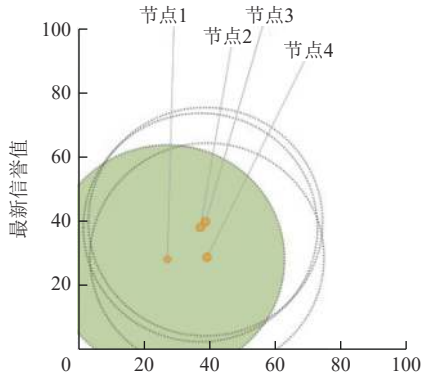
本文提出了一种新型多链联盟链共识机制, 通过引入节点身份机制来为不同节点划分不同职能, 本文使用节点信誉值及基于信誉值的节点聚类和替换算法寻找共识节点中的异常节点并清除来实现更高的系统效率, 同时引入了动态共识节点调整机制, 进一步优化系统资源使性能得到提升.

未来的工作包括 2 个方面: 1)继续优化资源池调度算法来对共识算法进行改进; 2)实现分层机制划分不同的工作链, 使更高层次的工作链作为协作链管理低层次工作链.

作者贡献声明: 路宇轩提出了算法思路和实验方案; 孔兰菊提出指导意见并修改论文; 路宇轩、张宝晨、闵新平负责完成实验并撰写论文.



(a) view 1信誉值　(b) view 2信誉值　(c) view 3信誉值



(d) 最终信誉值

节点	view 1信誉值	view 2信誉值	view 3信誉值	最终信誉值
节点1	*x*:50 *y*:50	*x*:50 *y*:25	*x*:25 *y*:26.2	*x*:26.6 *y*:27.1
节点2	*x*:50 *y*:50	*x*:50 *y*:35.2	*x*:35.2 *y*:36.1	*x*:36.4 *y*:37
节点3	*x*:50 *y*:50	*x*:50 *y*:51.3	*x*:51.3 *y*:39.2	*x*:29.1 *y*:29.5
节点4	*x*:50 *y*:50	*x*:50 *y*:51.3	*x*:51.3 *y*:52.2	*x*:52.6 *y*:38.7
节点5	备选	备选	备选	备选

(e) 信誉值表

Fig. 16　Exception node nonpersistent exception

图 16　异常节点非持续异常

参　考　文　献

[1] Castro M, Liskov B. Practical Byzantine fault tolerance[C]//Proc of the 3rd USENIX Symp on Operating Systems Design and Implementation (OSDI). Berkeley, CA: USENIX Association, 1999: 173−186

[2] Ongaro D, Ousterhout J. In search of an understandable consensus algorithm[C]//Proc of the 2014 USENIX Annual Technical Conf. Berkeley, CA: USENIX Association, 2014: 305−319

[3] Facebook. The diem blockchain[R/OL]. 2020[2022-09-11]. https://www.diem.com/en-us/white-paper/

[4] Lamport L, Shostak R, Pease M. The Byzantine Generals Problem[M]. New York: ACM, 2019

[5] Castro M, Liskov B. Practical Byzantine fault tolerance and proactive recovery[J]. ACM Transactions on Computer Systems, 2002, 20(4): 398−461

[6] Jiang Nianqi, Bai Fenhua, Huang Lin, et al. Reputation-driven dynamic node consensus and reliability sharding model in IoT blockchain[J]. Algorithms, 2022, 15(2): 28−50

[7] Aluko O, Kolonin A. PROOF-OF-REPUTATION: An alternative consensus mechanism for blockchain systems[J]. arXiv preprint, arXiv: 2108.03542, 2021

[8] Huang Baohua, Peng Li, Zhao Weihong, et al. Workload-based randomization Byzantine fault tolerance consensus protocol[J]. High-Confidence Computing, 2022, 2(3): 100070

[9] Hasan O, Brunie L, Bertino E. Privacy-preserving reputation systems based on blockchain and other cryptographic building blocks: A survey[J]. ACM Computing Surveys, 2023, 55(2): 1−37

[10] Jin Hai, Dai Xiaohai, Xiao Jiang. Towards a novel architecture for enabling interoperability amongst multiple blockchains[C]//Proc of the 38th Int Conf on Distributed Computing Systems (ICDCS). Piscataway, NJ: IEEE, 2018: 1203−1211

[11] Luo Kan, Yu Wei, Ling Chaogao, et al. A multiple blockchains architecture on inter-blockchain communication[C]//Proc of the 2018 IEEE Int Conf on Software Quality, Reliability and Security Companion (QRS-C). Piscataway, NJ: IEEE, 2018: 139−145

[12] Agrawal K, Aggarwal M, Tanwar S, et al. An extensive blockchain based applications survey: Tools, frameworks, opportunities, challenges and solutions[J]. IEEE Access, 2022, 10: 116858−116906

[13] Abdo J B, Sibai R E, Demerjian J. Permissionless proof-of-reputation-x: A hybrid reputation-based consensus algorithm for permissionless blockchains[J]. Transactions on Emerging Telecommunications Technologies. 2021, 32(1): e4148

[14] Gueta G G, Abraham I, Grossman S, et al. Sbft: A scalable decentralized trust infrastructure for blockchains[J]. arXiv preprint, arXiv: 1804.01626, 2018

[15] Yin Maofan, Malkhi D, K. Reiter M, et al. HotStuff: BFT consensus with linearity and responsiveness[C]//Proc of the 2019 ACM Symp on Principles of Distributed Computing. New York: ACM, 2019: 347−356

[16] Dwork C, Lynch N, Stockmeyer L. Consensus in the presence of partial synchrony[J]. Journal of the ACM, 1988, 35(2): 288−323

[17] Fischer M J, Lynch N A, Paterson M S. Impossibility of distributed consensus with one faulty process[J]. Journal of the ACM, 1985, 32(2): 374−382

[18] Schneider F. Implementing fault-tolerant services using the state machine approach: A tutorial[J]. ACM Computing Surveys, 1990, 22(4): 299−319

[19] Shoup V. Practical threshold signatures[C]//Proc of the Advances in Cryptology (EUROCRYPT 2000). Berlin: Springer, 2000: 207−220

[20] Ntoanidou S, Polidoros A, Dordas C, et al. Hierarchical clustering methods for binary data from molecular markers[J]. International Journal of Data Analysis Techniques and Strategies, 2020, 12(3): 190−212

[21] Boneh D, Lynn B, Shacham H. Short signatures from the weil pairing[J]. Journal of Cryptology, 2004, 17(4): 297−319

[22] Goyal R, Vaikuntanathan V. Locally verifiable signature and key aggregation[C]//Proc of the 42nd Annual Int Cryptology Conf. Berlin: Springer, 2022: 761−791

[23] Lei Hui, Qiu Chao, Yao Haipeng, et al. When blockchain-enabled Internet of things meets cloud computing[J]. Computer, 2019, 52(12): 16−17

[24] Choi K, Chang B. A theory of RPC calculi for client-server model[J]. Journal of Functional Programming, 2019, 29: e5

[25] Brandão D, S. Rosa N. Multiple transport protocols in an adaptive RPC-based framework[C/OL]//Proc of the 55th Hawaii Int Conf on System Sciences. 2022[2022-12-11]. https://scholarspace.manoa.hawaii.edu/items/b0ddbfc9-22f3-4fa0-97e3-a05d3dc810dd

[26] Wang Yinyin, Yang Yuwang, Qiu Xiulin, et al. CCF-LRU: Hybrid storage cache replacement strategy based on counting[J]. Applied Intelligence, 2022, 52(5): 5144−5158

[27] Liu Dongxiao, Huang Cheng, Ni Jianbing, et al. Blockchain-cloud transparent data marketing: Consortium management and fairness[J]. IEEE Transactions on Computers, 2022, 71(12): 3322−3335

[28] Xu Cheng, Zhang Ce, Xu Jianliang, et al. Slimchain: Scaling blockchain transactions through off-chain storage and parallel processing[J]. Proceedings of the VLDB Endowment, 2021, 14(11): 2314−2326

[29] Cheng Difei, Xu Ruihang, Zhang Bo. Fast density estimation for density-based clustering methods[J]. arXiv preprint, arXiv: 2109.11383, 2021

[30] Hanafi N, Saadatfar H. A fast DBSCAN algorithm for big data based on efficient density calculation[J]. Expert Systems with Applications, 2022, 203: 117501

Lu Yuxuan, born in 1999. Master candidates. Student member of CCF. His main research interests include blockchain consensus and node management.

路宇轩, 1999 年生. 硕士研究生. CCF 学生会员. 主要研究方向为区块链共识、节点治理.

Kong Lanju, born in 1978. PhD, professor, PhD supervisor. Senior member of CCF. Her main research interests include blockchain consensus, multi-chain architecture, and large-scale data management.

孔兰菊, 1978 年生. 博士，教授，博士生导师. CCF 高级会员. 主要研究方向为区块链共识、多链体系结构、大规模数据管理.

Zhang Baochen, born in 1992. PhD candidate. Student member of CCF. Her main research interests include sharding structure of blockchain, verification of blockchain transactions, and blockchain consensus algorithms.

张宝晨, 1992 年生. 博士研究生. CCF 学生会员. 主要研究方向为区块链分片架构、区块链交易验证、区块链共识算法.

Min Xinping, born in 1993. PhD, professor. His main research interests include blockchain consensus and consistency of digital assets.

闵新平, 1993 年生. 博士，教授. 主要研究方向为区块链共识、数字资产的一致性.