计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
Computer Systems & Applications,2024,33(4):179−186 [doi: 10.15888/j.cnki.csa.009455]

E-mail: csa@iscas.ac.cn
http://www.c-s-a.org.cn
Tel: +86-10-62661041

面向物联网的改进 PBFT 共识算法①

叶博文[1,2], 贾小林[1,2], 顾娅军[1,2]

[1](西南科技大学 计算机科学与技术学院 射频识别与物联网实验室, 绵阳 621010)
[2](绵阳市移动物联射频识别技术重点实验室, 绵阳 621010)
通信作者: 贾小林, E-mail: my_jiaxl@163.com

摘　要: 随着物联网的发展, 高效的共识算法是区块链技术应用于物联网的关键. 针对实用拜占庭容错 (practical Byzantine fault tolerance, PBFT) 算法在物联网场景中通信次数多、未考虑共识功耗、共识时延高等问题, 本文提出了一种基于二分 K 均值算法的改进 PBFT 共识算法 (binary K-means practical Byzantine fault tolerance algorithm, BK-PBFT). 首先, 获取节点地理坐标并计算节点综合评价值, 通过二分 K 均值算法将节点划分为一个双层多中心聚类集群. 然后, 先在下层集群再在上层集群对区块进行 PBFT 共识. 最后, 集群验证执行并存储区块, 完成共识. 此外, 本文证明了当节点均匀分布在每个簇时算法通信次数可以达到最少, 以及通信次数最少时的最优聚类数. 分析与仿真结果表明, 本文算法可以有效减少通信次数、降低共识功耗和共识时延.

关键词: 物联网; 区块链; 实用拜占庭容错; 聚类; 综合评价值; 功耗

引用格式: 叶博文,贾小林,顾娅军.面向物联网的改进 PBFT 共识算法.计算机系统应用,2024,33(4):179–186. http://www.c-s-a.org.cn/1003-3254/9455.html

Improved PBFT Consensus Algorithm for Internet of Things

YE Bo-Wen[1,2], JIA Xiao-Lin[1,2], GU Ya-Jun[1,2]

[1](RFID & IoT Laboratory, School of Computer Science and Technology, Southwest University of Science and Technology, Mianyang, 621010, China)
[2](Mobile Internet of Things and Radio Frequency Identification Technology Key Laboratory of Mianyang, Mianyang, 621010, China)

Abstract: With the development of the Internet of Things (IoT), efficient consensus algorithms are the key to applying blockchain technology to the IoT. This study proposes an improved PBFT consensus algorithm based on the binary K-means practical Byzantine fault tolerance algorithm (BK-PBFT) to address the issues of high communication times, lack of consideration for consensus power consumption, and high consensus latency in IoT scenarios. Firstly, it obtains the geographic coordinates of the nodes, calculates the comprehensive evaluation values of the nodes, and divides the nodes into a two-layer multi-center clustering cluster by the binary K-means algorithm. Then, PBFT consensus is performed on the blocks in the lower-level cluster and then in the upper-level cluster. Finally, the cluster validates and stores the blocks to complete the consensus. Additionally, this study proves that the algorithm can achieve the minimum number of communication times when nodes are evenly distributed in each cluster, and obtain the optimal cluster number under the least communication times. The analysis and simulation results show that the proposed algorithm can effectively reduce communication times, consensus power consumption, and consensus latency.

Key words: Internet of Things (IoT); blockchain; practical Byzantine fault tolerance (PBFT); clustering; comprehensive evaluation value; power consumption

① 基金项目: 国家自然科学基金面上项目 (614713606); 四川省自然科学基金 (2022NSFSC0548); 四川省重点研发计划 (2020YFS0360); 四川省教育厅教改项目 (JG2021-1414)
收稿时间: 2023-09-11; 修改时间: 2023-10-09; 采用时间: 2023-11-24; csa 在线出版时间: 2024-01-30
CNKI 网络首发时间: 2024-02-01

随着物联网的迅速发展[1], 其应用场景不断变化, 包括智能家居、智慧城市、工业互联网等领域. 然而, 由于传统的物联网存在中心化的弊端[2], 因此可以借助区块链技术[3,4]解决中心化问题. 然而, 常见的区块链共识算法[5]却并不适用[6]于物联网设备. 因此如何设计一个合适的共识算法使得区块链技术能够应用于物联网以解决物联网的中心化与数据安全问题是本文的研究动机. 相比较于常见的区块链共识算法, 实用拜占庭容错算法[7]不需要大量的算力, 不会产生中心化问题, 并且具有 1/3 总节点数的容错性, 因此可以考虑作为应用于物联网场景的共识算法. 但是随着总节点数的增多, 实用拜占庭容错算法的通信次数会急剧增加.

针对实用拜占庭容错算法性能下降的问题, 已有学者提出了多种改进方案. 文献[8]提出了一种基于 K-medoids 的实用拜占庭容错共识机制. 该方案通过在聚类过程中引入惰性系数, 作为聚类中心节点的更换概率, 从而使得共识节点的聚类过程更加可控. 然而, K-medoids 算法的时间复杂度较高, 尤其在大规模节点场景下, 聚类效率较低. 而在物联网场景中, 文献[9]构建了一个大规模无线密集型网络场景, 并为该场景提出了一种基于节点地理位置特征的聚类算法 C-PBFT. 该算法通过逐层共识的方式, 减少了共识通信的次数. 然而, 该算法在集群内共识时通信复杂度仍然较高, 并且在聚类处理时对簇内节点数量设置并没有做出解释. 为解决上述问题, 本文在基于地理位置特征之上, 还引入了节点评价机制, 从而通过二分 K 均值算法缩减方法共识时的节点规模, 以减少节点间的通信次数和时延. 此外, 本文还分析了此类利用聚类思路改进 PBFT 算法的聚类过程, 得到了通信次数最少时的簇内节点数量和聚类数 K 的最优值, 从而消除了二分 K 均值算法中 K 值的随机选取对聚类效果的影响.

鉴于物联网场景具有节点密集、资源受限等特征, 对于实用拜占庭容错算法的改进, 还需要考虑算法的共识功耗. 本文对共识过程中的功耗进行了分析, 得到了功耗最低时的最优聚类数. 首先, 本文介绍了相关的背景知识, 然后描述了本文 BK-PBFT 算法的流程. 最后, 分析和仿真实验结果表明, BK-PBFT 算法在减少通信次数、降低共识功耗以及共识时延方面具有显著的效果.

1 相关知识

1.1 实用拜占庭容错算法

在实用拜占庭容错算法 (practical Byzantine fault tolerance, PBFT) 中, 节点被分为两类: 主节点和从节点. 在任何时刻, 只有 1 个主节点, 而其他节点则为从节点.

如图 1 所示, 实用拜占庭容错算法主要包含 3 个阶段: 预准备阶段、准备阶段以及提交阶段.

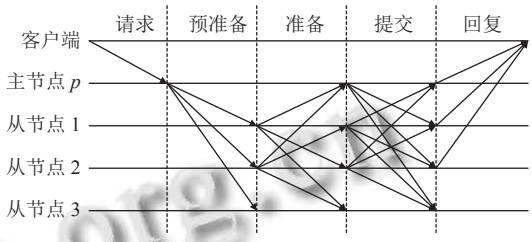

图 1 PBFT 一致性协议流程

预准备阶段: 一旦接收到客户端发送的$<REQUEST, o,t,c,\sigma_p>$消息, 主节点则进入预备阶段. 其中 o 代表客户端请求操作, t 代表消息发送的时间戳, c 代表客户端的 ID, 而σ_p则表示主节点对请求消息摘要的签名. 首先, 主节点为接收到的请求分配一个序号 n. 然后, 主节点向所有从节点广播$<<PRE\text{-}PREPARE, v,n,d,\sigma_p>, m>$消息, 其中 v 表示当前的视图编号, n 是刚刚分配的序号, m 是客户端请求消息, d 则为 m 的哈希值.

准备阶段: 在收到$<PRE\text{-}PREPARE>$消息并验证通过后, 从节点进入准备阶段. 从节点会广播$<PREPARE, v,n,d,i,\sigma_i>$消息, 其中 i 表示节点的 ID. 同时, 从节点也会接收到其他从节点广播的$<PREPARE>$消息.

提交阶段: 在接收到$<PREPARE>$消息后, 如果节点的$prepare(m,v,n,i)$状态为 true, 则进入提交阶段. 节点广播$<COMMIT, v,n,d,i,\sigma_i>$消息. 同时, 节点还会接收到其他节点广播的$<COMMIT>$消息. 根据收到的$<COMMIT>$消息, 如果节点的$commitied\text{-}local(m, v,n,d,i)$状态为 true, 则节点执行客户端发送的操作并将结果返回给客户端. 在同一请求下, 如果客户端接收到 f+1 个相同的结果, f 为恶意节点数量, 表示节点之间达成共识, 客户端将这个结果视为最终结果.

1.2 区块链结构

区块链发端于一篇题为“Bitcoin: A peer-to-peer electronic cash system”[10]的论文. 维基百科对区块链的定义是: 区块链是一种分布式账本, 其中包含不断增长的区块, 这些区块通过哈希值进行安全链接. 如图 2 所示, 一个区块由区块头和区块体两部分组成. 区块头包含了前一个区块的哈希值、时间戳、版本号、随机

数、Merkle 根以及目标哈希. 区块体则包含了所有已验证的交易记录. 这些记录经过 Merkle 树的哈希过程生成唯一的 Merkle 根, 然后被记录在区块头中.

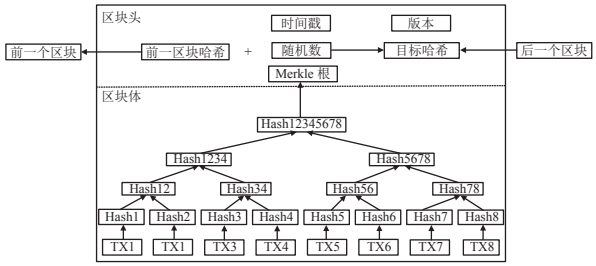

图 2 区块链结构

1.3 物联网场景

本文基于工业和信息化部提出的物联网区块链场景[11], 如图 3 所示, 其中包括终端用户设备、物联网服务器、物联网网关、全功能物联网设备等, 它们作为全节点存在于基础设施之上, 通过点对点的去中心化通信机制相互协作. 在这个场景中, 全节点指的是那些拥有良好的计算、存储和通信能力的节点, 它们需要存储完整的区块链数据并积极参与共识过程. 一些计算能力受限的物联网设备可以通过连接物联网网关来加入物联网区块链网络. 此外, 物理世界或信息世界的实体也可以通过绑定设备的方式映射到物联网区块链中. 为了简化实验与场景, 本文将全节点绑定到计算机端口, 通过计算机端口间的通信来模拟全节点之间的通信, 同时暂不考虑那些计算能力受限的物联网设备和物理/虚拟对象. 共识通过的区块会保存在节点内部. 通过二分 K 均值算法, 全节点有可能成为主节点或从节点, 从而参与到共识过程中.

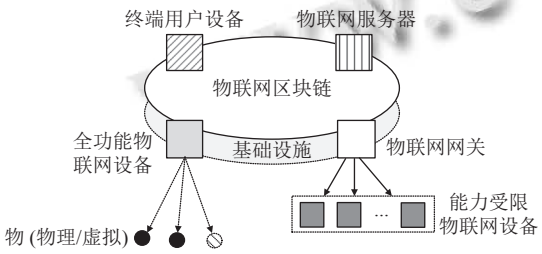

图 3 物联网区块链场景

2 BK-PBFT 算法

根据上述物联网区块链场景, 本文提出了基于二分 K 均值算法的改进 PBFT 算法, 二分 K 均值算法在节点数量较多时, 可以有效降低节点聚类的时间复杂度, 并且可以克服 K 均值算法收敛于局部最小问题. BK-PBFT 算法流程如图 4 所示.

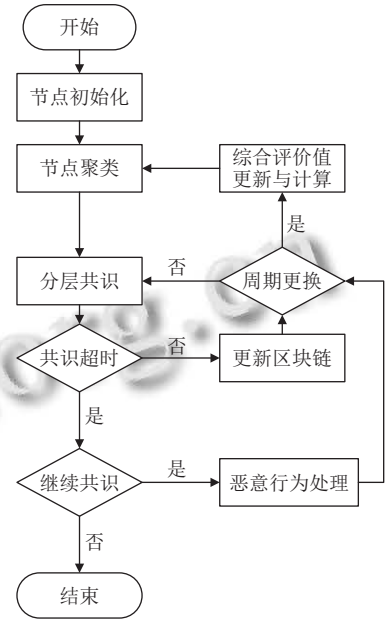

图 4 BK-PBFT 共识流程

BK-PBFT 算法的主要流程可以分为 4 个步骤, 即节点初始化、节点聚类处理、分层共识、评价值更新和恶意行为处理.

2.1 节点初始化

首先, 为每个节点在公钥基础设施 (PKI) 体系中完成密钥注册. 接着, 利用卫星导航系统或蜂窝网络等定位技术获取节点的地理坐标. 然后, 初始化节点的综合评价值, 部分节点可以初始化有较高综合评价值, 而其他节点可以采取随机赋值的方式初始化综合评价值. 最后, 节点广播 $<BROADCAST,l,e,i,\sigma_i>$ 消息, 其中包含节点的地理坐标 l、节点综合评价值 e、节点 ID 号 i, 以及消息摘要的签名 σ_i.

2.2 节点聚类处理

根据收到的节点地理坐标和节点综合评价值, 节点使用二分 K 均值算法对共识节点进行聚类处理, 形成一个双层多中心的节点簇.

具体而言, 二分 K 均值算法的核心思想是选择误差平方和最大的簇, 然后反复执行 K 均值算法, 直到达到预定的簇数量. 然而, 在聚类过程中: 首先, 由于 PBFT 算法至少需要 4 个节点才能运行, 因此在聚类时需要控制每个簇的节点数量, 每个簇的最优节点数量由第 3 节分析得出. 其次, 二分 K 均值算法使用计算的质心

来代表中心节点, 因此中心节点可能并不存在. 最后, 中心节点既要参与从节点集群的共识, 又要参与主节点集群共识, 因此, 中心节点的诚实性尤为关键. 在选择节点的过程中, 不仅要考虑节点的位置特征, 还要考虑节点的综合评价值以确保节点的诚实性. 算法 1 对 K 均值算法进行了改进, 并将 K 值设为 2, 然后应用于二分 K 均值算法.

算法 1. 改进 K 均值算法

输入: 节点集$D=\{x_1, x_2, \cdots, x_n\}$, 聚类数 2, 最大迭代次数 M;
输出: 簇列$C=\{C_1, C_2\}$.

1) 从节点集 D 中选出综合评价值前二的节点$\{\mu_1, \mu_2\}$作为初始中心节点
2) 对于 $m=1, 2, \cdots, M$
a) 簇列 C 初始化为$C_t=\varnothing, t=1, 2$
b) 对于 $i=1, 2, \cdots, n$, 计算节点x_i和中心节点$\mu_j(j=1,2)$的欧几里得距离$d_{ij}=\|x_i-\mu_j\|_2^2$, 如果x_i综合评价值过低, 则忽略计算此节点. 然后将x_i标记为最小的d_{ij}对应的类别 j, 如果C_j节点数量大于簇的最优节点数量, 则x_i标记为次小的d_{ij}对应的类别 j. 此时, 更新$C_j=C_j\cup\{x_i\}$
c) 对于 $j=1, 2$, 重新计算C_j的中心节点, 为综合评价值较高且离质心较近的节点
d) 如果中心节点发生变化, 转步骤 2)
3) 输出簇列$C=\{C_1, C_2\}$

二分 K 均值算法 (见算法 2) 使用 SSE 作为衡量聚类质量的目标函数. SSE 值越大, 表示簇的聚类效果越差. SSE 的定义如式 (1) 所示, 其中 x 代表簇中的一个节点, c_i代表簇的中心节点.

$$SSE=\sum_{x\in C_i} dist(c_i-x)^2 \tag{1}$$

算法 2. 二分 K 均值算法

输入: 节点集$D=\{x_1, x_2, \cdots, x_n\}$, 聚类数 K;
输出: 簇列$C=\{C_1, C_2, \cdots, C_k\}$.

1) 使用改进 K 均值算法将节点集划分为两个子簇, 将子簇加入簇列
2) 根据式 (1) 计算簇列中每个簇的 SSE
3) 选择 SSE 最大的簇用 K 均值算法划分为两个子簇, 将子簇加入簇列, SSE 最大的簇出列, 如果此时簇列中簇的数量小于 K, 转步骤 2)
4) 输出簇列$C=\{C_1, C_2, \cdots, C_k\}$

如图 5 所示, 经过聚类后, 节点被分为若干个簇, 簇的中心节点即为簇的主节点, 所有的簇的主节点构成一个上层的主节点集群, 其余节点构成了下层从节点集群.

2.3 分层共识

划分好集群后便开始分层共识, 分层共识流程主要分为下层从节点集群共识, 上层主节点集群共识, 区块验证执行及上链阶段, 如图 6 所示.

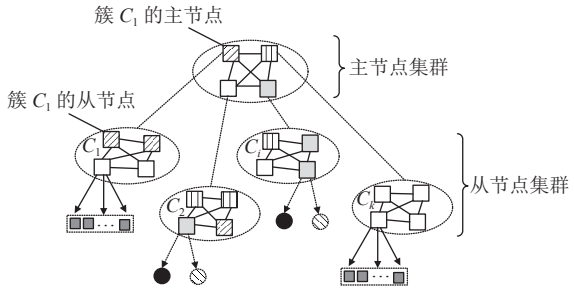

图 5 双层多中心节点集群

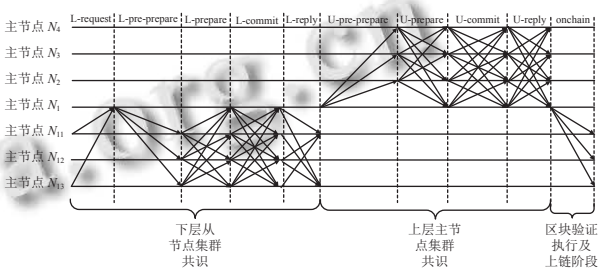

图 6 分层共识流程

1) 从节点集群共识

当簇 1 的从节点N_{11}和N_{13}要发送请求消息时, 它们对应 PBFT 算法中的客户端, 将请求发送到簇的主节点, 主节点将单位时间内收到的请求打包成区块, 然后将区块发送给从节点, 开启下层从节点集群的 PBFT 共识. 当主节点收到至少$\lfloor(n_c-1)/3\rfloor+1$条相同的节点答复后, 区块通过从节点集群共识, n_c指的是主节点所在簇的节点数量.

2) 主节点集群共识

在从节点集群完成共识后, 区块将在主节点集群开启 PBFT 共识. 主节点会按照接收区块的时间顺序, 在主节点集群广播已通过从节点集群共识验证的区块. 在 U-reply 阶段, 主节点需要广播包含对区块签名的答复信息, 当主节点收到至少$\lfloor(n_k-1)/3\rfloor+1$条相同的答复信息后, 主节点会执行区块内的请求, 将区块存储到本区块链上, 随后主节点将这些答复信息连同区块一起发送给所属簇的从节点.

3) 区块验证执行及上链阶段

从节点在收到上层主节点发送的答复信息和区块后, 会先验证答复信息中签名的正确性, 然后验证区块内请求消息签名的正确性, 然后执行区块内的请求, 最终将区块存储到本地区块链上.

2.4 评价值更新

在若干轮的共识内, 节点共识表现不一. 虽然一些节点表现出色, 但可能仍然无法成为主节点. 因此, 通

过设定评价指标, 在若干轮的共识内不断更新指标, 重新计算每个节点的综合评价值, 周期性地更替主节点. 之后具有较高综合评价值的节点将被选为主节点, 以参与上层共识, 而综合评价值过低的节点则会被拒绝参与共识.

1) 确立评价指标

首先把评价指标分为 3 类: 节点表现、节点能力和节点诚实度.

2) 计算指标权重

算法采用客观赋权法, 即 CRITIC 方法[12]计算评价指标的权重. 根据表 1, 构建一个包含多个指标的指标集合$X=\{x_1,x_2,x_3,x_4,x_5\}$. 其中, 可用性$x_1$表示节点的正常运行时间与总运行时间的比值, 共识次数x_2表示节点自加入物联网区块链以来成功完成共识的次数, 平均响应时间x_3表示节点从接收请求到返回结果的平均时间, 吞吐量x_4表示节点每秒处理的交易数量, 恶意行为次数x_5包括广播无效的请求和区块等行为.

表 1 节点评价指标表

类别	评价指标
节点表现	可用性共识次数
节点能力	平均响应时间吞吐量
节点诚实度	恶意行为次数

节点正向指标包括可用性、共识次数和吞吐量, 节点负向指标包括平均响应时间和恶意行为次数. 对于 n 个节点的数据, 首先使用式 (2) 对指标数据进行归一化处理.

$$x_{ij}=\begin{cases}\dfrac{X_{ij}-\min(X_j)}{\max(X_j)-\min(X_j)}, & j=1,\ 2,\ 4\\ \dfrac{\max(X_j)-X_{ij}}{\max(X_j)-\min(X_j)}, & j=3,\ 5\end{cases} \tag{2}$$

其中, X_{ij}为第i个节点的第 j 个评价值, $\min(X_j)$为所有节点第 j 个评价指标数据的最小值, $\max(X_j)$为所有节点第 j 个评价指标数据的最大值. 接下来, 通过式 (3) 和式 (4) 计算指标的变异性和指标的冲突性.

$$\begin{cases}\bar{x}_j=\dfrac{1}{n}\displaystyle\sum_{i=1}^{n}x_{ij}\\ S_j=\sqrt{\dfrac{\displaystyle\sum_{i=1}^{n}(x_{ij}-\bar{x}_j)^2}{n-1}}\end{cases} \tag{3}$$

指标的变异性通常用标准差来表示, 标准差越大, 表明该指标的数值差异越广, 因此能够提供更多的信息, 该指标本身的评价力度也越强, 因此应该分配更高的权重. 其中, n 表示节点数量, S_j表示第 j 个指标的标准差.

$$R_j=\sum_{i=1}^{p}(1-r_{ij}) \tag{4}$$

指标的冲突性通过相关系数来度量, 如果一个指标与其他指标的相关性较高, 那么该指标与其他指标的冲突性就较低, 这表示它传达了相似的信息, 因此在评价中可能会存在重复的内容, 因此需要减少该指标的权重分配. 在这里使用 p 表示评价指标的数量, r_{ij}表示指标 i 和指标 j 之间的相关系数. 然后, 使用式 (5) 计算指标的信息量.

$$C_j=S_j\sum_{i=1}^{p}(1-r_{ij})=S_j\times R_j \tag{5}$$

C_j越大, 第 j 个评价指标在整个评价指标体系中的作用越大, 就应该给其分配更多的权重.

最后使用式 (6) 计算第 j 个指标的客观权重.

$$W_j=\frac{C_j}{\displaystyle\sum_{j=1}^{p}C_j} \tag{6}$$

3) 计算节点综合评价值

算法使用逼近理想解排序法[13]计算节点综合评价值. 首先, 对于 n 个节点, p 个评价指标的数据利用式 (2) 做正向化处理, 再利用式 (7) 构造标准化矩阵Z_{ij}.

$$Z_{ij}=\frac{x_{ij}}{\sqrt{\displaystyle\sum_{i=1}^{n}x_{ij}^2}} \tag{7}$$

然后利用式 (8), 式 (9) 计算各评价指标与最优及最劣向量之间的差距D_i^+和D_i^-.

$$D_i^+=\sqrt{\sum_{j=1}^{p}W_j(Z_j^+-z_{ij})^2} \tag{8}$$

$$D_i^-=\sqrt{\sum_{j=1}^{p}W_j(Z_j^--z_{ij})^2} \tag{9}$$

其中, W_j为第 j 个指标的客观权重, Z_j^+, Z_j^-为标准化矩阵中第 j 列的最大值和最小值, z_{ij}为标准化矩阵中节点 i 第 j 个指标. 最后, 式 (10) 计算节点与最优方案的接近程度, 即节点的综合评价值.

$$C_i = \frac{D_i^-}{D_i^+ + D_i^-} \tag{10}$$

2.5 恶意行为分析及处理

恶意行为分为下层共识的恶意行为和上层共识的恶意行为.

在下层共识中, 恶意行为有 3 种: 第一, 主节点不响应从节点的请求, 不将请求消息打包成区块并发送给其他从节点; 第二, 主节点给簇内从节点发送不同的预准备消息, 使得从节点收到少于$2\times\lfloor(n_c-1)/3\rfloor+1$条从节点广播的相同的准备消息; 第三, 由于网络等原因, 节点收到少于$2\times\lfloor(n_c-1)/3\rfloor+1$条节点广播的相同的确认消息. 由于下层共识就是缩减了节点规模的 PBFT 共识, 所以 3 种恶意行为的处理方法和 PBFT 算法类似. 但是在流程上, 在更换了主节点后, 新的主节点需要打包从节点的确认消息, 将确认消息广播到主节点集群共识以得到主节点集群多数节点的认可.

在上层共识中, 发送区块的主节点对应 PBFT 共识中的主节点, 其余主节点则对应从节点, 那么上层共识中恶意行为有 4 种: 第一, 主节点给从节点发送不同的区块消息, 使得从节点收到少于$2\times\lfloor(n_k-1)/3\rfloor+1$条从节点广播的相同的准备消息; 第二, 由于网络等原因, 节点收到少于$2\times\lfloor(n_k-1)/3\rfloor+1$条节点广播的相同的确认消息; 第三, 在 U-reply 阶段, 主节点收到了少于$\lfloor(n_k-1)/3\rfloor+1$条相同的答复消息; 第四, 主节点向簇内从节点发送错误的区块或答复消息. 前 3 种恶意行为的处理方法仍和 PBFT 算法类似, 第 4 种恶意行为会导致簇内从节点验证答复信息或区块中的签名失败, 处理方法就是簇内从节点会发起 PBFT 算法中的视图转换以更换当前簇的主节点, 新的主节点需要打包从节点的确认消息, 将确认消息广播到主节点集群共识以得到主节点集群多数节点的认可.

3 性能分析与仿真

一轮共识的通信次数为 k 个簇广播的区块的共识通信次数之和. 一轮共识的功耗为处理 k 个簇广播的区块的功耗之和. 下文对一轮共识的通信次数和功耗分别分析. 为了更好地表述, 下文使用的与分析相关的符号如表 2 所示.

3.1 通信次数分析

首先计算一个簇广播的区块的共识通信次数d_i, 其等于区块在从节点集群共识通信次数与在主节点集群共识通信次数之和.

表 2 通信与功耗分析符号

符号	定义
r	单位字节处理功耗
n	共识节点数量
k	聚类数量
a	一条交易大小
b	区块头大小
c_i	簇i从节点数量
d_i	簇i广播的区块的共识通信次数
p	共识功耗

从节点集群共识包括 L-pre-prepare 阶段、L-prepare 阶段、L-commit 阶段和 L-reply 阶段. 分析图 6 的共识流程可知, 每个阶段的通信次数分别为c_i、c_i^2、$c_i(c_i+1)$以及c_i. 所以区块在从节点集群内共识通信次数为以上各阶段通信次数之和, 即$2c_i^2+3c_i$.

与从节点集群共识分析类似, 区块在主节点集群内通信次数为$3k(k-1)+kc_i$.

因此, 一个簇广播的区块的共识通信次数d_i为$2c_i^2+(3+k)c_i+3k(k-1)$.

实际情况下, 簇的从节点的数量不必相同, 而文献[9]和文献[11]中在聚类时每个簇的从节点数量默认设置为 n/k–1, 且未给出理由. 下文通过计算证明出$c_i=n/k-1$是一轮共识的通信次数最少的必要条件.

一轮共识的通信次数如式 (11) 所示:

$$\sum_{i=1}^{k} 2c_i^2+(3+k)c_i+3k(k-1) \tag{11}$$

又已知从节点的数量之和为 n–k, 即$\sum_{i=1}^{k} c_i=n-k$, 所以式 (11) 可改写为式 (12).

$$3k^2(k-1)+(3+k)(n-k)+2\sum_{i=1}^{k} c_i^2 \tag{12}$$

又由柯西不等式 (13):

$$\left(\sum_{i=1}^{n} x_iy_i\right)^2 \leqslant \left(\sum_{i=1}^{n} x_i^2\right)\left(\sum_{i=1}^{n} y_i^2\right) \tag{13}$$

可知式 (12) 第 3 项最小值如式 (14) 所示:

$$\sum_{i=1}^{k} c_i^2=\frac{1}{k}\sum_{i=1}^{k} 1^2\sum_{i=1}^{k} c_i^2 \geqslant \frac{1}{k}\left(\sum_{i=1}^{k} 1\times c\right)^2=\frac{(n-k)^2}{k} \tag{14}$$

当且仅当$c_1=c_2=\cdots=c_k=n/k-1$时, 式 (12) 的最

小值为 k 的函数 $f(k)$, 如式 (15) 所示:

$$f(k)=3k^2(k-1)+(3+k)(n-k)+\frac{2(n-k)^2}{k} \tag{15}$$

由于主节点集群和从节点集群至少需要 4 个节点, 因此函数自变量 k 的取值范围为$[4,n/5]$.

函数 $f(k)$ 在自变量 k 为$[4,n/5]$时的最小值, 就是一轮共识的通信次数的最小值. 对函数求导得, $f(k)=9k^4-8k^3+(n-1)k^2-2n^2$. 由于导函数为一元四次方程, 根据解方程的常规方法较为复杂, 本文采用 Go 语言数值算法库 goNum 求解. 当共识节点的数量 n 设为 100 时, 解得在聚类数量 k 为 7 时, 共识的通信次数取得最小值为 4 283 次.

在文献[11] C-PBFT 算法中, 一个簇广播的区块的共识通信次数$2n(n/k-1)+2k(k-1)$, 因此一轮共识的通信次数为$2n(n-k)+2k^2(k-1)$. 因此, C-PBFT 算法的通信复杂度为$O(n^2)$.

本文 BK-PBFT 算法的一轮共识的通信次数由上文分析可知为$3k^2(k-1)+(3+k)(n-k)+\frac{2(n-k)^2}{k}$, 因此 BK-PBFT 算法的通信复杂度为$O(n^2/k)$. 因此, BK-PBFT 算法的通信复杂度小于 C-PBFT 算法.

3.2 功耗分析

假设节点单位字节处理功耗相同, 仅考虑聚类数对共识功耗的影响. 一轮共识功耗 p 如式 (16) 所示, 即处理 k 个簇广播的区块的功耗之和. 而处理一个簇广播的区块的功耗为共识通信次数乘以区块大小乘以单位字节处理功耗.

$$p=\sum_{i=1}^{k}d_i\times(c_i\times a+b)\times r \tag{16}$$

通过第 3.1 节的计算得知, 当每个簇的从节点数量为 n/k–1 时, 此时一轮共识的通信次数才会取得最小值. 此时功耗 p 可以表示为函数 $g(k)$, 如式 (17) 所示:

$$\begin{cases} g(k)=kr[2c_1^2+(3+k)c_1+3k(k-1)](ac_1+b) \\ c_1=n/k-1 \end{cases} \tag{17}$$

其中, 常数项 a, b, r 根据实际需求设定. 本文设定交易大小 a 为 1 000 字节, 区块头大小 b 为 48 000 字节, 单位字节处理功耗 r 设为 10.

函数 $g(k)$ 在自变量 k 为$[4,n/5]$时的最小值, 就是一轮共识的最低功耗. 当共识节点的数量 n 设为 100 时, 解得在聚类数量 k 为 7 时, 共识功耗取得最小值为 2.6×10^9.

需要注意的是, 通信次数最少时, 区块内的消息数量增多, 导致共识功耗增加. 而共识功耗最小时, 区块内的消息数量减少, 导致通信次数增加. 如图 7 所示, 随着共识节点数量的增加, 取得最少通信次数的最优聚类数与取得最低功耗的最优聚类数之间的差距逐渐扩大. 因此, 当系统对功耗要求高时, 设置功耗最低时的聚类数, 当系统对时延要求高时, 设置通信次数最少时的聚类数.

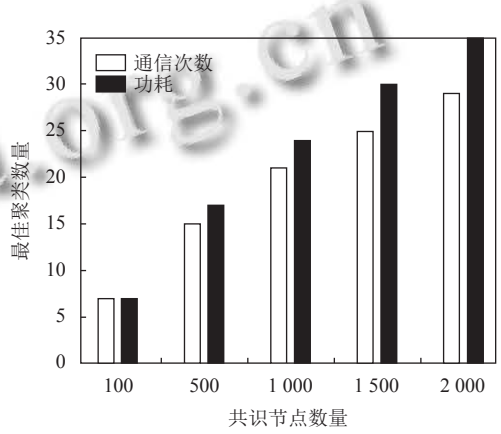

图 7 不同数量节点通信次数与功耗的最佳聚类数量

3.3 时延对比

为了评估 BK-PBFT 算法的时延性能, 在 Visual Studio Code 开发环境下, 采用 Golang 编程语言, 分别模拟文献[8]中的 PBFT 算法、文献[11]中的 C-PBFT 算法以及 BK-PBFT 算法. 共识时延定义为从从节点向主节点发送请求到从节点收到主节点返回的区块的时间间隔. 在节点数量分别为 50、60、70、80、90 和 100 的情况下进行了时延测试. 为了减小误差, 进行 10 次实验并取均值作为最终结果.

在节点数量分别为 50、60、70、80、90 和 100 时, BK-PBFT 算法在取得最少通信次数时的聚类数量分别为 5、6、6、6、7、7. 因此 C-PBFT 算法的聚类数量也设置为 5、6、6、6、7、7. 如图 8 所示, 当节点数量较少时, BK-PBFT 算法和 C-PBFT 算法的延时差距不大, 但随着节点数量的增加, 由于 BK-PBFT 算法只需在从节点集群和主节点集群中各执行一次 PBFT 共识, 因此它的时延表现更佳.

4 结论与展望

本文针对物联网环境下 PBFT 共识算法存在的效率低和未考虑共识功耗的问题, 提出了一种基于二分

K 均值算法的改进 PBFT 算法. 该算法基于节点的地理坐标和综合评价值进行了聚类, 以兼顾共识效率和安全. 算法还对共识过程中的通信和功耗问题进行了分析, 得到了通信次数最少和功耗最低时的最优聚类数量. 分析与仿真实验结果表明, BK-PBFT 算法在通信次数、共识功耗以及共识时延等方面都显著优于 PBFT 和 C-PBFT 算法. 然而, 由于本算法简化了物联网区块链场景, 因此还存在一些改进的空间, 可以考虑引入容器以进一步完善实验场景.

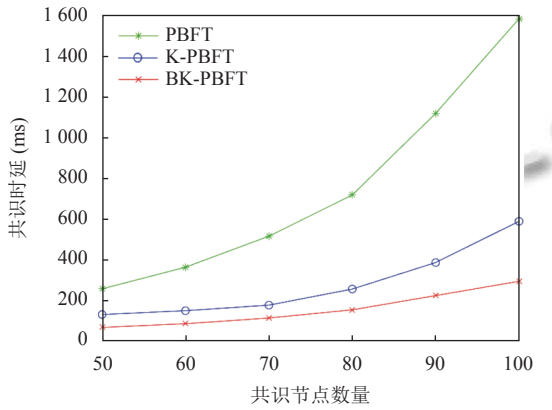

图 8 算法共识时延对比

参考文献

1 余文科, 程媛, 李芳, 等. 物联网技术发展分析与建议. 物联网学报, 2020, 4(4): 105–109. [doi: 10.11959/j.issn.2096-3750.2020.00195]

2 Verma R, Dhanda N, Nagar V. Security concerns in IoT systems and its blockchain solutions. Proceedings of the 2021 CIIR on Cyber Intelligence and Information Retrieval. Singapore: Springer, 2022. 485–495.

3 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. [doi: 10.16383/j.aas.2016.c160158]

4 Ruoti S, Kaiser B, Yerukhimovich A, *et al*. Blockchain technology: What is it good for? Communications of the ACM, 2019, 63(1): 46–53.

5 袁勇, 倪晓春, 曾帅, 等. 区块链共识算法的发展现状与展望. 自动化学报, 2018, 44(11): 2011–2022. [doi: 10.16383/j.aas.2018.c180268]

6 Salimitari M, Chatterjee M, Fallah YP. A survey on consensus methods in blockchain for resource-constrained IoT networks. Internet of Things, 2020, 11: 100212. [doi: 10.1016/j.iot.2020.100212]

7 Castro M, Liskov B. Practical Byzantine fault tolerance and proactive recovery. ACM Transactions on Computer Systems, 2002, 20(4): 398–461. [doi: 10.1145/571637.571640]

8 陈子豪, 李强. 基于 K-medoids 的改进 PBFT 共识机制. 计算机科学, 2019, 46(12): 101–107. [doi: 10.11896/jsjkx.181002014]

9 刘炜, 阮敏捷, 佘维, 等. 面向物联网的 PBFT 优化共识算法. 计算机科学, 2021, 48(11): 151–158. [doi: 10.11896/jsjkx.210500038]

10 Nakamoto S. Bitcoin: A peer-to-peer electronic cash system. https://bitcoin.org/bitcoin.pdf. (2018-04-10).

11 中华人民共和国工业和信息化部. YD/T 3905-2021 基于区块链技术的去中心化物联网业务平台框架. 北京: 人民邮电出版社, 2021.

12 Diakoulaki D, Mavrotas G, Papayannakis L. Determining objective weights in multiple criteria problems: The critic method. Computers & Operations Research, 1995, 22(7): 763–770.

13 Shih HS, Shyur HJ, Lee ES. An extension of TOPSIS for group decision making. Mathematical and Computer Modelling, 2007, 45(7–8): 801–813. [doi: 10.1016/j.mcm.2006.03.023]

(校对责编: 孙君艳)