计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
Computer Systems & Applications,2023,32(5):323−329 [doi: 10.15888/j.cnki.csa.009079]

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

基于对应比较图的 Fabric 排序机制优化①

刘润德[1,2], 陈志德[1,2]

[1](福建师范大学 计算机与网络空间安全学院, 福州 350117)
[2](福建师范大学 福建省网络安全与密码技术重点实验室, 福州 350007)
通信作者: 陈志德, E-mail: zhidechen@fjnu.edu.cn

摘　要: 针对 HLF (Hyperledger Fabric) 区块链系统在排序阶段中存在的缺陷, 提出了一种基于对应比较图的图排序优化方案. 利用对应比较图具有相关不变性质的图合并过程以及其算法运行时间短的特点, 设计了一种基于交易重要度的拓扑算法, 旨在减少由于默认的顺序排序而导致的序列化冲突问题. 通过实验与分析, 表明该方案有效解决了原始方案的序列化冲突问题, 减少了系统中无效事务的比例, 提升了系统交易效率, 节省了大量的计算与存储资源.

关键词: Hyperledger Fabric; 对应比较图; 交易重要度; 拓扑排序

引用格式: 刘润德,陈志德.基于对应比较图的 Fabric 排序机制优化.计算机系统应用,2023,32(5):323–329. http://www.c-s-a.org.cn/1003-3254/9079.html

Fabric Sorting Mechanism Optimization Based on Corresponding Comparison Graph

LIU Run-De[1,2], CHEN Zhi-De[1,2]

[1](College of Computer and Cyber Security, Fujian Normal University, Fuzhou 350117, China)
[2](Fujian Provincial Key Lab of Network Security and Cryptology, Fujian Normal University, Fuzhou 350007, China)

Abstract: In view of the defects of Hyperledger Fabric in the sorting stage, an optimization scheme of graph sorting based on the corresponding comparison graph is proposed. As the corresponding comparison has a graph merging process with correlation invariance and a short algorithm running time, a topological algorithm based on transaction importance is designed to reduce the serialization conflict caused by the default sequence sorting. The experiments and analysis show that this scheme effectively solves the serialization conflict problem of the original scheme, reduces the proportion of invalid transactions in the system, improves the transaction efficiency of the system, and saves a lot of computing and storage resources.

Key words: Hyperledger Fabric (HLF); corresponding comparison graph; transaction importance; topological sorting

近些年来, 由于区块链技术的去中心化、数据不可篡改和去信任化等特性, 区块链技术被广泛运用于金融领域. 其中, 由 Linux 基金会主导发起并由 IBM 公司开发的一个模块化、可拓展的开源系统 Hyperledger Fabric (HLF) 从众多区块链系统中脱颖而出.

虽然 HLF 较比特币、以太坊在交易效率方面已有了很大的突破, 它被认为是最适合作为企业级应用程序开发的区块链系统[1], 但与传统金融每秒交易的处理量相比, 其效率仍不尽人意, 几乎无法直接运用于现代金融领域[2]. 又因为区块链系统本身的不可篡改和去中心化的特性, 一旦区块链系统产生交易错误, 其损失难以控制, 金融系统交易受阻, 会造成不可弥补的损失.

HLF 区块链系统引入了一种新的“执行—排序—验证” (execute-order-validate, EOV) 的体系架构, 这种架构允许交易执行和验证的并行化[3], 在一定程度上解决了现有区块链系统的不确定性、资源消耗大和易被

① 基金项目: 国家自然科学基金 (62277010, 61841701); 福建省自然科学基金 (2020J01171, 2021J011013)
收稿时间: 2022-09-28; 修改时间: 2022-10-27; 采用时间: 2022-12-10; csa 在线出版时间: 2023-02-28
CNKI 网络首发时间: 2023-03-01

性能攻击等问题. 但是在实际交易过程中, HLF 系统中的排序节点在处理事务交易时并不会检查事务语义, 而是直接按照事务到达的先后顺序来进行排序处理的. 同时, 由于 HLF 系统中一个区块内可能包含上千笔交易, 所以 HLF 系统在事务交易进行至排序阶段时, 极易产生序列化冲突问题. 这种序列化冲突也称“块内冲突”, 严重影响了交易效率, 占用了大量的计算资源和存储资源来计算和存储一些未交易成功的无效事务.

首先, Baliga 等人[4] 通过调整链码参数、增加通道和增加对等节点的数量优化 HLF 系统性能. Gorenflo 等人[5] 通过减少交易期间的计算与 I/O 消耗提升 HLF 系统吞吐量. 上述方案均在系统层面对 HLF 进行优化, 其结果并不尽人意. Xu 等人[6] 设计了一种基于 Redis 数据库的“共享锁”机制来对抗并发冲突问题. 许沁琪[7] 利用缓存交易写集的方式在执行阶段检测交易是否冲突, 以减小冲突交易在 HLF 系统的资源消耗. 上述方案对 HLF 系统执行阶段的冲突问题提出了优化改进, 并未关注到排序服务存在的问题.

Sharma 等人[8] 发现了 HLF 系统的排序服务存在着很大的逻辑缺陷, 在优化了 HLF 系统的排序服务, 使其具有语义检查能力后, 对交易事务进行重新排序, 以此减少序列化冲突问题的产生. Sousa 等人[9] 使用拜占庭容错 (BFT) 优化改进了 HLF 系统的排序服务. Sun 等人[10] 认为文献 [8] 的重排序算法存在可信问题, 故提出了一种基于贪心算法的可信重排序算法.

上述优化方案均在其排序阶段对交易事务进行了预处理, 删除了某些容易引发序列化冲突的交易事务后再对这批交易进行排序处理. 考虑到现实金融交易过程中, 若某客户机发起的交易被重复“判定”为易冲突交易, 那么客户机可能需要发起多次交易提案才能交易成功, 这显然不符合实际交易场景.

因此, 需要一种对大量交易进行全局排序的排序优化方案. 将这些交易按重要程度进行连线即为本文方案的核心思想. 复杂网络理论中最重要的模型 Barabási-Albert 模型[11], 就是建立在重要节点优先连边的基础上的, 但困难的是找出哪些节点才是重要节点[12]. 李劲等人[13] 提出了一种描述节点间不相似性的距离度量方法. Li 等人[14] 提出了一种节点的多元度量方法. Pal[15] 提出了一种基于完全图的排序方法. Behera[16] 在文献 [15] 的基础上提出了一种名为“对应比较图”的用于排序的完全图模型.

本文基于对应比较图的排序方案, 针对 HLF 系统中交易事务进行至排序阶段时极易产生的序列化冲突问题.

(1) 定义了交易事务重要度的概念, 将一个区块内的全部交易事务进行重要性度量.

(2) 根据这些交易事务及其重要度构建对应比较图.

(3) 在生成的对应比较图上运行深度优先搜索算法 (DFS), 进行拓扑排序.

本文方案是对客户机发起的一批次交易事务进行全局排序, 缓解了客户机需要重复发起同一交易提案的问题. 并且通过实验分析表明, 本文优化方案有效减少了系统中的无效事务, 提升了系统的交易效率.

1 相关知识

1.1 Hyperledger Fabric

HLF 是一个许可的区块链系统, 它只追踪分类账数据结构的执行历史, 并未内置加密货币, 其所有节点都需要在加入系统之前进行身份验证, 这些节点的身份由证书颁发机构 (CA) 和成员服务提供商 (MSP) 提供和维护. HLF 区块链系统中的节点可大致为两类: (1) 执行链码、发起提案和验证提案的对等节点. (2) 对交易提案进行排序并生成区块的排序节点.

HLF 舍弃了大多数区块链系统采用的“排序—执行”的体系设计, 引入了一种“执行—排序—验证”的区块链体系结构, 如图 1 所示.

(1) 执行阶段. 系统进行交易的模拟、背书、创建读写集和收集背书结果. 在此阶段, 首先, 由客户机将已签名的交易提案发送给背书节点背书, 背书节点根据交易提案中的参数调用链码生成读写集. 其次, 背书节点对生成的读写集进行签名并连同背书声明一齐返回给客户机. 最后, 客户机接收到返回的背书结果, 并且会从多个背书节点收集足够多的背书结果用于验证, 若验证成功, 则向排序服务广播交易提案.

(2) 排序阶段. 排序服务在接收到的交易事务之间进行全局排序, 创建交易块并传递给网络中的所有对等节点进行验证. HLF 不会检查交易内容, 它只会在满足某些要求时将一批交易按顺序打包到一个块中, 例如接收一定数量的交易事务或交易事务达到时限.

(3) 验证阶段. 所有对等节点根据背书策略和签名验证接收到的块内的事务, 即 VSCC (validation system chaincode) 验证, 若验证通过, 该事务则为一个有效事务, 对于一个有效交易事务, 对等节点将会使用其写入集, 更新当前状态数据库, 然后处理下一个交易事务. 若未验证通过满足, 该事务将被标记为无效事务.

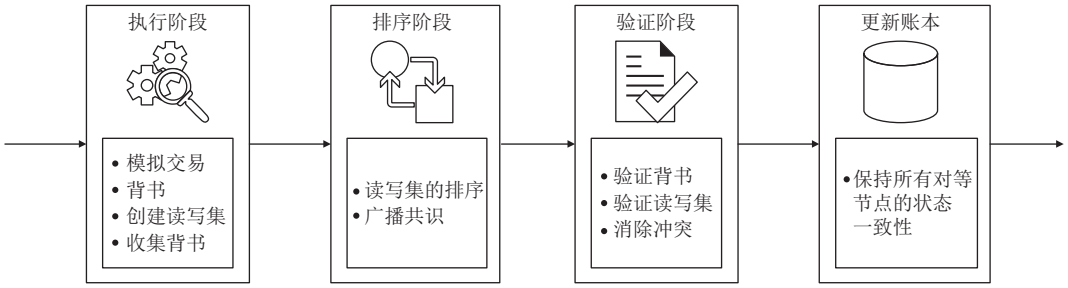

图 1 Hyperledger Fabric 体系结构

1.2 序列化冲突

Hyperledger Fabric 通过 EOV 模型来实现链码的高并发执行, 但同时也会产生大量的不可序列化的交易事务, 这些事务会在验证阶段被对等节点验证失败并被判定为无效事务, 但这些无效事务依旧会被打包成区块上传至链上, 严重影响了系统的交易效率及成功交易事务的吞吐量. 并且这些无效事务通常占比很大, 计算并存储这些无效的交易占用了大量的计算资源与存储资源.

如表 1 所示, 这些交易事务被判定为无效的主要原因是其记录读取集变量的账本状态在读取集生成到账本更新的期间发生了变化.

表 1 读写集变化导致验证失败的示例

Transaction	Read-set	Write-set	Is it verified?
Tx_1	—	$(k_1, v_1 \to v_2)$	√
Tx_2	$(k_1, v_1), (k_2, v_1)$	$(k_2, v_1 \to v_2)$	×
Tx_3	$(k_1, v_1), (k_3, v_1)$	$(k_3, v_1 \to v_2)$	×
Tx_4	$(k_1, v_1), (k_4, v_1)$	$(k_4, v_1 \to v_2)$	×

在 HLF 系统中, 排序服务以黑盒的方式处理交易事务且不以任何方式检查事务语义, 所以排序服务会按照事务到达的先后顺序对事务进行排序. 表 1 中的一系列交易就是由于交易顺序的不恰当, 在第 2 笔交易事务还未进行验证前, 第 1 笔交易事务 Tx_1 的版本号已经由 v_1 更新为了 v_2, 但随后的 3 笔事务在验证读集时仍在读取 v_1, 交易事务的读写集不一致导致后续的一系列事务无法成功验证, 从而产生了一系列的无效交易事务. 这就是 HLF 的序列化冲突, 由于这种交易冲突是发生在同一个区块内部, 所以也称“区块内冲突”.

2 方案设计

本文方案使用的主要符号及其含义如表 2.

2.1 交易事务重要度定义

在对 HLF 系统交易重要程度定义中, 本文方案将结合相关性与多样性的概念, 在相关性与多样性之间取得平衡, 共同定义本文方案中交易事务的重要性概念.

表 2 主要符号及其含义

符号	含义
$A, B, \cdots$	矩阵$A, B, \cdots$
A^{T}	矩阵A的转置
$a, b, \cdots$	向量$a, b, \cdots$
α	阻尼因子
$I, J, \cdots$	集合$I, J, \cdots$

首先给出相关性度量的定义, 个性化 PageRank 是图数据上实现相关性查询的有效方法[17]. 令 G 为一个包含 n 个节点的图, 节点的个性化 PageRank 排序向量可由如下迭代公式表示:

$$r = aA^{\mathrm{T}}r + (1-\alpha)q \tag{1}$$

其中, α $(0<\alpha<1)$ 是阻尼因子, q 是一个$n\times 1$查询向量 $(q(i)\geqslant 0, \sum_{i=1}^{n} q(i)=1)$, A 是一个行规范化的图邻接矩阵, 即 $\sum_{j=1}^{n} A(i,j)=1, n=1,2,\cdots,n$, r 是一个$n\times 1$排序向量, p 如下所示是一个$1\times n$结果向量:

$$p = qB_{1\times n} \tag{2}$$

矩阵 B 为一个$1\times n$的所有元素都为 1 的矩阵, 为简化重要度的定义引入矩阵 C, C 可以视为查询向量 q 的个性化邻接矩阵, 矩阵 C 的定义如下:

$$C = \alpha A^{\mathrm{T}} + (1-\alpha)p \tag{3}$$

给定一个排序列表 S $(|S|=k)$, 本文方案提出的交易重要度定义如下:

$$F(S) = 2\sum_{i\in S} r(i) - \sum_{i,j\in S} C(i,j)r(j) \tag{4}$$

若将矩阵 A 的转置代入式 (4) 中得:

$$F(S) = 2\sum_{i\in S} r(i) - \alpha \sum_{i,j\in S} A(j,i)r(j) - (1-\alpha)\sum_{j\in S} r(j)\sum_{i\in S} q(i) \tag{5}$$

式 (5) 与式 (4) 为等价关系.

因此, 计算 HLF 系统交易重要度的问题就可定义为:

$$S' \leftarrow \arg\max_{|S|=k} F(S) \tag{6}$$

其中, S'为排序列表 S 的子集. 由于 HLF 中交易事务的特性, 即后来的交易事务的键值不会小于当前交易事务的键值, 故可以将这些交易事务连线得到一张用于度量重要度的图. 再通过个性化 PageRank 方法和图的多样性度量可以很容易计算出节点的重要度值.

2.2 对应比较图的构建

使用拓扑排序算法对 HLF 排序机制优化的基础就是完全图的构建. 对应比较图是一个完全图, 它就是将每个包含了交易事务索引和重要度值的元素表示为一个顶点, 将每次重要度值的大小比较结果表示为一条弧, 从重要度小的节点指向重要度大的节点, 由此构造出一个存储了所有比较结果的图.

如图 2 所示, 图中顶点的元素信息为 (交易事务索引, 重要度值), 所以这个弧不仅比较了重要度值, 还表示了交易顺序.

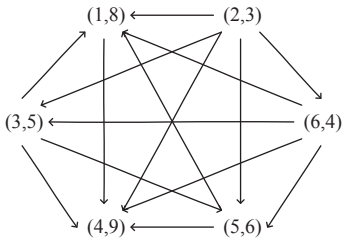

图 2 对应比较图

设 G 为欲构造的图, S'是含有对应图 G 的值的数组, 欲构造的弧为 (u, v), 邻接列表 $adj[]$ 是由一组具有一般双链接列表和数组功能的双链接列表组成的. 算法 1 为弧添加算法.

算法 1. 弧添加算法

```
Function addARC(G, S', u, v)
    w = adj[u].front
    while adj[u].index > 0 do
        if S'[v] < S'[w] then
            INSERT(adj[u], w.index, v)
            G.size++
            return
        if adj[u].index = adj[u].length then
            APPEND(adj[u], v)
            G.size++
        else w = w.next
```

算法 1 中弧添加的基本思想就是不断地从一个重要度值最小的顶点遍历到下一个重要度值最小的顶点, 即从对应比较图中遍历到重要度值最小的顶点 (2, 3), 再遍历到 (6, 4). 并对邻接列表正向或反向的排序实现弧的添加. 新顶点插入的基本思想是从第 1 个元素到最后一个元素遍历, 然后在邻接列表中的正确位置插入新顶点以实现排序的总体不变性.

接着, 由第 2.1 节可以得知S'是一个由 k 个元素组成的数组, 所以, 为了构造对应比较图, 就只需比较两个顶点的重要度大小, 并添加弧. 算法 2 为构造对应比较图的算法.

算法 2. 图构造算法

```
Function constructGRAPH(S', k, r)
    G = null graph of order k
    for i =1 to k do
        for t = 1 to r do
            j = (i + t) mod k
            if S'[i] < S'[j] then
                addARC(G, S', i, j)
            else
                addARC(G, S', j, i)
    return G
```

算法 2 中, G 是一个 k 阶空图, r 为节点与其他节点在图中的比较范围, 例如当 r 等于 1 时, 节点仅与其左右相邻节点进行比较并添加弧. 这时就得到了一个包含 HLF 交易重要度信息的对应比较图.

2.3 基于对应比较图的拓扑排序算法

对应比较图包含如下独特的性质.

(1) 如果 G 是一个对应比较图, 则 G 是无环图, 因此, G 是一个有向无环图.

(2) 如果 G 是一个对应比较图, 则 G 有一个拓扑排序.

(3) 如果 G 是一个对应比较图, 在 G 上运行深度优先搜索算法 (DFS) 生成一个堆栈, 并按完成时间递减排序, 那么这个堆栈就是 G 的拓扑排序.

基于上述对应比较图的特性, 将第 2.2 节中包含 HLF 交易重要度信息的对应比较图输入拓扑排序算法 (算法 3).

算法 3. 拓扑排序算法

```
Function GraphSort(G)
    x = max.index(S')
    S* = {x}
    DFS(G, S*)
    return array(S', S*)
```

拓扑排序算法的主要思想就是从数组 S'的最大值元素开始在图 G 上运行深度优先算法. 其中, 变量 x 存储了线性时间内运行的 S'的最大元素的索引, 引入深度优先搜索算法, DFS(G, S*) 是运行深度优先搜索算法的函数, 其中, S*是深度优先搜索算法生成的堆栈, 即 S*是深度优先搜索算法完成后图 G 的拓扑排序. 最后返回值 array(S', S*) 是根据索引列表 S*从排序列表 S'中构造的一个数组.

这个数组将一个区块内的交易事务的索引按重要度从大到小排列, 排序节点将按照索引对这一批交易事务进行交易排序. 以图 2 为例, 按拓扑排序算法 (算法 3) 在图上运行深度优先搜索, 即可获得一条如图 3 所示的哈密顿路径.

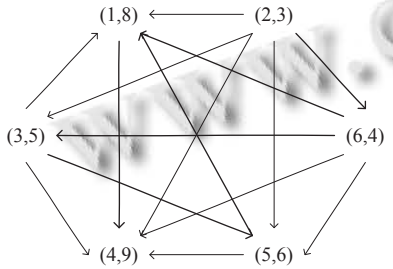

图 3 哈密顿路径

因为存在一条哈密顿路径, 在运行深度优先搜索时只需要进行 5 条边和 6 个顶点的遍历, 那么 DFS 的堆栈即为拓扑排序数组{3, 4, 5, 6, 8, 9}, 所以排序后的交易事务索引数组为{2, 6, 3, 5, 1, 4}, 如此, 就完成了依据重要度值的对应比较图拓扑排序, 排序服务就会按照交易事务索引对块内的一批交易事务进行排序.

3 方案分析

上述优化方案为 HLF 系统的交易事务引入了一种重要度定义, 通过将多样性与相关性这两项指标相平衡从而更准确的定义交易事务的重要度属性. 然后将重要度值与交易事务的索引构造成对应比较图 G, 再对 G 进行拓扑排序.

因此, 本节将从重要度定义的可靠性证明以及对优化前后的 HLF 系统的吞吐量、交易时间和有效事务占比等性能指标进行实验测试, 分析本文优化方案的可行性.

3.1 方案可靠性分析

本节将对本文第 2.1 节的目标函数, 即式 (4) 作方案可靠性分析及证明. 目标函数由两项相减得到本文方案定义的重要度排序序列. 前一项为 2 倍的排序分数和, 用于度量相关性. 后一项为是任意两个节点i, j $(i, j \in S)$在排序列表 S 的距离度量和, 用于度量多样性. 因此, 本文第 2.1 节中所提出的目标函数同时度量了相关性与多样性, 即每个节点的相关性越高 ($r(i)$ 越高), 重要度量 $F(S)$ 越高. 因此, 要证明本文方案重要度定义的可靠性, 首先, 需要满足空排序列表的重要度量 $F(S)$ 总是为 0; 其次, 如果不断地向排序列表中添加节点, 需保证排序列表的整体效用不会降低; 最后, 如果排序列表已经有很多的节点了, 需保证新添加节点的边际效用相对较小.

本文方案的可靠性证明, 即下述 3 个条件的证明.

(1) $F(\emptyset) = 0$.

(2) $F(S)$ 是单调非递减的, 即$F(J) \geqslant F(I)$.

(3) $F(S)$ 是 S 的一个子模. 即:

$$F(I \cup R) - F(I) \geqslant F(J \cup R) - F(J)$$

其中, $\emptyset$为一个空集, I、J、R 为 3 个集合, $I \subseteq J$, $R \cap J = \emptyset$. 目标函数显然满足$F(\emptyset) = 0$.

$F(J) \geqslant F(I)$的证明如下.

令$T = J \cap I$, 将目标函数代入$F(J) - F(I)$得:

$$\begin{aligned} &F(J) - F(I) \\ &= 2\sum_{i \in T} r(i) - \sum_{i \in I}\sum_{j \in J} C(i,j) r(j) - \sum_{i \in T}\sum_{j \in J} C(i,j) r(j) \\ &= \left(\sum_{j \in T} r(j) - \sum_{j \in T}\sum_{i \in I} C(i,j) r(j)\right) \\ &\quad + \left(\sum_{i \in T} r(i) - \sum_{i \in T}\sum_{j \in J} C(i,j) r(j)\right) \end{aligned} \tag{7}$$

又因为矩阵 C 是一个列随机矩阵, 即其每列的和都为 1, 且 r 中的每个元素都是非负的, 故式 (7) 的前半部分有:

$$\begin{aligned} \left(\sum_{j \in T} r(j) - \sum_{j \in T}\sum_{i \in I} C(i,j) r(j)\right) &= \sum_{j \in T} r(j)\left(1 - \sum_{i \in I} C(i,j)\right) \\ &= \sum_{j \in T} r(j) \sum_{i \notin I} C(i,j) \geqslant 0 \end{aligned} \tag{8}$$

同理, 式 (7) 的后半部分有:

$$\begin{aligned} \left(\sum_{i \in T} r(i) - \sum_{i \in T}\sum_{j \in J} C(i,j) r(j)\right) &= \sum_{i \in T}\left(r(i) - \sum_{j \in J} C(i,j) r(j)\right) \\ &= \sum_{i \in T}\sum_{j \notin J} C(i,j) r(j) \geqslant 0 \end{aligned} \tag{9}$$

将式 (8) 和式 (9) 代入式 (7) 中得:

$$F(J)-F(I)\geqslant 0 \tag{10}$$

即$F(J)\geqslant F(I)$, 证毕.

$F(I\cup R)-F(I)\geqslant F(J\cup R)-F(J)$证明如下.

令$T=J\cap I$, 将式 (4) 目标函数代入$(F(I\cup R)-F(I))-(F(J\cup R)-F(J))$中得:

$$\begin{aligned}&(F(I\cup R)-F(I))-(F(J\cup R)-F(J))\\&=\left(\sum_{i\in J}\sum_{i\in R}C(i,j)r(j)-\sum_{i\in I}\sum_{j\in R}C(i,j)r(j)\right)\\&\quad+\left(\sum_{i\in R}\sum_{j\in J\cup R}C(i,j)r(j)-\sum_{i\in R}\sum_{j\in I\cup R}C(i,j)r(j)\right)\\&=\sum_{j\in R}\sum_{i\in T}C(i,j)r(j)+\sum_{i\in R}\sum_{j\in T}C(i,j)r(j)\geqslant 0\end{aligned} \tag{11}$$

即$F(I\cup R)-F(I)\geqslant F(J\cup R)-F(J)$, 证毕.

3.2 方案性能分析

本节将通过实验测试分析对比原始方案与本文优化方案. 本文的方案优化是为了适应现阶段金融交易的效率需要且同时缓解客户机在现实交易中需频繁发起交易提案的问题, 所以本文方案的性能分析重点主要为吞吐量及块内交易有效事务比.

为了分析本文方案的交易处理性能, 本文优化了 HLF 系统的排序机制, 并对交易流程进行了模拟实验, 具体实验环境及实验配置见表 3.

表 3 实验环境及配置

名称	配置/版本
操作系统	CentOS 8.4 64位
CPU核数	8核
运行内存	16 GB
Golang	V1.14.3
Hyperledger Caliper	V0.5.0
Hyperledger Fabric	V2.0.0

本文使用 Hyperledger Caliper 基准测试工具进行以下实验测试. 本文采用了 IBM 公司官方提供的专门用于 HLF 区块链模拟性能测试的实验案例, 此 HLF 网络中有一个名为 mychannel 的通道, 由 org1、org2、org3 这 3 个组织共同负责管理维护, 此案例网络由 2 个对等节点和 1 个排序节点组成, 在此基础上, 本文实验方案设置了 100 个客户机节点进行模拟交易以测试本方案性能, 并通过 Golang 语言实现本方案的链码以控制交易事务排序机制.

首先, 本文对优化方案与原始方案的有效事务与总交易事务数量比进行测试对比, 设置每秒发起的交易事务数量 (transaction per second, TPS) 为 1 100, 交易持续时间 (txDuration) 参数为 30, 测试连续 30 s 内每秒完成的交易总量、有效事务数量和无效事务数量, 并重复 10 次取均值, 得到的实验结果如图 4、图 5 所示, 每秒均为一组实验数据, 共 30 组.

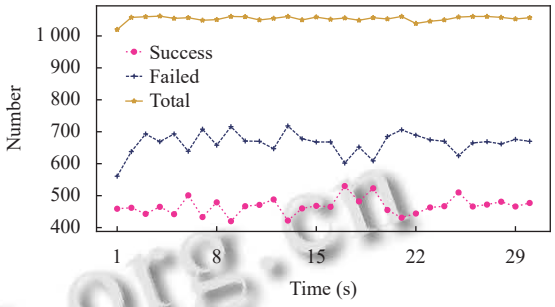

图 4 优化方案有效事务比

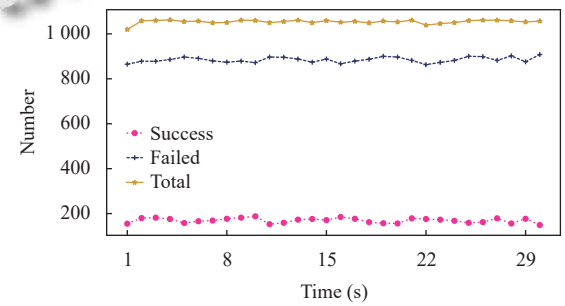

图 5 原始方案有效事务比

可以看出本文优化后的方案中有效事务的数量大都处于 450 至 500 之间, 30 s 持续期间内提交的有效事务数量的平均值为 467, 由于交易期间会产生通道阻塞和交易并发等问题, 实际完成的交易总量平均值为 1 050, 约占交易总量的 44.48%. 原始方案中有效事务的数量均处于 150 左右, 30 s 持续期间内提交的有效事务数量的平均值为 169, 实际完成的交易总量平均值为 1 054, 约占交易总量的 16.03%. 本文优化方案较原始方案有效事务占比提升约 28.45%, 说明本文方案的交易效率较原始方案确有提升, 有效地解决了 HLF 系统中排序阶段易产生的序列化冲突问题.

其次, 本文对系统本身的吞吐量性能进行对比测试, 对比优化方案与原始方案在控制交易事务发起数量的情况下系统的吞吐量性能. 对比结果如图 6 所示.

图 6 中, 横坐标表示每秒发起的交易数量, 设置每秒发起的交易事务数量为 10、20、40、60、80、100. 可以看出, 优化方案较原始方案在事务发起数量逐渐增加后, 系统的吞吐量性能由略微提升, 说明本优化方案在未影响原始方案的基本性能的同时, 吞吐量性能还有所提升.

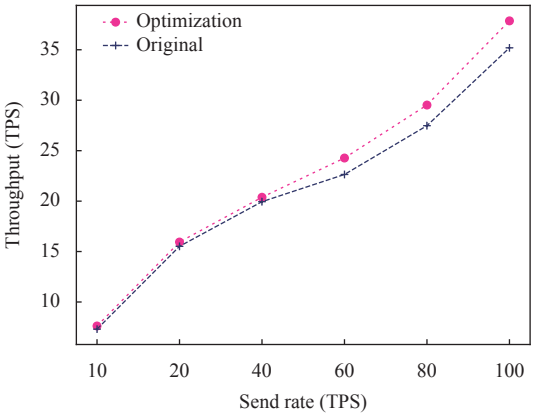

图 6 吞吐量性能测试

此外, 本文在实验过程中发现, 其中最明显的一个对等节点的 CPU 平均占用率从原始方案的 34.32% 减少到优化方案的 16.31%. 综上所述, 本文方案的交易效率较原始方案确有提升, 有效地解决了 HLF 系统中排序阶段易产生的序列化冲突问题, 同时节省了大量的计算与存储资源.

4 总结

针对 HLF 系统中交易进行至排序阶段时容易出现的序列化冲突问题, 提出了基于对应比较图的 HLF 排序机制优化方案, 为交易事务定义重要度并对重要度参数进行对应比较图的拓扑排序, 并根据索引为交易事务全局排序. 最后通过实验对比分析原始方案与本文优化方案的吞吐量性能及有效事务占比, 实验结果表明本文优化方案确实有效解决了 HLF 系统中的序列化冲突问题, 提升了系统交易效率, 节省了大量的计算与存储资源.

参考文献

1 Dinh TTA, Wang J, Chen G, *et al*. Blockbench: A framework for analyzing private blockchains. Proceedings of the 2017 ACM International Conference on Management of Data. Chicago: ACM, 2017. 1085–1100. [doi: 10.1145/3035918.3064033]

2 Gupta S, Hellings J, Rahnama S, *et al*. Building high throughput permissioned blockchain Fabrics: Challenges and opportunities. Proceedings of the VLDB Endowment, 2020, 13(12): 3441–3444. [doi: 10.14778/3415478.3415565]

3 Androulaki E, Barger A, Bortnikov V, *et al*. Hyperledger Fabric: A distributed operating system for permissioned blockchains. Proceedings of the 13th EuroSys Conference. Porto: ACM, 2018. 30. [doi: 10.1145/3190508.3190538]

4 Baliga A, Solanki N, Verekar S, *et al*. Performance characterization of Hyperledger Fabric. Proceedings of the 2018 Crypto Valley Conference on Blockchain Technology (CVCBT). Zug: IEEE, 2018. 65–74. [doi: 10.1109/CVCBT.2018.00013]

5 Gorenflo C, Lee S, Golab L, *et al*. FastFabric: Scaling Hyperledger Fabric to 20 000 transactions per second. Proceedings of the 2019 IEEE International Conference on Blockchain and Cryptocurrency (ICBC). Seoul: IEEE, 2019. 455–463. [doi: 10.1109/BLOC.2019.8751452]

6 Xu L, Chen W, Li ZX, *et al*. Locking mechanism for concurrency conflicts on Hyperledger Fabric. Proceedings of the 20th International Conference on Web Information Systems Engineering. Hong Kong: Springer, 2020. 32–47. [doi: 10.1007/978-3-030-34223-4_3]

7 许沁琪. Fabric 区块链系统事务执行性能优化技术 [硕士学位论文]. 深圳: 中国科学院大学 (中国科学院深圳先进技术研究院), 2021. [doi: 10.27822/d.cnki.gszxj.2021.000144]

8 Sharma A, Schuhknecht FM, Agrawal D, *et al*. Blurring the lines between blockchains and database systems: The case of Hyperledger Fabric. Proceedings of the 2019 International Conference on Management of Data. Amsterdam: ACM, 2019. 105–122. [doi: 10.1145/3299869.3319883]

9 Sousa J, Bessani A, Vukolic M. A byzantine fault-tolerant ordering service for the Hyperledger Fabric blockchain platform. Proceedings of the 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN). Luxembourg: IEEE, 2018. 51–58. [doi: 10.1109/DSN.2018.00018]

10 Sun QC, Yuan YY, Guo T, *et al*. A trusted solution to Hyperledger Fabric reordering problem. Proceedings of the 8th International Conference on Dependable Systems and Their Applications (DSA). Yinchuan: IEEE, 2021. 202–207. [doi: 10.1109/DSA52907.2021.00031]

11 Barabási AL, Albert R. Emergence of scaling in random networks. Science, 1999, 286(5439): 509–512. [doi: 10.1126/science.286.5439.509]

12 任晓龙, 吕琳媛. 网络重要节点排序方法综述. 科学通报, 2014, 59(13): 1175–1197. [doi: 10.1360/972013-1280]

13 李劲, 岳昆, 蔡娇, 等. 基于距离度量的多样性图排序方法. 软件学报, 2018, 29(3): 599–613. [doi: 10.13328/j.cnki.jos.005455]

14 Li RH, Yu JX. Scalable diversified ranking on large graphs. IEEE Transactions on Knowledge and Data Engineering, 2013, 25(9): 2133–2146. [doi: 10.1109/TKDE.2012.170]

15 Pal RK. CompleteGraphSort: A complete graph structure based sorting algorithm. Proceedings of the 2011 IEEE International Conference on Computer Science and Automation Engineering. Shanghai: IEEE, 2011. 193–197. [doi: 10.1109/csae.2011.5952832]

16 Behera BD. Sorting an array using the topological sort of a corresponding comparison graph. Theoretical Computer Science, 2020, 845: 76–97. [doi: 10.1016/j.tcs.2020.09.004]

17 Haveliwala TH. Topic-sensitive PageRank: A context-sensitive ranking algorithm for Web search. IEEE Transactions on Knowledge and Data Engineering, 2003, 15(4): 784–796. [doi: 10.1109/TKDE.2003.1208999]

(校对责编: 孙君艳)