计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.190800148

面向混合索引的区块链系统的可查询性优化

郑浩瀚　申德荣　聂铁铮　寇　月
东北大学计算机科学与工程学院　沈阳 110169
(haohanzheng@foxmail.com)

摘　要　区块链技术具有去中心化和不可篡改性等特性,被认为是下一代的颠覆性核心技术。然而,现有区块链系统在数据管理方面的性能较弱,通常只能根据 Hash 值查询相关交易。当前对于查询的研究大多是将数据同步存储到外部数据库中,通过借用外部数据库进行查询,或是研究如何保证全节点的可靠性,没有从实际意义上解决区块链查询效率低下的问题。文中提出了一种新的解决方案。首先,将区块链数据划分成不同属性;其次,根据不同数据属性,结合区块链本身的 Merkle 树和多种索引结构,提出了一种新的索引——MHerkle 树,该结构在充分保证区块链不可篡改性的情况下增强了区块链的查询性能;然后,设计了 MHerkle 树的索引构建算法,并根据索引提出了基于不同属性的查询算法以及范围查询算法;最后,通过实验验证了所提索引的可行性和有效性。

关键词:区块链;查询;索引;不可篡改;优化
中图法分类号　TP391

Queryability Optimization of Blockchain System for Hybrid Index

ZHENG Hao-han,SHEN De-rong,NIE Tie-zheng and KOU Yue
College of Computer Science and Engineering,Northeastern University,Shenyang 110169,China

Abstract　Blockchain technology has the characteristics of decentralization and immutability,and is considered to be the next generation of disruptive core technology. However,the existing blockchain system is weak in data management and can only query related transactions according to the hash value. The current research on query mostly stores data synchronously into an external database,and then uses an external database to query,or focuses on how to ensure the reliability of the whole node,but the problem of low query efficiency of blockchain remains unsolved in a practical sense. A new solution is proposed in the paper. First,dividing blockchain data into different attributes. Next,based on different attributes,combining with the Merkle tree of the blockchain and multiple index structures,a new index-MHerkle tree-is proposed to enhance the query performance of the blockchain,while ensuring the immutability of blockchain;Then,the index construction algorithm of MHerkle tree is designed,and the query algorithm based on different attributes and the range query algorithm are proposed according to the index. Finally,experiment shows the feasibility and effectiveness of the proposed index.

Keywords　Blockchain,Query,Index,Immutability,Optimization

1　引言

随着比特币、以太坊等区块链系统的飞速发展,区块链技术作为其底层技术也逐渐成为了最受关注的技术之一。区块链技术是分布式存储技术、密码学、共识机制等多种技术的混合体,可以解决传统支付中对第三方的信任问题。区块链本质上是一种去中心化的数据库,其以一种可验证且永久保存的方式记录互相不信任的参与方之间的交易,并将其保存到区块中。区块链真正做到了在分布式系统中让节点在互不信任的情况下互相交易,被认为将改变传统的互联网和金融行业。

随着区块链技术的不断推广,研究人员对区块链本身的应用也从最初的仅仅作为数字货币扩展到了各行各业,对区块链本身查询性能的要求也因此逐步提高。然而,当前区块

到稿日期:2019-08-30　返修日期:2019-12-06　本文已加入开放科学计划(OSID),请扫描上方二维码获取补充信息。
基金项目:国家自然科学基金(61672142,U1811261,61602103);国家重点研发项目(2018YFB1003404);辽宁省自然科学基金(20180550321);中央高校基本科研业务费(N171606005)
This work was supported by the National Natural Science Foundation of China(61672142,U1811261,61602103),National Key R&D Program of China(2018YFB1003404),Liaoning Science and Technology Foundation (20180550321) and Fundamental Research Funds for the Central Universities(N171606005).
通信作者:申德荣(shendr@mail.neu.edu.cn)

链的查询性能极其低下，仅支持以交易哈希为关键字等少数查询，完全无法满足基本的查询需求。例如，在供应链中，商品 g 在 t_n 时刻被供应商 c_n 买下，用户可能想查询 t_1 至 t_2 时刻被买下的商品；又比如，用户想根据商品名称查询该商品之前被买下的记录。此时，只能全局遍历查询，查询效率很低。

本文的贡献如下：1)将区块链中的数据划分成不同属性，提出了一种混合多种索引的 MHerkle 索引(Merkle-Hybrid 索引)，支持基于索引的快速查询；2)基于 MHerkle 索引提出了多种查询算法，并针对不同属性提出单值查询和范围查询；3)通过实验证明，该索引结构具有良好的查询性能。

2 相关工作

继比特币[1]、以太坊[2]、HyperLedgerr[3]之后，区块链技术在近年呈现高速发展的趋势，并在多个方面取得了突破性进展。例如，随着区块链系统的发展，出现了 PoW，PoS 和 PBFT 等多种共识算法；Filecoin[4]和 Storj[5]力求将区块链不可篡改特性加入分布式存储系统中；BLOCKBENCH[6]是支持多种区块链系统性能的测试框架；以太坊和 OmniLedger[7]基于分片来提升当前区块链系统的吞吐量。

近年来，研究人员尝试将区块链的去中心化和不可篡改性融入数据管理中。针对区块链的数据管理性能低的不足，研究者们提出了有关区块链的数据管理研究，如提升存储效率的研究、兼顾存储效率和查询性能的研究，以及提升查询效率和丰富查询语义方面的研究。

在提升存储效率和兼顾查询性能方面，文献[8]针对基于比特币的简单支付验证(Simple Payment Verification，SPV)会造成实际意义上的中心化问题，提出由一组节点共同存储区块链上的所有区块，以达到去中心化的目的，并降低节点的存储负担；文献[9]提出了一种存储系统 UStore，该系统为每个 key 值维护了一张有向无环图以方便 key 值的历史版本查询，具有丰富的语义，并可用作区块链系统的底层存储；文献[10]则提出了 UStore 的改良版本 ForkBase，由 Merkle 树改良生成 POS-Tree，其在提高存储性能的同时提高了查询性能。

在提升查询效率和丰富查询语义方面，Li 等[11]基于区块链的特点，设计了 EtherQL 系统，将区块链数据拷贝到外部数据库中，借助外部数据库的功能管理数据。文献[12]针对 Fabric 在处理时态历史数据方面的不足，通过添加副本和向元数据中插入数据来提高 Fabric 处理历史数据的效率。文献[13]提出了一套轻量级的可验证查询框架 vChain，其主要思想是用户可以使用类似于比特币全节点的节点进行数据查询，并采用密码学技术验证结果的正确性和完整性，从而降低查询验证的代价。文献[14]则进一步提出，若支持索引更新，则验证查询结果的代价过大。为此，该文引入了 MB 树索引，该索引通过结合 Merkle 树和 B+树，使区块链从传统的基于 Hash 查询数据变为根据 key 值查询数据；进一步地，提出了 GEM^2-tree，通过拆分 MB 树保证了更新效率。VQL[15]同样基于外部数据库进行存储，与 EtherQL 不同的是 VQL 将中间层数据 Hash 也加入到了数据库中，使得中间层数据具备可验证性。同时，也有研究者致力于将区块链不可篡改的特性加入数据库中，如 BigChainDB[16]，其设计起源于分布式数据库，通过向其中加入区块链特性，使该系统可以兼具数据安全性和查询高效性，其数据量可以达到 PB 级别。ChainSQL[17]和 TrustSQL[18]也致力于类似的研究。

通过分析，有关查询效率和查询语义的研究存在如下局限性：1)将区块链数据同步到外部数据库中，无法保证数据的不可篡改，同时会增加系统的额外开销；2)将区块链的特性添加到传统数据库中，没有提升区块链本身的查询性能；3)在使用全节点进行查询时，设法让全节点返回交易不存在证明，不能提升节点本身的查询效率。可见，现有研究都没有从真正意义上解决区块链数据查询效率低下和查询语义较少的问题。

与已有工作相比，本文的优势如下：1)不使用外部数据库，解决了外部数据库中数据易被篡改的问题；2)侧重改善区块链的原有索引，使用索引结构提升查询性能；3)本文提出的索引结构并不会影响原有区块链数据的存储和查询。

3 索引设计

3.1 区块链现有的索引结构

现有的区块链索引结构主要分为区块结构和链结构，由一些链接在一起的区块组成。比特币区块链结构如图 1 所示。本文主要讨论区块内部的索引结构，以比特币区块链为例，每个区块由一个区块头(block header)和一个区块体(block body)组成。区块头由几组数据组成，第一组数据是一组指向父块的 Hash 值(Pre Hash)，用于将块与区块链中的前一块连接起来；第二组数据是难度(Bits)、时间戳(Timpstamp)和随机数(Nonce)，这组数据与挖矿有关；第三组数据是 Merkle 树根(Merkle Tree Root)，存在于每个区块的开头，区块体会链接到此根值。

如图 1 所示，Merkle 树根所延伸出的 Merkle 树即为区块体。区块体记录了大量的交易数据，其中交易数据以 Hash 形式存储于 Merkle 树的叶节点，非叶子节点的值是依据其孩子节点值的 Hash 值计算而得出的，直至构造出 Merkle 根。在比特币系统中，轻量级节点可通过比较 Merkle 根来确定数据存储是否正确。

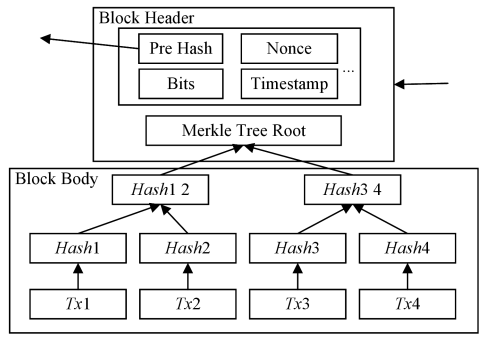

图 1 区块链结构示意图

Fig. 1 Blockchain structure diagram

3.2 Merkle 索引

Merkle 树最显著的一个优点是其提供了交易证明机制。为了计算某个交易数据是否真实存在，用户可先计算出 tx-

hash，并将其定位到交易数据所在区块。如图 1 所示，我们若想计算 $Tx1$ 是否存在，则需要得到 $Hash2$，$Hash3$ 4，并根据 $Tx1$ 的 hash 值与 $Hash2$ 进行 hash 计算生成 $Hash1$ 2，之后再与 $Hash3$ 4 进行 hash 计算，来重构出 Merkle 根，并据此验证交易是否真实存在。

当前区块链结构支持的查询功能较为简单，仅支持基于 Hash 值的相关查询。现有的查询改进研究，大多是在区块链系统中在 LevelDB 上封装查询层，或者新建独立于 Merkle 树的索引结构，无法满足数据不可篡改性。而在关于 Merkle 树的研究中，文献[14]提到的 MB 树索引对原有 Merkle 树结构做出了调整，将传统 Merkle 树中叶子节点的交易 Hash 值替换为数据关键字，并结合 B+树，使得 Merkle 树支持基于关键字的查询；但其无法保证所引入关键字的不可篡改性，同时只能针对关键字属性进行查询，无法实现对多个属性的查询。本文则致力于在不影响原有区块链结构的基础上提升区块链查询性能，并保证新建立索引的不可篡改性。

3.3 MHerkle 索引结构

在统计学中，研究人员常常把变量分为离散型变量和连续型变量。对于区块链中的数据，我们也可以做出类似区分。例如，对于类似于〈“Peter”，1〉和〈“Tom”，2〉的数据，称其中的姓名字段为离散型属性，而数值字段为连续型属性。在本文的查询优化中，对于连续型属性，需要考虑范围查询。基于此，本文首先对区块链中的数据做出定义。

定义 1（数据集合） 令 T 为区块链中存储的数据集合，T_i 为一组数据，*txhash* 为数据的主键，D_i 为 T_i 中的离散型属性集合，C_i 为 T_i 中的连续型属性集合。对于每一个数据集合 T，本文给出如下定义：

$$T=\langle T_1,T_2,\cdots,T_n\rangle$$

$$T_i=\langle txhash,D_i,C_i\rangle$$

根据区块链的特点，我们将数据哈希值定义为数据主键，如〈“Peter”，1〉的主键为 *txhash*＝Hash（“Peter”，1）。已有的区块链系统将 *txhash* 与交易数据在磁盘中的偏移量一一对应，支持根据 *txhash* 查找交易数据，本文将其视为一级索引。本文的工作是基于 Merkle 树构建二级索引，通过非一级索引属性快速找出对应的 *txhash*，从而找到数据，使得区块链可以支持基于非主键属性的查找。

3.3.1 连续型属性索引的构建

本节首先解决连续型属性索引构建问题。对于连续型属性，本文采用 MB 树的基本结构并对 MB 进行了改进。首先采用 Merkle 树存储数据 Hash 值，将 Hash 值作为主键，从而便于根据非主键属性进行查找树的索引构建。查找树与 MB 树的构建方式相同，索引首先根据连续型属性对数据集合进行排序，并对排序后的数据取 Hash 值，再对相邻两个节点的 Hash 值进行 Hash 运算（若数据量为奇数，则在末尾复制最后一个数据以保证数据量为偶数），从而得到父节点，索引中父节点也会保存左右子树连续型属性的最大值，以便于查找。

本文的索引对 MB 树做出的改进之一在于，MHerkle 树会将两个最大值也添加到本节点的 Hash 值中，避免存储的两个最大值遭到篡改。索引构建类似于 Huffman 树从下向上构建的思想，将连续型属性也存储于树中，同时弥补了 MB 树中存储的属性值易被篡改的缺陷，从而保障了 Merkle 树原有的存储结构和交易证明机制。MHerkle 树的不可篡改性将在后文分析。

如图 2 所示，在区块头中，MHerkle 树添加了两个数据，即本区块第一个数据集合和最后一个数据集合的连续型属性值。图 2 所示的例子是以数值型属性构造索引，数值属性对应的数值分别为 0 到 7。考虑一种情况，用户想要查询连续型属性为 3 的所有数据，那么系统会先为用户比较 3 是否在区块头的 start 和 end 之间，然后根据 MHerkle 树中的数值型属性查找 *txhash*，并通过 *txhash* 找到数据。之后顺序查询下一个区块，直到遍历完所有区块。

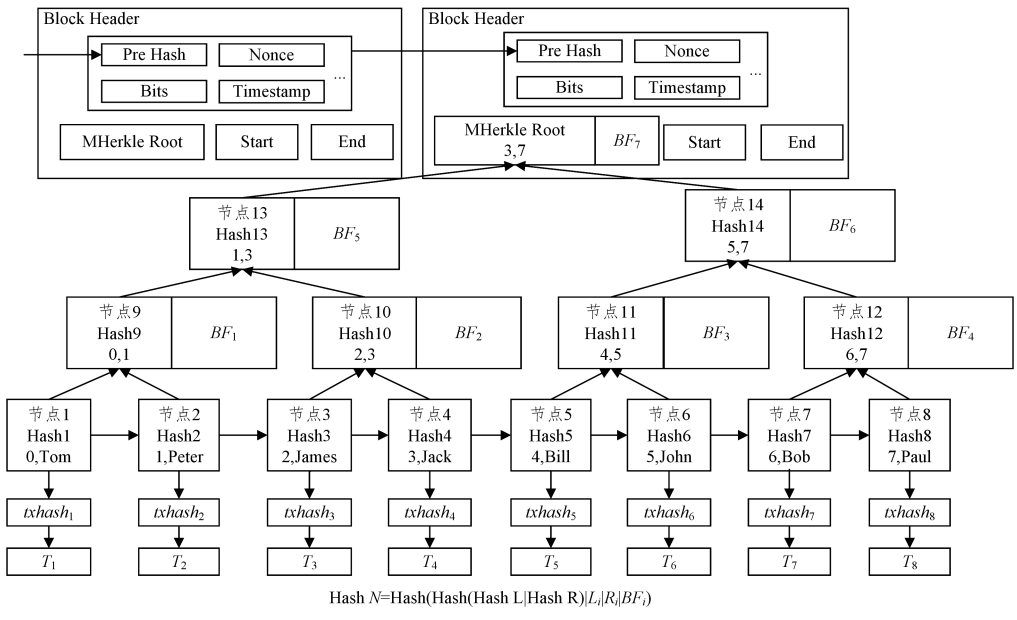

图 2 MHerkle 结构示意图

Fig. 2 MHerkle structure diagram

3.3.2 离散型属性索引的构建

本节提出离散型属性的存储和查询方法。布隆过滤器是由一个二进制向量和一系列随机映射函数组成的，可以高效地检索所查询的元素是否在集合中。

本文将布隆过滤器引入到 MHerkle 树中，在构建索引时，会将节点的左右子树的离散型属性全部存储到布隆过滤器中。在查找离散型属性时，以根节点为起始点，顺次判断节点左右子树是否存在所查找的离散型属性，若存在则往下查询，否则返回“不存在”。

如图 2 所示，其中 BF 为布隆过滤器(Bloom Filter)。在查询离散型属性为 James 的数据时，本文会先从 MHerkle 树的根节点开始，顺序查找 BF_5，BF_2 和叶子节点 2，以找到想要查找的 $txhash$，并找到数据集合 T_3。在查找过程中，由于应用了布隆过滤器，因此可以进行多次剪枝，从而大大提升了查找效率。而对于索引中不存在的数据，由于根节点的布隆过滤器没有对其进行存储，因此会直接查找上一区块。

3.3.3 MHerkle 索引的构建

(1)MHerkle 索引的定义

前文介绍了连续型属性和离散型属性的索引的构建，构建索引时 MHerkle 树将根据查询需求从属性集合中选取用以建树的连续型属性和离散型属性。对于 MHerkle 树中的节点，本文做出如下定义。

定义 2(叶节点) 令 $hash(\cdot)$ 为 Hash 函数；‘|’为运算连接符；T_i 为叶节点对应的数据；$txhash$ 为数据的 Hash 值，同时也是该条数据的主键；D_{ij} 为要查找的离散型属性；C_{ij} 为要查找的连续型属性。对于每一个叶节点，本文对其做出如下定义：

$$txhash=hash(T_i)$$

$$hash_i=hash(T_i|D_{ij}|C_{ij})$$

定义 3(非叶节点) 令 $hash(\cdot)$ 为 Hash 函数，‘|’为运算连接符，l 和 r 分别为节点的左子节点和右子节点，L_i 为左子树所存储的连续型属性的最大值，R_i 为右子树所存储的连续型属性的最大值，BF_i 为节点所存储的布隆过滤器。对于每一个非叶节点，本文对其做出如下定义：

$$hash_i=hash(hash(hash_l|hash_r)|L_i|R_i|BF_i)$$

(2)MHerkle 索引不可篡改性分析

在 MHerkle 索引中，本文的一个创新就是将节点所存储的关键字和布隆过滤器放入了节点对应的 Hash 值中，其目的在于可以基于此进行检查，防止关键字和布隆过滤器遭到篡改。本文对关键字和布隆过滤器的检查提出了两种方案：1)节点定期对 MHerkle 树进行全局重构，重新计算每个节点的 Hash 值，检查是否有节点遭到篡改；2)在查询时对查询到的节点进行 Hash 计算，检查该节点是否遭到篡改，例如，若节点 9 中的值 0 被篡改成 2，则对其重新计算得到的 Hash 值为 hash(hash($Hash_0$|$Hash_1$)|2|1|BF_1)，与节点原有的 Hash 值 $Hash_9$ 不同，可认为该节点遭到篡改。

在进行交易存在性验证时，用户将根据自己的 MHerkle 根验证数据是否存在。用户如果想要检验 T_2 是否存在，则可以根据(1,Peter)，($Hash_1$,0,Tom)，($Hash_{10}$,2,3,BF_2)，($Hash_{14}$,5,7,BF_6)的值反构出 MHerkle 的根 Hash 值。若重构出的 $root$ 与自己的 MHerkle 根的 Hash 值相同，则认为数据存在。在此过程中，用户也可根据 Hash 结果判断索引是否遭到篡改。

4 查询处理方法

4.1 索引构建算法

MHerkle 树采取从下向上构建索引的方法，索引一旦构建完成便不会更新，新的数据只能等待下一区块的索引构建。索引构建过程如算法 1 所示。

算法 1 索引构建算法

输入：无序数据集合 $T(T_1,T_2,\cdots,T_n)$

输出：MHerkle 树节点集合 vMerkleTree

```
1. 根据连续型属性对数据进行排序
2. vMerkleTree.clear();
3. for 1 to n
4.    hash_i=hash(T_i|D_ij|C_ij);
5.    〈hash_i,D_ij,C_ij〉→vMerkleTree;
6. end for
7. for i=T.size() to 1;i=(i+1)/2
8.    for j=1 to i;j+=2
9. hash_ij=hash(hash(hash_l|hash_r)|L_ij|R_ij|BF_ij);
10.   〈hash_ij,L_ij,R_ij,BF_ij〉→vMerkleTree;
11.   end for
12. end for
13. return vMerkleTree;
```

算法 1 中，第 1 行首先根据连续型属性的字典序对数据集合进行排序；第 2－6 行是计算出排序后的数据的 Hash 值，并将其对应的叶子节点放入集合中；第 7－13 行是将子节点的 Hash 值两两进行 Hash 运算，并求出其左右子树所存储的连续型属性的最大值。图 2 中，计算 $Hash_9$ 的值时，先要对叶子节点 1 和叶子节点 2 进行计算，得到 Hash($Hash_1$|$Hash_2$)；之后计算节点 1 和节点 2 的值 0 和 1，以及布隆过滤器 BF_1，并将其加入节点的 Hash 值中，得到 $Hash_9$；后续不断计算，直到得到 MHerkle 根值。

比特币区块链中，Merkle 树采取扁平化存储策略，即将节点的 Hash 值按序存储于 $vMerkleTree$ 中，这样在进行数据验证时就可以通过 $vMerkleTree$ 集合中节点的位置得到数据的验证路径。对于构造出的 $vMerkleTree$，本文将其先存储于内存中，若内存空间不够，再将一部分 $vMerkleTree$ 存储于磁盘中，可以认为本文的 $vMerkleTree$ 是一个内存结构，这样有利于全节点获得更快的查询速度。全节点启动时也会对 $vMerkleTree$ 进行预加载，便于轻量级节点进行查询。在算法 1 中，输出节点集合 $vMerkleTree$ 即是输出 MHerkle 树。

4.2 连续型属性查询算法

4.2.1 单值查询算法

根据之前所创建的索引，本文提供了针对连续型属性的查询算法。

下面仍以数值属性为例进行说明。在本文的区块链系统中，每个区块的区块体对应一个 MHerkle 索引，每个 MHerkle 索引的叶子节点对应一个数据集合 $T(T_1, T_2, \cdots, T_n)$。进行数据查询时，输入条件为区块集合，查询算法会循环调用每一个区块，并找到其中是否存在满足查询条件的数据 T_i。当用户输入数值 c 时，系统首先会与最后一个区块的区块头中的 *start* 和 *end* 进行比较，若 $c \in [start, end]$，则遍历 MHerkle 树，并返回结果，之后顺序比较上一个区块，直到创世区块(genesis block)。数据查询过程如算法 2 所示。

算法 2 数据查询算法

输入：所查询数值 c，区块集合 B

输出：结果集合 R

```
1. while !B. isempty();
2. if c∈[start,end]
3.    node=B. first(). tree. root;
4. end if
5. while node∉leafnode
6.    if c<=node. L;
7.     node=node. left;
8.    else if c>node. L && c<=node. R;
9.     node=node. right;
10.  end if
11. end while
12.  if node. Cij==c&&node∈leafnode
13.  txhash=node. txhash,txhash. Ti→R;
14. end if
15. B. remove();
16. end while
17. return R;
```

算法 2 中，第 1 行循环调用每一个区块；第 2－4 行首先判断所查询的数值 c 是否在该区块中；第 5－11 行则是顺着 MHerkle 树向下查询，直到找到叶子节点，其中第 6－9 行是与节点左右子树最大值进行比较；第 12－14 行是先判断叶子节点的数值属性是否等于要查询的数值，如果是则找到该节点对应的 *txhash*，并通过 *txhash* 找到对应的数据，将其放入结果集 R 中，并顺序判断下一区块。图 2 中，对于 $c=2$，本文会先从 MHerkle 根节点开始，顺序查找节点 13、节点 10 和节点 3，然后找到对应数据的 Hash 值 $txhash_3$，从而得到数据集合 T_3。

4.2.2 范围查询算法

为提供范围查询，本文修改了 MHerkle 树的结构，将叶子节点串联在一起形成一个有序链表，从而方便按照排序顺序向后遍历。图 2 中，从第一个叶子节点(即最中间的节点)开始顺序向后遍历，实现叶子节点的连接操作。范围查询过程如算法 3 所示。

算法 3 范围查询算法

输入：所查询数值范围[cs,ce]，区块集合 B

输出：结果集合 R

```
1. while ! B. isempty();
2. if [cs,ce]∩[start,end]==∅
3.    continue;
4. else if [start,end]∈[cs,ce]
5.    B. first(). T→R;
6. else if [cs,ce]∈[start,end]
7.    node=cs 对应节点;
8.    right=ce 对应节点;
9.    while node! =right
10.   txhash=node. txhash,txhash. Ti→R;
11.     node=node. next;
12.   end while
13.   else if cs∈[start,end]&& ce∉[start,end]
14.   node=cs 对应节点;
15.   while node!=null
16.     txhash=node. txhash,txhash. Ti→R;
17.     node=node. next;
18.   end while
19.   else if cs∉[start,end]&& ce∈[start,end]
20.     right=ce 对应节点;
21.     node=B. first(). leafnodestart;
22.   while node! =right
23.     txhash=node. txhash,txhash. Ti→R;
24.     node=node. next;
25.   end while
26. end if
27. B. remove();
28. end while
29. return R;
```

算法 3 中，第 1 行循环调用每一个区块；第 2－3 行先判断所查询的数值范围与当前区块的数值范围是否无交集，若无交集则顺序查找下一区块；第 4－5 行继续判断当前区块的数值范围是否为所查询的数值范围的子集，若是则将当前区块的所有数据放入结果集中；第 6－27 行则将剩余情况分为 3 种，并分情况讨论每种情况的结果，最后根据边界条件返回结果集 R。

4.3 离散型属性查询算法

由于离散型属性不需要进行范围查找，本文仅提供离散型属性的单值查询算法。根据之前所创建的索引，以姓名属性举例，当用户输入姓名 d 时，系统会从最后一个区块的 MHerkle 树根节点开始遍历，若能找到姓名 d 的对应数据则返回结果；之后顺序比较上一个区块，直到创世区块。数据查询过程如算法 4 所示。

算法 4 离散型属性查询算法

输入：所查询属性 d，区块集合 B

输出：结果集合 R

```
1. while !B. isempty()
2. node=B. first(). tree. root;
3. node→queue;
4. while queue ! isempty()
5.    node=queue. pop();
6.    if node∈leafnode&&node. Dij==d
7.      txhash=node. txhash,txhash. Ti→R;
8.    else if node∉leafnode&&node. BF.
```

```
    contain(d)
9.    node.left→queue,node.right→queue;
10.   end if
11. B.remove();
12. end while
13. return R;
```

算法 4 中，第 1 行循环调用每一个区块。第 2－3 行将区块的根节点放入队列中。第 4－12 行对 MHerkle 树进行层次遍历，其中第 6－7 行判断该节点是否为叶子节点，若是则比较其离散型属性是否为所查询的 d，若是则找到数据 Hash 值，并将其对应的数据放入结果集 R 中；第 8－9 行是判断该节点是否为非叶节点，若是则判断该节点存储的布隆过滤器中是否有所查找的 d，若存在则将其左右子节点放入队列。第 13 行返回最终结果。

5 实验与结果

5.1 实验设置与对比实验

5.1.1 实验设置

本文基于比特币 v0.1.0 源码，改写了其中的区块结构和交易结构，并借助其中的磁盘读写模块，实现了本文实验代码的编写。

实验环境：Intel(R) Core(TM) i7-8700 CPU，内存容量为 8GB，操作系统为 64 位 Windows10；底层数据库采用 BerkeleyDB。

5.1.2 对比实验

本文以比特币原有的结构为对比。由于比特币原有结构并不支持除 Hash 值之外的查询，因此本文改写了比特币原有的交易结构，增加了其中的属性，并将其作为 baseline 方法。

在 baseline 中，本文采取遍历查询。具体查询过程如下：在单一区块中，baseline 会遍历区块中的所有数据，查找包含所查询值的数据，若存在则返回结果。由于本文改写了比特币区块链原有的结构，因此 baseline 方法可以实现基于其他属性的查询。而对于多区块查询，本文会循环调用每一个区块，并基于单区块查询的算法，在每一区块中查询所需要的结果，如存在则返回结果，否则顺序调用下一区块，直到查询到结果。对于不同的实验，本文将分别设置评价标准进行测试。

5.2 实验结果及分析

5.2.1 索引查询效率的比较

本文首先将索引和区块数据全部加载进内存，比较在不同交易量下 baseline 和 MHerkle 索引在内存中对数据查询的性能(在比特币区块链中，交易量即为数据量)。图 3 对比了连续型属性查询的时间；图 4 对比了离散型属性查询的时间。可以看出，随着交易量的增大，baseline 呈现出线性增长趋势；而基于索引的查询时间趋势几乎为水平直线。通过分析可知，由于 baseline 采用遍历方法，其时间复杂度约为 $O(N)$；而基于索引的查询的时间复杂度约为 $O(\log N)$，因此呈现出了上述实验结果。图 5 对比了 MHerkle 索引中连续型属性和离散型属性的查询时间。可以看出，离散型属性的查询时间比连续型属性的查询时间长，这是由于本文离散型属性采用的是字符串变量，而连续型属性采用的是整型变量，因此在同一交易量下会呈现图 5 所示的结果。

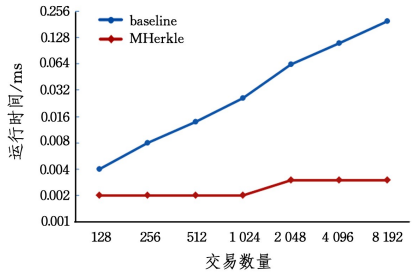

图 3 连续型属性查询时间的比较

Fig. 3 Comparison of continuous attributes' query time

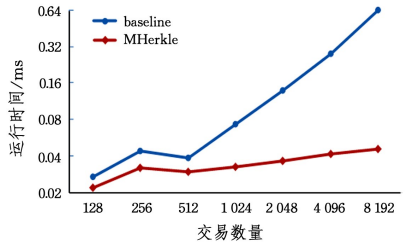

图 4 离散型属性查询时间的比较

Fig. 4 Comparison of discrete attributes' query time

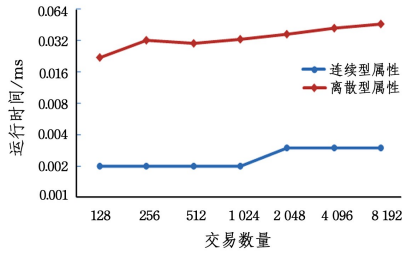

图 5 MHerkle 索引查询时间的比较

Fig. 5 Comparison of MHerkle index's query time

5.2.2 单区块索引查询时间的比较

图 6 给出了单个区块下连续型属性单值查询时间随区块内交易量的变化趋势。可以看出，baseline 的查询时间呈现线性增长趋势，而 MHerkle 索引的查询时间几乎为直线。这是由于 baseline 需要每次从磁盘读取完整的区块数据进行遍历；而基于 MHerkle 索引的查询则只需找到相应交易，直接读取交易数据即可，区块大小对查询时间的影响几乎可以忽略不计。

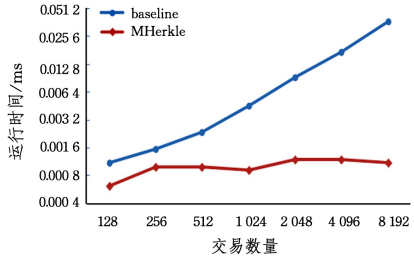

图 6 单区块单值查询时间的比较

Fig. 6 Comparison of single value query time in one block

图 7 显示了单个区块下连续型属性范围查询时间随区块内交易总量变化的趋势，其中所要查询的交易量为 10 笔。图 7 中，baseline 查询时间仍呈现线性增长趋势，且查询时间与单值查询的查询时间几乎相同；而 MHerkle 索引的查询时

间仍然为水平直线。分析实验结果可知，由于 baseline 采用遍历查询，其对单值和范围的查询过程完全相同，因此其时间消耗几乎相同。但是，由于在范围查询时，MHerkle 索引需要针对每笔查询到的交易频繁读取磁盘数据，因此在查询效率上比单值查询稍差。

图 8 给出了单个区块内离散型属性的查询时间，其中坐标横轴为区块内的交易总量。可以看出，离散型属性的查询时间与连续型属性单值查询的查询时间几乎相同。分析可知，由于索引查询时间与数据从磁盘读取的时间差距过大，离散型属性与连续型属性的索引查询时间的差值几乎可以忽略不计，实验结果符合预期。

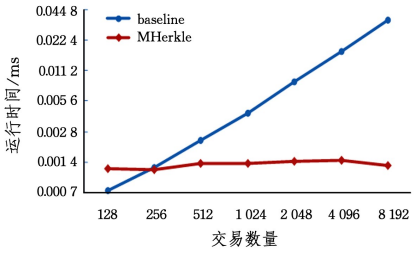

图 7 单区块范围查询时间的比较

Fig. 7 Comparison of range value query time in one block

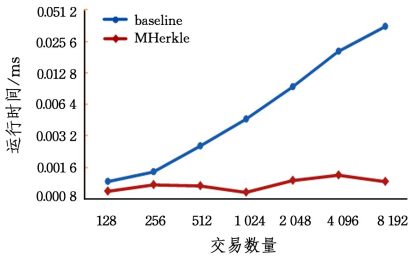

图 8 单区块离散型属性查询时间的比较

Fig. 8 Comparison of discrete attributes' query time in one block

5.2.3 多区块索引查询时间的比较

为了便于实验，多区块情况下的实验数据均不会重复，以便于我们根据数据所在区块深度做出对比。图 9 给出了多个区块下连续型属性单值查询时间随数据所在区块深度变化的趋势，其中交易总量为 32 278，单个区块内的交易总量为 2048。实验结果显示，baseline 的查询时间随区块深度呈现线性增长趋势，而 MHerkle 索引的查询时间变化趋势几乎为水平直线。分析实验结果可知，由于 baseline 需要将每一个所查询的区块读取到磁盘中，直到查找到结果，因此其查询时间与区块深度线性相关；而基于 MHerkle 索引的查询则不受区块深度的影响。

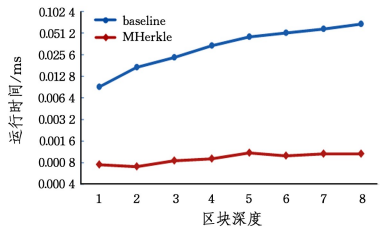

图 9 多区块单值查询时间的比较

Fig. 9 Comparison of single value query time in multiple blocks

图 10 给出了多个区块下连续型属性范围查询时间随数据所在查询的交易量范围变化的趋势，其中交易总量为 32278，单个区块内的交易总量为 2048。

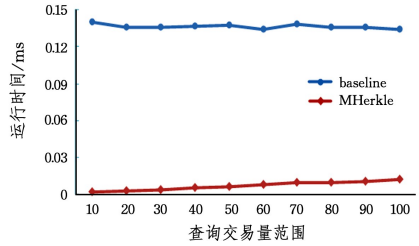

图 10 多区块范围查询时间的比较

Fig. 10 Comparison of range value query time in multiple blocks

可以看出，基于 MHerkle 索引的查询效率比 baseline 的查询效率高，并且随着所要查询的交易量的上升，baseline 的查询时间呈水平直线趋势，而基于 MHerkle 索引的查询时间则直线上升。分析可知，baseline 采用遍历查询，因此当进行范围查询时，其查询时间并不会随着所要查询的交易量的变化而变化；而 MHerkle 索引则要对每笔查询到的交易进行磁盘读取，因此查询时间随查询交易量范围的增大而线性上升。

图 11 给出了多个区块下离散型属性的查询时间，横坐标为数据所在区块的深度，其中交易总量为 16 384，单个区块内的交易总量为 1 024。查询时间所呈现的趋势与连续型属性的结果类似，实验结果符合预想结果。

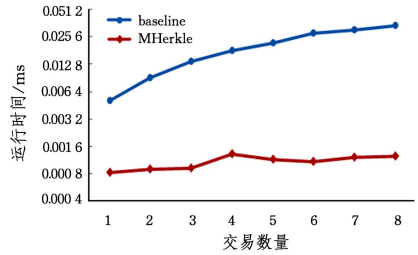

图 11 多区块离散型属性查询时间的比较

Fig. 11 Comparison of discrete attributes' query time in multiple blocks

结束语 本文首先对区块链中的数据进行划分，接着结合多种索引结构，对不同数据属性建立相应的索引，并提出了 MHerkle 树索引，该索引在不改变区块链不可篡改性的基础上，使得查询性能大大提升；之后基于 MHerkle 树，提出了基于多种属性的单值和范围查询的算法。实验结果表明，本文提出的索引可提高查询效率，使得区块链可以适用于更多的场景。针对本索引中连续型属性的值不能重复的缺陷，下一步拟在叶子节点引入 Hash 桶；针对范围查询中需要多次从磁盘读取的问题，将继续寻找解决方法；根据现有索引结构，设计数据存在性和不存在性的证明；继续探索更高效的查询算法。

参 考 文 献

[1] NAKAMOTO S. Bitcoin：A peer-to-peer electr-onic cash system [EB/OL]. https://bitcoin.org/bitcoin.pdf.

[2] BUTERIN V. Ethereum：A next generation sma-rt contractand

decentralized application platform [EB/OL]. https://github.com/ethereum/wiki/Whitecom/ethereum/wiki/White-Paper.

[3] ANDROULAKI E, BARGER A, BORTNIKOV V, et al. Hyperledger fabric: a distributed operating system for permissioned blockchains[C]// Proceedi-ngs of the Thirteenth EuroSys Conference. ACM, 2018: 30.

[4] PROTOCOL LABS. Filecoin: A Cryptocurrency Operated File Storage Network[EB/OL]. https://filecoin. io/filecoin. pdf.

[5] SHAWN W. Storj A Peer-to-Peer Cloud Storage Network[EB/OL]. https://storj. io/storj. pdf.

[6] DINH T T A, WANG J, CHEN G, et al. BLOC-KBENCH: A Framework for Analyzing Private Bl-ockchains[C]// Special Interest Group on Managem-ent Of Data. 2017: 1085-1100.

[7] KOKORIS-KOGIAS E, JOVANOVIC P, GAS-SER L, et al. OmniLedger: A Secure, Scale-Out, D-ecentralized Ledger via Sharding[C]// IEEE Symposium on Security and Privacy. 2018: 583-598.

[8] XU Z H, HAN S Y, CHEN L. CUB, a Consens-us Unit-based Storage Scheme for Blockchain System [C]// IEEE International Conference on Data Engineering. 2018: 173-184.

[9] DINH A, WANG J, WANG S, et al. UStore: A Distributed Storage With Rich Semantics[J]. arXiv: 1702. 02799, 2017.

[10] WANG S, DINH T T A, LIN Q, et al. ForkBas-e: An Efficient Storage Engine for Blockchain and Forkable Applications[J]. Proceedings of the VLDB Endowment, 2018, 11(10): 1137-1150.

[11] LI Y, ZHENG K, YAN Y, et al. EtherQL: A Q-uery Layer for Blockchain System[C]// Database S-ystems for Advanced Applications. 2017: 556-567.

[12] HIMANSHU G, SANDEEP H, KUSHAGRA A, et al. Efficiently Processing Temporal Queries on Hyperledger Fabric [C]// IEEE International Confer-ence on Data Engineering. 2018: 1489-1494.

[13] XU C, ZHANG C, XU J. vChain: Enabling Verifiable Boolean Range Queries over Blockchain D-atabases[C]// Special Interest Group on Management of Data. 2019: 141-158.

[14] ZHANG C, XU C, XU J, et al. GEM^2-Tree: A Gas-Efficient Structure for Authenticated Range Queries in Blockchain[C]// IEEE International Conference on Data Engineering. 2019: 842-853.

[15] PENG Z, WU H T, XIAO B, et al. VQL: Provi-ding Query Efficiency and Data Authenticity in Blockchain Systems[C]// IEEE International Confe-rence on Data Engineering Workshops. 2019: 1-6.

[16] TRENT M, RODOLPHE M, ANDREAS M, et al. BigchainDB: A Scalable Blockchain Database [EB/OL]. https://www. bigchaindb. com/whitepaper/bigchaindb-whitepaper. pdf.

[17] BEIJING PEERSAFE TECHNOLOGY CO, LTD. White paper for blockchain database applicat-ion platform [EB/OL]. http://www. chainsql. net.

[18] TENCENT FIT, TENCENT RESEARCH INSTITUTE. White paper for tencent trustSQL[EB/OL]. https://trustsql. qq. com.

ZHENG Hao-han, born in 1995, postgraduate. His main research interests include blockchain and so on.

SHEN De-rong, born in 1964, Ph.D, professor, is a senior member of CCF. Her main research interests include Web data processing and distributed database.