doi:10.11835/j.issn.1000-582X.2023.07.002

基于状态树的链上数据高效可信查询索引模型及方法

原 旭,黄笠煌,陈志奎,于 硕
(大连理工大学 软件学院,辽宁 大连 116620)

摘要:区块链技术以其去中心化,不可篡改等特性在分布式数据管理领域中逐渐得到关注。但区块链系统在数据查询处理方面存在查询功能单一、效率低以及查询可信性难以保证等问题。笔者基于以太坊状态树的设计思路,在保证索引不可篡改的前提下,提出一种全局索引结构KMPT,可一次定位目标区块,避免了遍历区块的检索过程,同时结合块内索引TMPT,实现了基于内容的高效区块链数据检索。经实验验证,相比于仅构建块内索引的方法,该索引模型在可接受的索引构建代价内极大提升了查询检索的效率和稳定性,还可同时提供查询数据存在或不存在证明,提升了查询结果的可信性。

关键词:区块链数据查询;区块链内容检索;可信索引模型;不可篡改索引;状态树
中图分类号:TP311 **文献标志码:**A **文章编号:**1000-582X(2023)07-009-14

Efficient and trusted query index model and method for blockchain data based on Merkle Patricia tree

YUAN Xu, *HUANG Lihuang*, *CHEN Zhikui*, *YU Shuo*
(School of Software Technology, Dalian University of Technology, Dalian, Liaoning 116620, P. R. China)

Abstract: Blockchain technology has attracted significantly attention in the field of distributed data management because of its decentralized and immutable nature. However, current blockchain systems face limitations in data query processing including single query function, low query efficiency and difficulties in ensuring query credibility. To address these challenges, in this paper, a global index structure called KMPT is proposed, inspired by the design concept of Ethereum Merkle Patricia tree on the premise of ensuring the immutability of index. The KMPT structure aims to realize the function of locating the target block at one time, avoiding the retrieval process of traversing blocks. Furthermore, by incorporating the intra-block index TMPT, the proposed approach enables high-efficiency content-based blockchain data retrieval. Experiments demonstrate that, compared with the method of only building intra block index, the proposed index model significantly improved the efficiency and stability of query retrieval within the acceptable index construction cost. In addition, it can provide the proof of existence or

收稿日期:2021-08-12
基金项目:国家自然科学基金项目资助(62076047);中央高校基本科研业务费专项资金资助(DUT20LAB136,DUT20TD107)。
Supported by National Natural Science Foundation of China (62076047), and the Fundamental Research Funds for the Central Universities (DUT20LAB136, and DUT20TD107).
作者简介:原旭(1970—),男,副教授,主要从事人工智能与区块链、工业互联网方向研究,(E-mail) david@dlut.edu.cn。
通信作者:陈志奎(1968—),男,博士,教授,主要从事大数据计算、物联网和人工智能方向研究,(E-mail) zkchen@dlut.edu.cn。

non-existence of data query at the same time, enhancing the credibility of query results.

Keywords: blockchain data query; blockchain content retrieval; trusted query index model; immutable index; Merkle Patricia tree

随着比特币[1],以太币[2]等加密货币的兴起,其底层区块链技术得到了越来越多的关注。区块链以其去中心化、不可篡改、多方共享以及可回溯特性[3]为解决数据可信存储问题提供了新的可能。在区块链中,共识算法负责数据的写入,在如何提升共识算法效率方面,目前已有很多研究并且取得了不错的成效[4],但在对区块链数据库的读性能,即查询处理方面的研究相对较少。现有的区块链系统在数据查询处理方面仍暴露出很大的局限性,如比特币系统中仅支持根据交易哈希值查询具体交易,而无法针对交易的具体细节展开查询;以太坊支持账户查询,由于其引入了状态树[5]维护全局账户状态[6],可根据账户地址对账户状态进行高效的查询,但状态树本身与交易并无关联,查询交易数据仍需通过交易哈希值进行查询。在面向区块链中数据的查询处理优化技术方面,于戈等[7]首先提出将区块链用于解决不可信环境下的分布式数据管理问题,深入讨论了区块链系统与传统分布式数据库之间的异同点,系统地总结了目前区块链系统在数据存储结构、查询处理方面的优化技术。王千阁等[8]总结了当前区块链系统因数据存储模式限制而普遍面临着查询功能简单、查询性能较低等严重问题,深入研究了区块链系统数据存储与查询优化之间的关系。

目前区块链系统在数据查询方面的优化方法大体可分为两类:一类是外联数据库方法,很多研究者将区块链中的数据加载到链下数据库中存储,利用已有成熟的数据库索引技术实现了对区块链数据高效而丰富的检索。Li等[9]将以太坊区块链数据和K-V数据库里的数据加载到MongoDB数据库中,利用MongoDB进行查询操作,包括范围查询和Top-K查询,查询效率和查询功能丰富性都得到了提高。通过区块链网络同步数据库操作以更新链下数据库,并利用链下数据库对外提供高效丰富的查询访问服务。这类工作本质上都是将区块链数据转移到链下数据库中存储,难以保证数据和索引的不可篡改性,牺牲了数据安全性,同时增加了额外的存储负担,没有从根本上解决区块链系统数据查询的问题[10-11]。另一类是内置索引方法,与外联数据库方法不同,内置索引方法是在原有的区块链数据结构上进行设计修改,将索引内嵌于区块结构中,没有给系统额外带来过大的存储负担,致力于在保证数据不可篡改性的同时丰富查询功能并提升查询效率。例如,焦通等[12]提出了一种可查询区块链数据的模型Blockchain DB,并设计了一种基于红黑树和Merkle树结合的块内索引结构MerkleRBTree,实现基于Key值的数据记录最新版本查询。贾大宇等[13]提出了一种区块链存储容量可拓展模型的高效查询方法—ElasticQM,采用一种基于B-M树的区块存储结构组织块内交易来加快块内内容检索的效率。Zhang等[14]用数据关键字来代替传统Merkle树中叶子节点存储的交易哈希值,并结合B+树设计了MB树索引,实现了基于数据关键字的查询。块内进行索引设计可以加快块内内容检索的速度,但仍需从最新区块开始向前遍历区块检索,当目标数据所在区块深度较深时,这种检索方式由于在无关区块中进行了大量的无效搜索,降低了检索效率,更糟糕的是,若目标交易不存在,需遍历到创世区块才能返回目标交易数据不存在信息,检索响应慢,且无法提供数据不存在性证明,查询可信性无法验证。

为加快遍历检索过程中无关区块的过滤速度,郑浩瀚等[15]对MB树索引进行了改进,提出一种混合块内索引结构,标识本区块存储交易连续属性值的范围,在区块头中引入一个布隆过滤器[16],标识本区块交易离散属性值集合,在开始对某区块进行检索前先将查询目标值和块头信息比较,以此来避免对无关区块进行无效搜索,该混合索引结构在保证索引不可篡改的同时实现了针对连续型和离散型变量的检索,并加速了检索过程中区块的过滤速度。现有系统如比特币、以太坊等也都通过在区块头引入布隆过滤器来避免对无关区块的无效搜索,以提高检索速度。然而,在区块头引入布隆过滤器或设置标识的方法虽然加快了区块的过滤速度,但存在2个问题:一是布隆过滤器存在假阳性错误,有一定的误报率,无法实现完全准确的过滤,设置标识字段方法也存在同样的问题;二是方法虽避免了对大部分区块的无效检索,提升了检索效率,但过滤区块仍需遍历区块头,遍历过程中不断产生的磁盘IO过程限制了检索的效率。

综上所述,目前区块链数据查询处理研究存在如下局限。

1)外联数据库方法无法保证数据不可篡改性,损失了区块链特性,同时给系统带来了额外的存储负担,没有从根本上解决区块链系统的查询问题。

2)内置索引方法仅针对块内数据构建索引,无法确定目标数据所在区块。因此,需遍历区块进行检索,难以避免在大量无关区块中的无效搜索过程,大大降低了查询检索的效率。

3)当目标数据不存在时,仅使用块内索引方法查询响应慢,且无法向轻节点提供数据不存在证明,查询结果的正确性无法验证,查询可信性差。

针对以上问题,在保证数据和索引不可篡改的前提下,笔者提出了一种基于以太坊状态树的全局区块索引模型KMPT(key-Merkle Patricia tree),完成由数据内容到目标区块之间的映射,实现由最新区块一次锁定目标数据所在的区块,避免了遍历区块的检索过程,同时可提供数据不存在性的证明。此外,对块内数据记录设计块内索引TMPT(transaction-Merkle Patricia tree),加快在目标块中的检索速度,提供数据存在性的证明。实现基于内容的高效可信查询,属于内置索引方法范畴。本研究的主要贡献如下。

1)针对内置索引方法在检索中如何避免遍历区块的问题,基于以太坊状态树设计了一种针对区块链中数据内容的不可篡改全局区块索引,一次锁定目标块,避免了遍历无关区块检索的过程,结合块内索引,极大提升了基于数据内容对区块链数据记录的检索效率。

2)基于提出的索引模型,可同时向轻节点提供查询目标数据存在性和不存在性证明,提升了查询可信性。

1　索引模型

1.1　数据及数据操作定义

为解决现有区块链系统中数据模型单一问题,Blockchain DB[13]模型中将交易数据结构扩展到任意记录格式。笔者在Blockchain DB数据模型的基础上研究查询方法,将该模型中的数据和数据操作定义如下。

1.1.1　数据结构

为使区块链中数据更具一般化,将原有交易结构重新定义。如图1所示,交易由交易头(transaction)和数据(data)组成。数据记录两者表述等价。

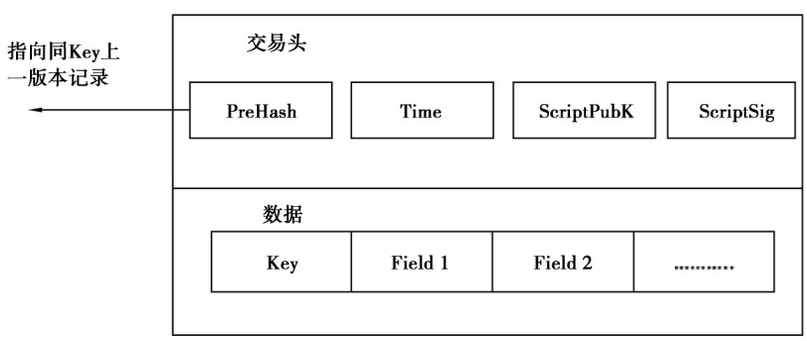

图1　交易结构图

Fig. 1　The structure of Transaction

其中交易头包含PreHash,指向相同Key的上一版本记录;Time是该版本记录的发布时间;ScriptPubK指定拥有该记录写操作权限的下一拥有者公钥,ScriptSig是拥有本记录写操作权限的拥有者的签名,两者用于数据写权限控制。

1.1.2　数据操作

增加记录:向区块链数据库中添加某一新Key记录时,数据拥有者可以指定下一拥有者对该条记录写操作权限的公钥,然后用自己的私钥发布该记录签名。

修改记录:针对某一Key记录的修改通过发布一个新的相同Key记录以追加方式实现。只有拥有对该Key记录写操作权限的操作者才可修改记录,记录发布后需验证其私钥签名是否与父记录中的公钥匹配,匹

配则有效,否则该修改无效。

查询记录:所有参与者都可以进行查询操作,对以Key为关键字的查询会返回其最新的版本,可通过溯源操作查询所有修改的历史版本。

删除记录:数据一旦写入便无法删除。

1.2 索引模型

索引模型如图2所示,分为块内TMPT索引和全局状态索引KMPT,TMPT与KMPT都基于以太坊MPT结构实现。与以太坊中交易树和状态树类似,TMPT组织块内数据记录,而KMPT维护全局Key记录状态。所不同的是,以太坊中交易树是由块内交易序号构建的MPT,状态树是由账户地址编码构建的MPT,而本文中TMPT与KMPT都是根据数据记录Key值来构建的。

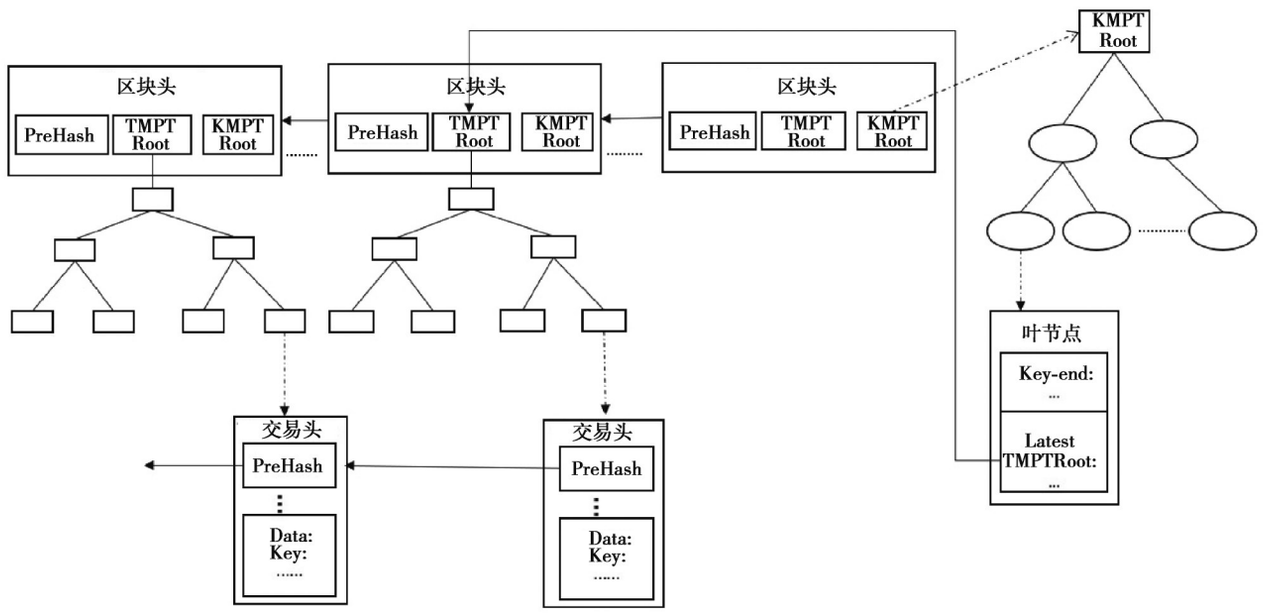

图2 索引结构图

Fig. 2 The structure of index

如图2所示,每个区块内数据记录存储于TMPT叶节点中,TMPT以块内数据Key值为关键字构建而成,组织块内数据,同时将TMPT根哈希值添加到区块头中。TMPT是一棵多叉搜索树(前缀树),与基于二叉搜索树的块内索引结构相比检索效率更高。此外,每个区块头中再添加一个根哈希值KMPT Root,链接到当前的全局状态索引树KMPT树根,KMPT是由Key值构建的一棵MPT,与以太坊状态树叶节点中Value存储账户余额信息不同,KMPT叶节点中存储一个哈希值LatestTMPTRoot,指向当前该Key记录最新版本所在区块的TMPT树根,实现定位目标区块的功能,避免了遍历区块检索目标数据的过程。需说明的是,KMPT叶节点中维护的是目标数据所在区块TMPT根哈希,得到此根哈希后可以直接开始块内检索,而块内检索的过程即是提供目标数据Merkle存在性证明路径的过程,这样便保证了查询结果的可验证性,也是之所以不在KMPT叶节点中直接存储目标数据最新版本哈希而是维护其所在TMPT根哈希的原因。

值得一提的是,块内TMPT检索树只组织本区块内数据记录,加快在区块内部基于Key值的查找效率,而KMPT维护的是全局Key状态,且每个区块都有一颗完整的KMPT,存储了到本区块时刻的全局Key记录状态信息。为了减轻存储负担,这里KMPT采用与以太坊状态树相同的存储策略,即区块间共享KMPT树中节点,构建和更新索引时,仅对本区块内数据所关联的Key状态以新建分支的方式进行,并与前序区块共享其他未改变状态的KMPT节点,以减小存储开销。

1.2.1 数据插入

插入算法是TMPT和KMPT构建的基础,以图3为例说明在MPT中插入4个Key-Value对的构建过程。

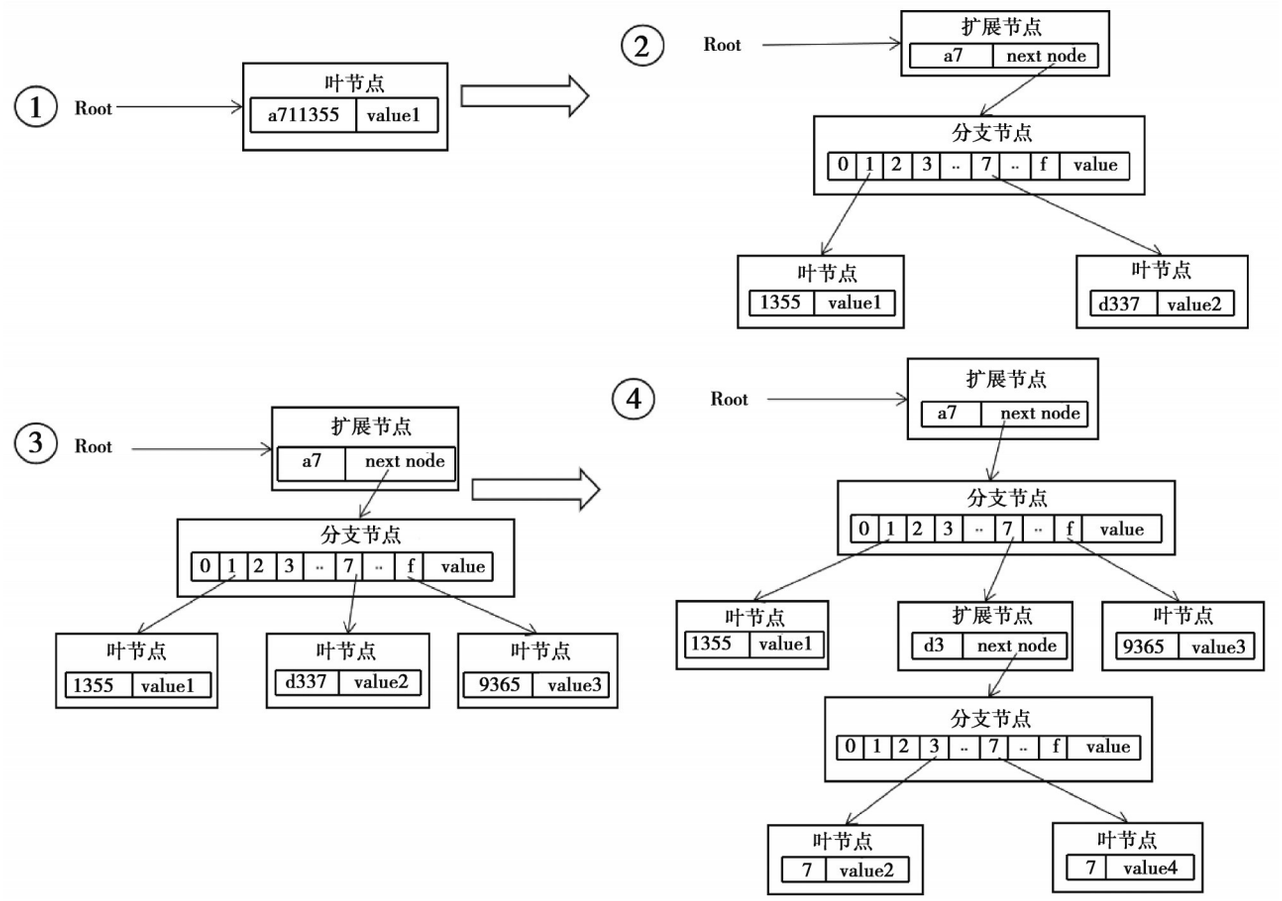

图 3　MPT 构建过程示意图

Fig. 3　Schematic diagram of MPT construction process

如图 3-①所示,首先插入<a711355,value1>,由于此时树中只有一个节点,则将该 Key-Value 对直接保存为叶节点,且此时该叶节点为树的根节点。

如图 3-②所示,插入<a77d337,value2>,由于该 Key 与 a711355 共享前缀 a7,故创建共享前缀为 a7 的扩展节点为根节点,且创建一个分支节点链接两个叶节点。

如图 3-③所示,继续将<a7f9365,value3>插入,也是与之前节点共享前缀 a7,故只需在分支节点上新增一个叶节点即可。

最后插入<a77d397,value4>,如图 3-④所示,该 Key 与 a77d337 共享前缀 a77d3,因此需再创建一个共享前缀为 d3 的扩展节点,并新建一个分支节点链接两者。

需说明的是,每次插入一个节点后须从下至上更新其所在分支路径上的哈希值及根哈希,节点哈希值计算方法与 Merkle Tree 类似,规则如下。

叶节点

$$\text{Hash(Leaf Node)}=\text{Hash(Node.Key-End,Node.Value)},$$

扩展节点

$$\text{Hash(Extension Node)}=\text{Hash(Node.Shared nibble(s),Node.Next Node)},$$

分支节点

$$\text{Hash(Branch Node)}=\text{Hash(Node.Children[0],...,Node.Children[f],Node.Value)}。$$

分支节点哈希值的计算方法是将存储的每个子节点哈希值与节点 Value 中存储的值一起进行哈希,最后得到该分支节点哈希值,若存储哈希值为空,则使用空字节参与运算。

在索引模型中，TMPT与KMPT的构建都基于以上插入算法。区别在于TMPT维护的Key–Value对中，Key为数据的Key值，Value为具体交易数据的哈希值，而在KMPT中，Key为数据Key值，Value为Key记录最新版本所在TMPT根哈希值，即LatestTMPTRoot。

1.2.2 索引构建

索引构建以区块为单位，首先根据区块内数据Key值构建块内TMPT索引，块内索引构建完成后将TMPT根哈希添加到区块头中。然后遍历块内数据记录，以新建分支方式构建及更新KMPT，最后将KMPT根哈希添加到区块头中完成索引构建。相应的索引构建算法.

算法1. 索引构建算法：

输入：已验证交易集合transactionMap

输出：Block

```
Begin:
TMPTRoot=null;          //初始化TMPTRoot
for(Transaction tx: transactionMap){          //遍历交易集合
TMPTRoot=TMPT.insert(TMPTRoot,tx.key,tx);        //将tx插入交易树TMPT中
}          //块内TMPT索引构建完毕
//构建全局KMPT索引
KMPTRoot=Blockchain.getCurrentBlock().KMPTRoot;    //获取当前最新块KMPT根哈希
for(Transaction tx: transactionMap){ //遍历交易集合
KMPTRoot=KMPT.newBranch(KMPTRoot,tx.Key,TMPTRoot);//将Key—TMPTRoot插入KMPT中
}
Block.TMPTRoot=TMPTRoot;                    //将TMPT根哈希添加到区块头中
Block.KMPTRoot=KMPTRoot;   //将KMPT根哈希添加到区块头中
return Block;
End.
```

算法1构建块内TMPT索引。首先初始化TMPT根，遍历交易集合，将交易插入到块内TMPT中，并记录每次插入后的最新TMPTRoot，循环结束，此时块内TMPT索引构建完毕。

构建全局KMPT索引。首先从当前系统中最新块获取最新KMPTRoot，遍历交易集合，以当前KMPTRoot、当前交易Key值及已经构建好的本块TMPT根哈希为参数，将该Key–TMPTRoot对以新建分支的方式插入到当前KMPT中，记录插入后的KMPT根哈希。

最后循环结束，KMPT索引构建完毕，将构建完成后的最新TMPT及KMPT根哈希添加到区块头中，将区块返回，即完成一个区块的索引构建过程。

1.2.3 检索过程

针对Key值进行的检索，分为2个检索过程。首先根据最新块KMPT根哈希到当前KMPT全局状态树中检索到目标Key对应状态叶节点，从该叶节点中获取到目标Key最新版本所在区块TMPT根哈希，由此TMPTRoot开始进行块内检索。在TMPT和KMPT中的检索过程均与前缀树的匹配过程相同，不再赘述。需注意的是，检索过程中需保存检索路径，以提供Merkle证明路径，相应的Key记录最新版本检索算法如下。

算法2. Key记录最新版本检索算法

输入：Key

输出：<transaction,path> Key对应数据记录最新版本及验证路径

```
Begin:
Stack path;                    //用栈保存Merkle验证路径
Block=Blockchain.getCurrentBlock(); //获取当前最新块
KMPTRoot=Block.KMPTRoot;                    //获取最新块中KMPT根哈希
```

```
Leafnode=KMPT.Find(Key,KMPTRoot,path);　　//在KMPT中检索Key对应叶节点
if(Leafnode==null)　　//若Key对应记录不存在
return <null,path>;　　//返回空的记录及KMPT检索路径
else{　　//若Key记录存在
target_TMPTRoot=Leafnode.Latest TMPTRoot;　　//从该叶节点中获取到Key记录最新版本所在区块的TMPTRoot
path.clear();　　//将栈path清空
transaction=TMPT.Find(Key,target_TMPTRoot,path);　　//块内检索目标Key记录
return <transaction,path>;　　//返回目标Key记录和块内TMPT检索路径
}
End.
```

算法 2 创建一个用于保存检索路径的栈 path,以进行 Merkle 存在性和不存在性证明。获取系统当前最新块,并获取最新块中保存的当前 KMPT 根哈希,由此根哈希在 KMPT 全局状态索引树中检索到目标 Key 状态节点 Keynode,并将在 KMPT 中的检索路径保存在 path 中。若该 Key 记录不存在,即 KMPT 中无该 Key 对应的状态节点,此时将返回一个空的交易记录和在 KMPT 中的检索路径 path,此 path 可用于对该 Key 记录的不存在性证明。若该 Key 记录存在,从在 KMPT 中查询到的状态节点中获取该 Key 最新版本所在区块中的交易记录 TMPT 根哈希,由于此时 Key 存在,则不再需要 path 中保存的用于不存在证明的 KMPT 检索路径,随后将 path 清空,然后根据获取到的 TMPT 根哈希在目标区块中检索目标 Key 记录,同时将块内检索路径保存在 path 中,用于该 Key 的最新版本存在性的证明。将检索到的目标 Key 记录和块内检索路径返回。

基于全局 Key 记录状态索引模型,检索目标 Key 对应数据记录时,可根据最新块 KMPTRoot 在 KMPT 树中检索到目标 Key 记录最新版本所在区块交易树 TMPT 根哈希,直接开始块内检索,避免了遍历区块检索方式带来的大量无效搜索,节省了查询过程的时间消耗。

此外,在索引模型中,若 Key 对应数据记录存在,块内 TMPT 检索的过程即是提供 Key 最新版本记录 Merkle 存在性证明路径的过程;若 Key 不存在,则在 KMPT 中的检索过程即是提供 Key 记录不存在证明路径的过程。因此,可提供基于 Key 值查询的数据存在性和不存在性的证明。

针对查询最新版本的存在性证明与其他区块链系统中的 Merkle 存在性证明类似,这里仅针对不存在性证明进行说明。如图 4 所示,KMPT 中存储有 Key 为 a711355、a77d337、a7f9365、a77d397 的 4 个 Key 记录状态信息,对某个不存在的 Key 为 a77d367 的记录查询,需提供其不存在证明。由前缀树的特性可知,某 Key 记录在树中的位置仅由 Key 值决定,即若该 Key 为 a77d367 的记录存在,则必存在于图中虚线节点处,为证明其不存在,则将该虚线节点所在分支发给轻节点验证即可。

如算法 2 所述,在对该 Key 的 KMPT 检索过程中,将其检索路径保存于 path={P1,P2,P3,P4}中,轻节点接收到该分支路径 path 后,首先计算 P1 节点哈希,并将其与自身存储最新区块头中 KMPTRoot 对比,一致则 P1 有效,开始对该 Key 为 a77d367 的记录进行前缀匹配以检验分支,为防止分支遭到篡改,在匹配过程中需验证其有效性,即匹配 P2 之前先计算其哈希值,将其与 P1 中 Next node 域中的哈希值比较,一致则 P2 有效,对 P2 进行匹配;用同样的方法验证 P3 和 P4;验证 P4 有效,即说明该分支有效,对 P4 匹配失败,即证明 Key 为 a77d367 的记录确实不存在,完成不存在性证明的过程。

数据可溯源是区块链的重要特性,在不同场景中区块链数据溯源的含义有所区别:在数字资产领域,数据溯源一般指追溯出某笔数字资产的历史流转过程,即价值流转轨迹;在本文所属的区块链数据库领域,数据溯源是指追溯出某数据记录的所有历史版本,即数据修改轨迹。基于本文索引模型可实现对 Key 记录的所有历史修改版本进行溯源查询,具体溯源查询算法如下。

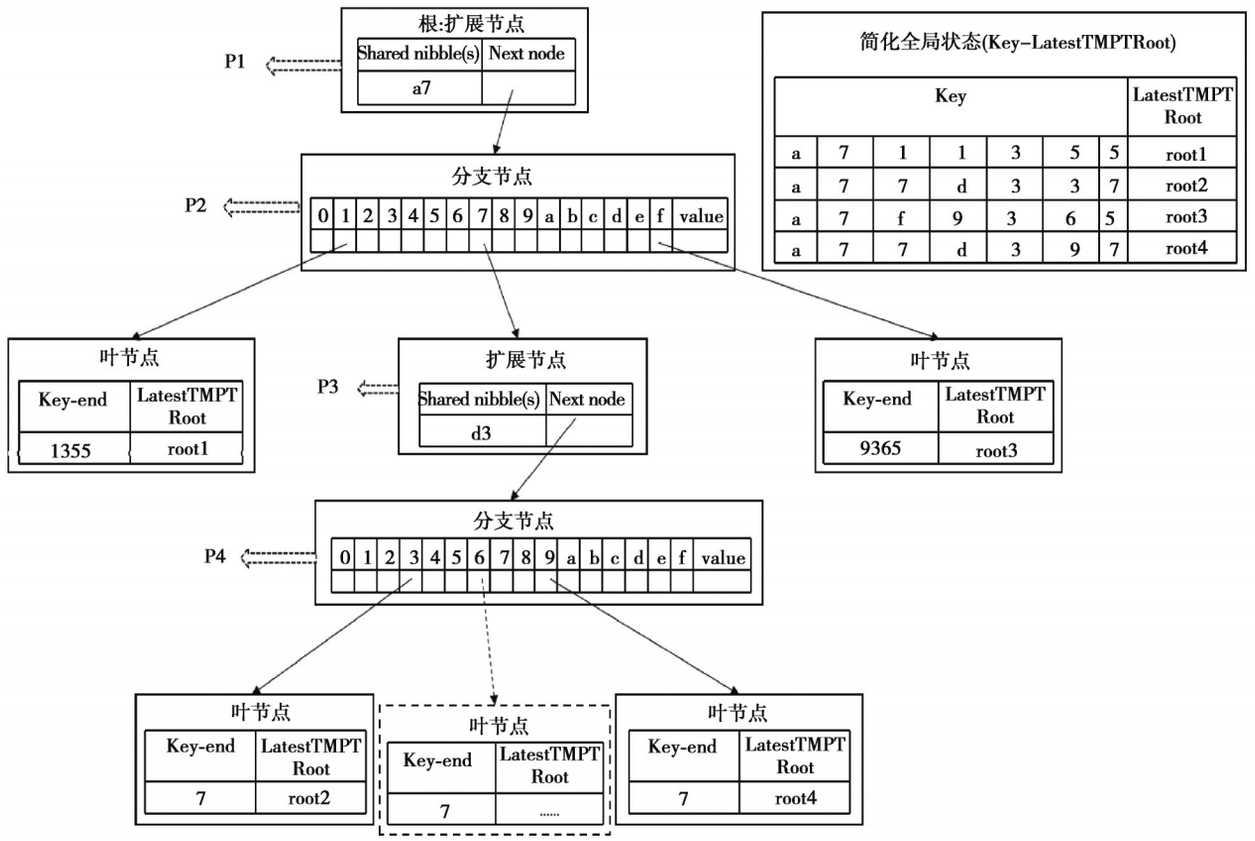

图 4　不存在证明路径

Fig. 4　The path of proof of non-membership

算法 3. Key 记录溯源查询算法：

输入：Key

输出：Key 记录所有历史版本 transactionList

```
Begin:
List transactionList;              //创建 transactionList 保存查询结果
<transaction,path>=Blockchain.Find(Key);        //查询到 Key 记录最新版本
if(transaction==null)              //Key 对应记录不存在
return null;    //返回空值结束程序
else{
transactionList.add(transaction);          //将 Key 最新版加入 transactionList
temp=transaction.Prehash;          //获取上一版本记录哈希值
While(temp!=null){
transaction=LevelDB.get(temp);   //从 LevelDB 中读取上一版本记录具体信息
transactionList.add(transaction); //将读取出的记录加入 transactionList
temp=transaction.Prehash;          //继续获取上一版本记录哈希值
}
}
return transactionList;    //最后返回 Key 记录所有历史版本 transactionList
End.
```

算法 3 创建用于保存溯源查询结果的交易数据记录列表,在系统中查询 Key 对应的最新版本记录,若查询 Key 对应记录不存在,则返回空值并结束程序,若查询 Key 记录存在,首先将查询到的 Key 最新版本存入 transactionList 中,再获取上一版本的记录哈希值,若哈希值不为空值,则从 LevelDB 中取出上一版本交易记录,存入 transactionList,继续获取上一版本记录哈希,重复此过程,直到上一版本记录哈希为空值,即完成所有历史版本的溯源查询,最后返回保存的记录列表,结束。

1.2.4　检索效率分析

区块链由一个个区块根据哈希指针链接形成,本质上属于单链表结构,假设区块链中有 N 个区块,每个区块中包含 M 笔数据记录。传统基于 Merkle 树结构的区块链由于没有提供高效的检索方法,查询某一特定 Key 值记录需遍历搜索所有区块和区块中的数据,这一过程时间复杂度 T_1 为

$$T_1 = O(N \cdot M)。$$

而块内索引方法由于在区块内基于 Merkle 树构建关键字索引,将在块内的检索过程时间复杂度降到了对数级别,但由于无法避免遍历区块过程,故其时间复杂度 T_2 为

$$T_2 = O(N \cdot \log M)。$$

索引方法引入了基于状态树的 KMPT 全局区块索引结构,避免了遍历区块的检索过程,只需在最新 KMPT 树中检索到目标区块后,直接在目标区块 TMPT 树中进行块内检索,将 2 个过程的时间复杂度都降到了对数级别,即索引模型的检索时间复杂度 T_3 为

$$T_3 = O(\log NM)。$$

显然,

$$T_3 < T_2 < T_1。$$

即索引模型的检索时间复杂度在理论上要明显低于传统基于 Merkle 树方法和块内索引优化方法,具有更高的检索效率。

2　实　验

实验采用的硬件环境为 Intel(R) Core(TM) i5-7300HQ CPU(2.50 GHz),RAM(8 GB),操作系统 Windows 10,参考 Bitcoin 开源代码 v0.1.0 和以太坊开源代码 Geth v1.8.0,使用 Java 语言构建实验代码,底层数据库使用 LevelDB,主要针对交易格式、交易块内组织形式、状态树结构等底层数据结构进行修改,实现本文索引模型。同时,将 Blockchain DB 模型中的 Merkle RB tree 方法作为块内索引方法的代表,以及 Merkle tree 方法作为传统区块链系统的代表,与提出方法进行对比实验。由于研究属于区块链全节点中的查询方法,不涉及网络多节点间共识,所以在单机上进行实验,并将区块数据记录有效性验证过程视为共识的过程。实验中的主要指标为算法运行时间,并将算法独立运行 50 次的平均值定义为算法的运行时间。实验数据固定 2 个数据字段,分别为 Key 和 Field1 两个域,其中 Key 为数据的关键字,每条数据记录的 Field1 域设置值为该记录所在区块的区块号。实验使用的数据集为自采数据集,公开在 https://gitee.com/huang_li_huang/blockchain_query.git 中。

2.1　区块深度对于查询时间的影响

实验 1.数据存储时以区块为单位按照时间顺序添加到链上,再持久化存储到 LevelDB 中。对某一关键字为 Key 的记录查询需返回其最新版本。当系统中最新区块的块高度为 h_1,目标数据最新版本所在区块高度为 h_2 时,定义区块深度为 $h = h_1 - h_2$。本实验的目的是探究目标数据最新版本所在区块深度对查询时间的影响情况。实验中顺序写入 100 万条 Key 升序(Key 为 0~999 999)的数据记录,每个区块设置存储 1 000 条数据记录,总共 1 000 个区块,当查询目标交易所在区块深度为 h = 100,200,300,...,1 000 时,对比 Merkle tree、Merkle RB tree 及本文索引方法所需查询时间,实验结果见图 5。

由实验结果可知,Merkle tree 方法和 Merkle RB tree 方法的查询时间随目标 Key 记录最新版本所在区块深度呈线性增加的趋势,这是由于 Merkle tree 并没有提供高效的查询方法,检索需从最新区块开始遍历,在区块内部同样也采用遍历交易列表的方式,检索时间复杂度为 $O(N \cdot M)$,当目标 Key 记录所在区块深度较

深时，需在最新块和目标块之间的区块中做大量的无效检索，查询时间随区块深度线性增长；而 Merkle RB tree 由于使用红黑树优化了块内检索方式，在块内不需遍历交易列表查询，将块内检索的时间从线性降到了对数级别，其检索时间复杂度为 $O(N \cdot \log M)$，比 Merkle tree 方法的检索效率更高。但由于无法确定目标数据所在区块，检索还需采用遍历区块方式进行，且遍历区块过程不断进行磁盘读写，大大限制了检索的效率，使查询时间仍随区块深度线性增加；而基于提出的索引模型 KMPT，可从最新块中的 KMPT 树根开始，检索到目标 Key 记录所在区块中的 TMPT 根，实现了定位目标块的效果，避开了遍历区块不断读写磁盘的检索过程，同时在块内使用 TMPT 结构加速块内检索速度，无需遍历交易列表，将时间复杂度降到 $O(\log NM)$，因此，查询基本不受区块深度的影响，可在很短的时间内稳定完成检索任务。

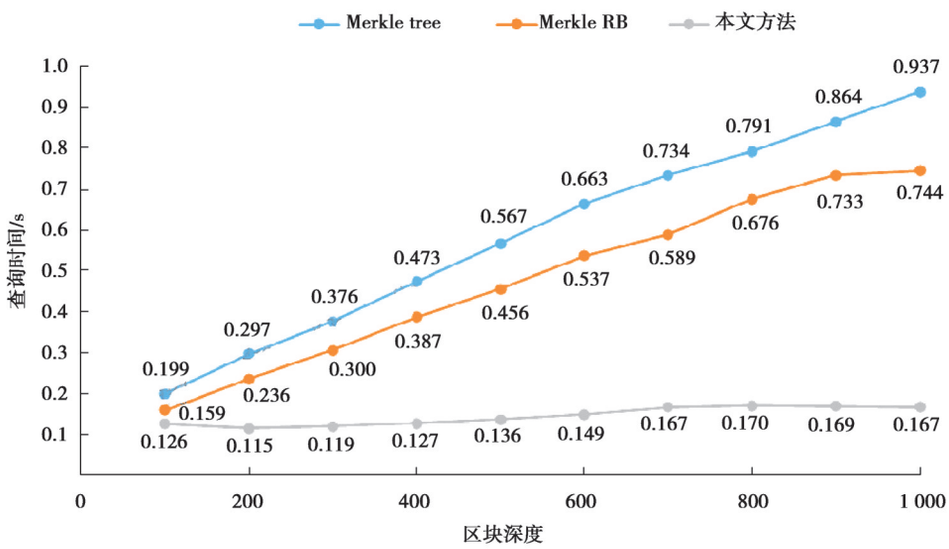

图 5 区块深度对查询时间影响

Fig. 5 Effect of block depth on query time

2.2 目标 Key 不存在情况下查询响应时间与区块数的关系

实验 2. 本实验的目的是探究极端情况下，查询目标 Key 不存在时 3 种方法的查询响应时间。同实验 1，顺序写入 Key 升序数据记录，每个区块设置存储 1 000 笔数据记录，当系统中总区块数为 100（Key 为 0~99 999）、200（Key 为 0~199 999）、300（Key 为 0~299 999）……1 000（Key 为 0~999 999）时，输入一个不存在的目标 Key 值，对比测试 3 种方法查询响应时间。实验结果如图 6 所示。

由于查询目标 Key 值不存在，Merkle tree 和 Merkle RB tree 方法需遍历检索完全部区块数据才可返回不存在信息，与实验 1 类似，两者查询响应时间都随系统区块数线性增长，其中 Merkle RB tree 方法由于加快了块内检索的速度，将检索时间从 Merkle tree 方法的 $O(N \cdot M)$ 降到了 $O(N \cdot \log M)$，所以响应时间比 Merkle tree 方法更短，但受制于其遍历区块检索时需不断进行磁盘读写，导致这一优化结果并不明显。不同于前两者，基于本文索引模型的方法当查询目标 Key 值不存在时，只需根据最新块 KMPTRoot 到当前最新 KMPT 树中检索，不需遍历检索区块数据，即可返回不存在响应及不存在证明路径，且这一过程不需进行块内检索，时间开销比检索数据存在时的 $O(\log NM)$ 更低，因此在很短时间内便可返回查询不存在响应，且响应时间受区块数的影响微乎其微，在提升了查询不存在可信性的同时具有更高效更稳定的表现。

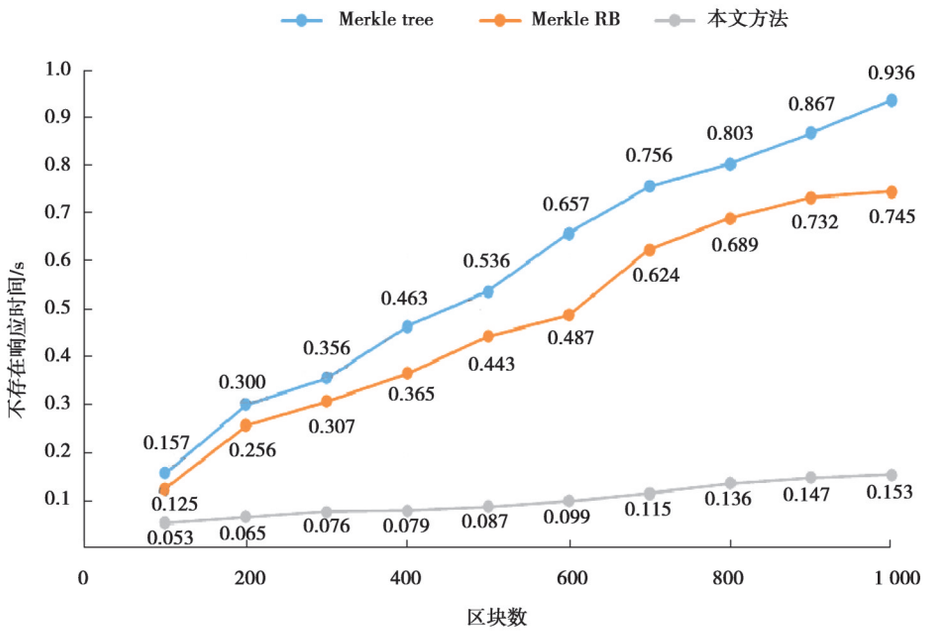

图 6　查询不存在响应时间与区块数关系

Fig. 6　Relationship between response of non-existent time and number of blocks

2.3　数据溯源查询效率对比

实验 3. 在 Blockchain DB 模型中,数据溯源指查询出某特定 Key 记录所有历史版本,以追溯出该 Key 记录数据的修改轨迹。溯源查询需先查询到目标 Key 最新版本,再由最新版本中 Prehash 依次追溯出该 Key 的所有历史版本,即分为最新版本查询和追溯历史版本 2 个过程,因此溯源查询时间与最新版本的查询时间相关。在 Merkle RB tree 方法中,最新版本的查询时间受目标数据所在区块深度影响,为对比 Merkle RB tree 方法与本文 KMPT 方法在追溯历史版本上的效率,实验中首先设置每个区块中都写入 Key 为 0~999 标识的数据,每条数据 Field1 域中的值为该数据所在区块号,以此来模拟数据版本的更新,对比测试历史版本数分别为 10,20,30,...,100 时,Merkle RB tree 方法与提出的索引模型方法溯源查询时间。实验结果如图 7 所示。

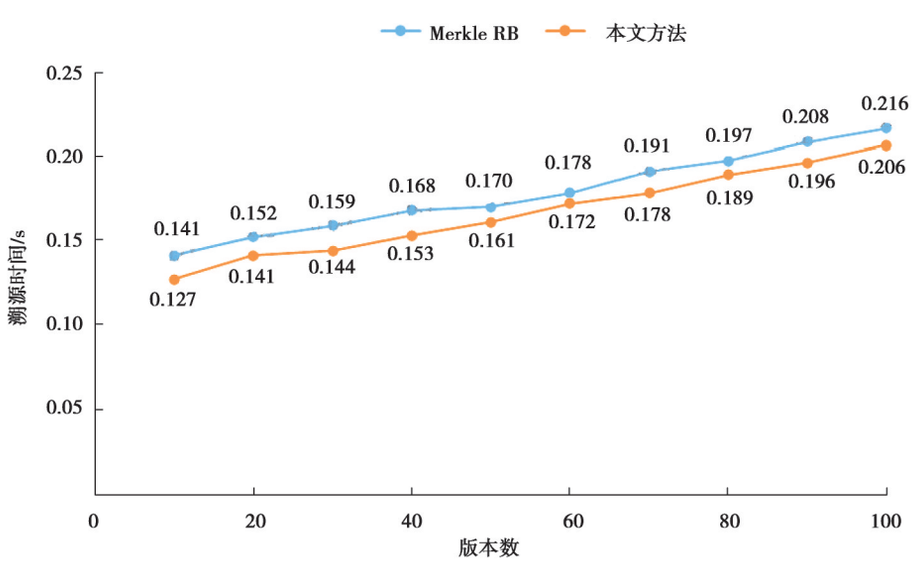

图 7　历史版本数对溯源查询效率影响

Fig. 7　Effect of historical versions on traceability query efficiency

由图7可知,Merkle RB tree方法与KMPT方法溯源查询时间主要集中于最新版本的查询时间,由于两者都通过数据记录中的Prehash追溯历史版本,且这一过程的时间消耗都由底层数据库Level DB决定,因此具有相近的历史版本追溯效率,此外可知历史版本数对溯源查询的时间影响较小。

由于溯源查询受最新版本查询时间影响,且历史版本数对溯源查询的时间影响较小,实验继续探究最新版本所在区块深度对溯源查询时间的影响。首先为每个Key记录设置100个历史版本,测试当其最新版本所在区块深度为100,200,300...1 000时,对比Merkle RB tree方法与本文方法溯源查询的效率。

实验结果如图8所示,由于两者溯源查询时间主要集中于最新版本的查询时间,由实验1可知,Merkle RB tree方法最新版本查询时间随区块深度线性增加,而本文方法最新版本的查询基本不受区块深度影响,故当最新版本所在区块深度增加时,Merkle RB tree方法溯源查询总时间线性增加,而KMPT方法溯源查询仍不受影响,查询效率远高于Merkle RB tree方法,即本文索引模型由于提高了最新版本的查询效率,且查询时间与最新版本所在区块深度无关,相比块内Merkle RB tree索引方法而言,有更高更稳定的溯源查询效率。

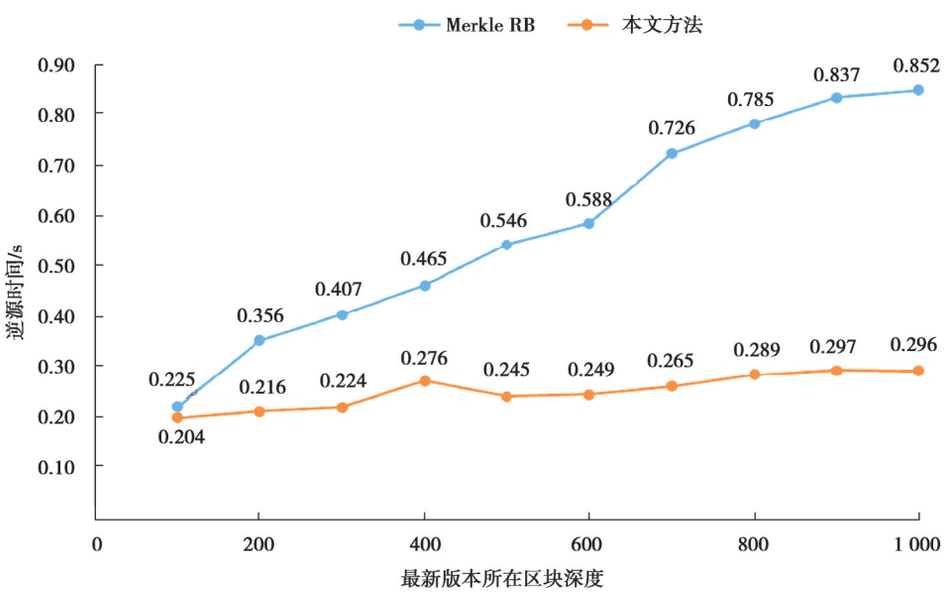

图8 最新版本区块深度对溯源查询效率影响

Fig. 8 Effect of the latest version block depth on the efficiency of traceability query

2.2.4 索引构建代价对比

实验4.索引的更新以区块为单位,区块内包含的数据数量决定了单块索引的构建时间,实验中通过向区块链数据库中写入不同大小的块,即包含1 000,2 000,3 000,...,8 000条数据的区块,探究对比在不同数据数量下Merkle RB tree方法与本文索引构建的时间代价情况。实验结果如图9所示。

本文索引方法与Merkle RB tree方法单区块索引构建时间都随区块数据量增大而增加,且本文方法的构建时间比Merkle RB tree方法更长,这是由于本文索引模型在构建块内索引的基础上增加了全局KMPT索引结构,故相比于单纯的块内索引模型Merkle RB tree来说,构建索引的时间开销有所增加。但与查询效率、查询可信性及稳定性的极大提升相比,单块索引构建的时间代价付出维持在毫秒级别,没有为系统带来过大的额外负担。即本文索引模型在可接受的代价范围内,实现了查询效率、查询稳定性和查询可信性的极大提升。

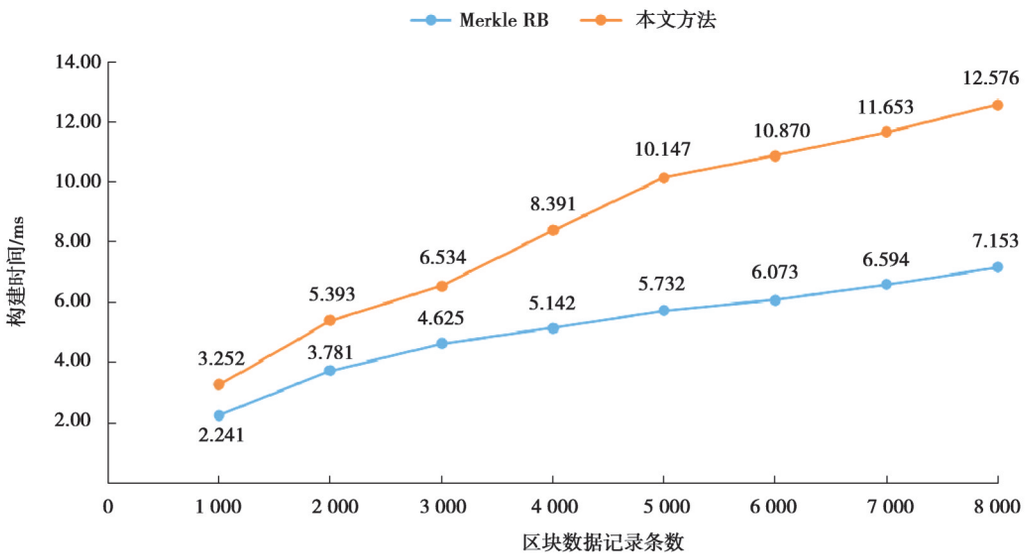

图 9　索引构建代价对比

Fig. 9　Comparison of index construction cost

3　结束语

目前在区块链内置索引查询优化方法中,由于无法确定目标数据所在区块,检索需遍历区块进行,导致查询效率低下。针对这一问题,在 Blockchain DB 数据模型的基础上,基于以太坊状态树结构提出一种全局状态索引模型,在保证索引及数据不可篡改的前提下,实现根据检索 Key 值直接定位目标数据所在区块,避免了遍历区块检索的过程,大大提升了最新版本记录的检索时间,且查询可同时提供数据存在和不存在性的证明,提升了查询可信性。最后,通过实验测试,与 Blockchain DB 模型中以 Merkle RB tree 为代表的块内检索方法相比,本文索引模型在数据最新版本查询、数据不存在响应、数据溯源查询上具有更高更稳定的查询效率。

区块链数据库致力于在不互信的多节点环境中建立可信的数据存储环境,这对数据存储安全有重大意义。在确保数据安全、不可篡改、可信性等特性不受损失的前提下,研究如何提升区块链数据库的读写性能,是发挥其重大应用价值的关键。目前对区块链数据写入(共识算法)的研究已取得很多不错的进展,然而在对其读性能即查询检索方面的研究则相对较少,有关区块链数据查询还需很多后续工作,如研究如何更高效地实现更多的查询功能以丰富查询语义,如范围查询、Top-K 查询等。此外,研究新型区块链数据结构如 DAG 下的高效查询,在未来也具有重要的应用价值。

参考文献

[1] Nakamoto S. Bitcoin: a peer-to-peer electronic cash system [EB/OL]. [2021-05-12]. https://www. bitcoinpaper. info/bitcoinpaper-html/.

[2] Buterin V. Ethereum: a next-generation smart contract and decentralized application platform[EB/OL]. [2021-05-12]. https://courses.cs.duke.edu/spring23/compsci512/papers/ethereum.pdf.

[3] 袁勇, 王飞跃. 区块链技术发展现状与展望[J]. 自动化学报, 2016, 42(4): 481-494.
Yuan Y, Wang F. Blockchain: the state of the art and future trends[J]. Acta Automatica Sinica, 2016, 42(4): 481-494. (in Chinese)

[4] Bamakan S M H, Motavali A, Bondarti A B. A survey of blockchain consensus algorithms performance evaluation criteria[J]. Expert Systems With Applications, 2020, 154: 113385.

[5] Tikhomirov S. Ethereum: state of knowledge and research perspectives[C]//International Symposium on Foundations and Practice of Security. Cham: Springer, 2018: 206-221.

[6] Wood G. Ethereum: a secure decentralised generalised transaction ledger[EB/OL]. [2021-05-12]. https://www. win. tue. nl/

~mholende/seminar/references/ethereum_yellowpaper.pdf.

[7] 于戈, 聂铁铮, 李晓华, 等. 区块链系统中的分布式数据管理技术: 挑战与展望[J]. 计算机学报, 2021, 44(1): 28-54.
Yu G, Nie T Z, Li X H, et al. The challenge and prospect of distributed data management techniques in blockchain systems[J]. Chinese Journal of Computers, 2021, 44(1): 28-54. (in Chinese)

[8] 王千阁, 何蒲, 聂铁铮, 等. 区块链系统的数据存储与查询技术综述[J]. 计算机科学, 2018, 45(12): 12-18.
Wang Q G, He P, Nie T Z, et al. Survey of data storage and query techniques in blockchain systems[J]. Computer Science, 2018, 45(12): 12-18. (in Chinese)

[9] Li Y, Zheng K, Yan Y, et al. EtherQL: a query layer for blockchain system[C]//International Conference on Database Systems for Advanced Applications. Cham: Springer, 2017: 556-567.

[10] Muzammal M, Qu Q, Nasrulin B, et al. A blockchain database application platform[EB/OL]. 2018 [2021-05-12]. https://arxiv.org/abs/1808.05199.

[11] Peng Y Q, Du M, Li F F, et al. FalconDB: blockchain-based collaborative database[C]//Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data, June 14-19, 2020, Portland, OR, USA. New York: ACM, 2020: 637-652.

[12] 焦通, 申德荣, 聂铁铮, 等. 区块链数据库: 一种可查询且防篡改的数据库[J]. 软件学报, 2019, 30(9): 2671-2685.
Jiao T, Shen D R, Nie T Z, et al. BlockchainDB: querable and immutable database[J]. Journal of Software, 2019, 30(9): 2671-2685. (in Chinese)

[13] 贾大宇, 信俊昌, 王之琼, 等. 存储容量可扩展区块链系统的高效查询模型[J]. 软件学报, 2019, 30(9): 2655-2670.
Jia D Y, Xin J C, Wang Z Q, et al. Efficient query model for storage capacity scalable blockchain system[J]. Journal of Software, 2019, 30(9): 2655-2670. (in Chinese)

[14] Zhang C, Xu C, Xu J L, et al. GEM^2-tree: a gas-efficient structure for authenticated range queries in blockchain[C]//2019 IEEE 35th International Conference on Data Engineering (ICDE), April 8-11, 2019, Macao, China. IEEE, 2019: 842-853.

[15] 郑浩瀚, 申德荣, 聂铁铮, 等. 面向混合索引的区块链系统的可查询性优化[J]. 计算机科学, 2020, 47(10): 301-308.
Zheng H H, Shen D R, Nie T Z, et al. Queryability optimization of blockchain system for hybrid index[J]. Computer Science, 2020, 47(10): 301-308. (in Chinese)

[16] Abdennebi A, Kaya K. A bloom filter survey: variants for different domain applications[EB/OL]. 2021-06-23 [2021-07-12]. https://arxiv.org/abs/2106.12189.

（编辑 罗 敏）