专 业 学 位 硕 士 学 位 论 文

区块链数据内容检索与溯源查询优化研究

Research on Content Retrieval and Traceability Query Optimization of Blockchain Data

作 者 姓 名：黄笠煌

工 程 领 域：软件工程

学　　号：32017016

指 导 教 师：

完 成 日 期：

大连理工大学

Dalian University of Technology

大连理工大学学位论文独创性声明

作者郑重声明：所呈交的学位论文，是本人在导师的指导下进行研究工作所取得的成果。尽我所知，除文中已经注明引用内容和致谢的地方外，本论文不包含其他个人或集体已经发表的研究成果，也不包含其他已申请学位或其他用途使用过的成果。与我一同工作的同志对本研究所做的贡献均已在论文中做了明确的说明并表示了谢意。

若有不实之处，本人愿意承担相关法律责任。

学位论文题目：_____

作 者 签 名 ：_____ 日期：_____年____月____日

摘　要

数据内容检索和溯源查询处理是区块链的两个重要数据读取方式，但两者都存在着效率上的问题，亟需得到优化。在数据内容检索方面，现有的研究方法难以避免采用遍历区块的搜索方式，导致检索效率没有得到根本性的改善。在数据溯源查询方面，为减轻溯源请求频繁场景中底层存储系统的压力，提高溯源效率，现有研究普遍通过引入复杂数据结构的方式进行优化，给全节点带来了过大的维护负担，难以兼顾溯源效率与资源消耗。因此，在分析现有区块链全节点中内容检索和溯源查询处理方式后，本文主要完成以下两方面的研究工作：

首先，针对内容检索效率低下的问题，本文提出了一种基于以太坊状态树的索引优化模型，使得检索可精准定位目标数据所在区块，避免了遍历区块的搜索过程，内容检索的效率和稳定性均得到了很大改善，实现了基于关键字 Key 的高效区块链数据内容检索。此外，受益于状态树索引的全局性特征，该模型可同时提供查询数据存在或不存在证明，进一步提升了查询结果的可信性。

其次，针对在溯源查询请求频繁的场景中，难以兼顾全节点溯源查询效率与资源消耗的问题。本文设计了基于多级缓存的溯源查询优化方案，用高效的内存操作替代了重复低效的磁盘 IO 溯源检索，提升了溯源查询的效率，同时该多级缓存结构降低了全节点维护缓存的内存开销，兼顾溯源效率与资源消耗，减轻了全节点负担。

最后，经实验验证，本文所提针对全节点中内容检索和溯源查询的效率优化方案是有效的。此外，我们将以上两个理论研究成果集成到真实应用项目“中国图片区块链知识产权保护平台”中，其表现进一步验证了本文优化方案的有效性。

关键词：区块链；区块链数据库；数据内容检索；溯源查询

Research on Content Retrieval and Traceability Query Optimization of Blockchain Data

Abstract

Data content retrieval and traceability query processing are two important data reading methods of blockchain, but both have inefficiency shortcomings. In terms of data content retrieval, the existing research methods cannot avoid the search method of traversing blocks, resulting in no fundamental improvement in retrieval efficiency. In terms of data traceability query, in order to improve the traceability efficiency in scenarios with frequent traceability requests, existing research generally optimizes by introducing complex data structures, which brings excessive maintenance burden to the whole node. Therefore, this paper mainly completes the following two aspects of research work:

Firstly, aiming at the problem of inefficient content retrieval, this paper proposes an index optimization model based on Ethereum state tree, which can accurately locate the block where the target data is located, avoid the search process of traversing the block, and greatly improve the efficiency and stability of content retrieval. In addition, benefiting from the global characteristics of the state tree index, the model can provide proof of the existence or absence of query data at the same time, which further improves the credibility of query results.

Secondly, in scenarios where traceability query requests are frequent, it is difficult to balance the efficiency of full-node traceability query with resource consumption. In this paper, a traceability query optimization scheme based on multi-level cache is designed, which replaces the repeated and inefficient disk IO traceability retrieval with efficient memory operations, improves the efficiency of traceability queries, and reduces the memory overhead of the whole node to maintain the cache, taking into account the traceability efficiency and resource consumption, and reducing the burden of the whole node.

Finally, experiments verify that the efficiency optimization scheme proposed in this paper for content retrieval and traceability query in the whole node is effective. In addition, we integrate the above two theoretical research results into the real-world application project "China Image Blockchain Intellectual Property Protection Platform", and their performance further verifies the effectiveness of the optimization scheme in this paper.

Key Words： Blockchain; Blockchain Database; Data Content Retrieval; Traceability Queries

目　　录

1 绪论

1.1 研究背景及意义

随着比特币[1]，以太币[2][3]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，许多具有代表性的区块链新产品如 Hyperledger Fabric[4]、Corda[5]等不断涌现。区块链起源于比特币系统，是支撑比特币运行的核心技术，实现了在无中介介入的情况下，互不信任的双方可以直接使用比特币进行点对点支付，即去中心化概念。从数据结构层面来说，区块链是将数据存储于区块之中，并按照时间顺序相连而形成的链式数据结构，并以密码学方式保证其不可篡改和不可伪造。相比于传统的集中式数据库，区块链存储数据具有去中心化、不可篡改、数据可追溯、查询可验证等特点，因此可以应用于多方不互信的应用场景，确保数据安全可信。

从业务上来说，在多方不互信场景中构建可信的数据存取环境，解决各类业务应用场景中数据信任问题，这正是区块链的真正价值所在，也是其能成为热点话题的根本原因。合理应用区块链技术构建信任价值网络和可信数据基石，将对各行业各领域的数字化业务进程产生质的改变。国家层面来说，习近平总书记强调：“把区块链作为核心技术自主创新的重要突破口”“加快推动区块链技术和产业创新发展”，体现了对区块链技术的高度重视。市场层面来说，如今区块链作为解决数据可信治理问题的代表方案，其应用领域已经从数字货币延伸到供应链管理[6][7]、物联网[8][9]、医疗健康[10][11][12]、版权保护[13][14]等诸多领域，体现了各领域在数据可信方面的广泛业务需求和对区块链技术的高度期望。

从技术上来说，区块链本质上是一个分布式的可信数据库，良好的读写性能是一个数据库系统所必须的，然而当前的区块链技术还处于发展之中，受制于其网络架构和特殊的数据组织方式，其数据读写效率还存在着明显的不足，这也是阻碍其进一步广泛应用的关键原因之一。为构建读写高效的区块链数据库系统，充分发挥其在应用场景中的业务价值，很多学者针对改善其数据读写效率展开了优化研究。

在写数据方面，区块链全节点需消耗大量的计算存储资源于分布式环境中的网络共识，才能完成数据的写入，即共识写入机制。为提升共识机制的写入效率，降低节点资源消耗，目前已有很多研究并且取得了不错的成效[15]。一些研究[16][17][18][19][20]针对原先工作量证明的共识方式进行了优化，避免了在挖矿竞争中的极大资源浪费，提升了共识写入的效率。为缩小共识参与节点的范围，降低网络共识通信的复杂度，以分块[21][22]，分组[23][24][25]等思想为代表的研究在共识效率，安全性等方面表现出了更出色的性能。另

外，针对各种不同应用场景的混合共识研究[26][27]进一步拓展了共识写入机制的应用范围和适应能力。近些年来，共识写入机制在写入效率以及安全性方面都取得了很大的研究进展。

而在读数据方面，即针对区块链数据查询处理方面的研究则相对较少。但对于构建高效的区块链数据库系统目标来说，优化其读性能同样是必不可少的研究课题。特别地，现有典型区块链系统在数据内容检索和数据溯源查询方面仍存在着很大的优化空间。

在数据内容检索方面，传统的区块链系统采用 Merkle 树结构组织块内数据，而 Merkle 树本身并无索引结构，导致在这类区块链中无法针对数据内容展开高效的检索。如比特币系统中仅支持根据交易哈希值检索交易内容，而无法针对交易的具体细节展开高效检索；以太坊支持账户查询，由于其引入了状态树维护全局账户状态，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数据并无关联，获取交易数据内容仍需通过交易哈希值进行检索。在这类区块链系统中，若要实现对数据内容的检索，普遍只能采用遍历区块并遍历区块内数据的方式，检索效率低下。

在溯源查询方面，溯源查询作为区块链实现数据可溯源，确保数据可信的重要特性功能之一，在如今许多场景中的应用需求不断增长。然而在当前的大部分区块链系统中，数据溯源都依靠数据中存储的上一版本数据哈希值，从底层存储系统中依次递归取出其所有历史版本，这种方法在溯源查询请求频率不高的场景中是合理且经济的。而在溯源查询请求频繁的场景中，当对同一数据的溯源查询请求存在大量重复，且数据历史版本较多，即溯源链路较长时，这种方法会向底层存储系统发送大量重复的查询请求，产生大量冗余的磁盘 IO 过程，溯源查询的响应速度受到影响，效率降低，同时给全节点带来了过重的负担，影响整个系统的吞吐效率。

综上所述，为实现高效的区块链数据读写目标，进一步发挥其在各领域中的应用价值，研究区块链数据查询处理技术必须的，其中，就研究区块链数据内容检索与溯源查询优化两个课题来说，更具迫切性和重要意义。

1.2 国内外研究现状

本节介绍当前区块链在数据内容检索和溯源查询处理两个方面的研究进展情况，总结了这些研究的特性及共同之处，最后分析并指出了现有方法存在的不足和下一步研究问题。

1.2.1 区块链内容检索研究现状

与传统意义上的数据库系统采用的关系型数据结构设计不同，为保证链上数据不可篡改和查询可验证性，为数据可信提供基础支撑，区块链数据存储结构具有特殊性，这决定了其数据检索方式要区别于传统的关系型数据库系统，且其数据存储，检索方案的设计优化必须保证查询数据不可篡改性和可验证性兼备，即须在数据可信性不受损失的前提下进行。针对目前区块链系统在数据内容检索方面普遍面临的效率低下问题，现有的优化研究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,如 EtherQL[28]。EtherQL 通过以太坊提供的监听接口将区块链中数据导入到链下数据库 MongDB[29]中，通过 MongoDB 提供数据分析查询服务。类似的还有 ChainSQL[30], Blockchain relational database[31]以及 FalconDB[32]。其中 ChainSQL 通过 Ripple[33]网络同步数据库操作, 将数据库操作回放到数据库中，通过区块链建立了一个不可篡改、低成本的多活数据库。Blockchain relational database 在 PostgreSQL[31]上实现了一个支持丰富查询、去中心化、防篡改的数据库。FalconDB 基于 MySQL[34]、IntegriDB[35]以及 Tendermint[36]实现了一个支持协作的数据库，完成了查询功能的丰富。

这类工作都通过借助传统数据库技术以实现丰富的查询功能，但这种将区块链数据转移到链下存储的方案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，最重要的是，没有从根本上解决区块链系统本身的数据检索问题。

另一类是内置索引方法，不同于外联数据库方法，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之中，以改善全节点数据检索的能力。内置索引方法并没有给系统带来额外的存储负担，致力于在保证数据不可篡改性的同时，丰富全节点查询功能并提升查询效率，从根本上提升了区块链系统的查询能力。如贾大宇等人，提出一种区块链存储容量可拓展模型的高效查询方法 ElasticQM[37]，并提出一种基于 B-M 树的区块存储结构组织块内交易，以加快块内内容检索的效率。类似地，在 Gem^2-tree[38]中，将传统 Merkle 树中叶子节点存储的交易哈希值用数据关键字代替，并结合 B+树设计了 MB 树索引，实现了基于数据关键字 Key 的查询。BCDB 模型[39]重新定义了区块链中的数据格式，将其从固定的交易数据结构推向更一般化的数据，以支持更多类型应用场景的数据存储需求。此外，提出了基于红黑树与 Merkle 树结合的块内索引结构 Merkle RB，在保证数据不可篡改的同时实现了基于数据内容的查询。然而这些研究本质上都是在对区块链单块块内索引结构进行优化，块内进行索引设计固然可以加快块内内容检索的速度，但由于缺乏全局性的索引结构，使得它们并没有从根本上

改变基于原始遍历区块的检索模式，即仍需从最新区块开始依次向前遍历区块检索目标数据，当目标数据所在区块深度较深时，这种检索方式由于在前面无关区块中进行了大量的无效搜索过程，大大降低了检索的效率。更糟糕的是，若目标数据不存在，需遍历到创世区块才能返回目标数据不存在信息，导致查询不存在时响应慢，且受块内索引的局部性所限，无法为查询提供数据不存在性证明，查询可信性不足。

值得一提的是，为加快遍历区块检索过程中无关区块的过滤速度，解决上述问题，郑浩瀚[40]等人对 MB 树索引进行了改进，提出一种混合块内索引结构，即在区块头中引入一个字段标识本区块存储交易连续属性值的范围，同时引入一个布隆过滤器[41]，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无关区块的过滤速度。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字段的方法虽加快了区块的过滤速度，但存在两个问题：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，遍历过程中不断的磁盘 IO 仍限制着检索的效率。

综上，现有研究工作在实现内容检索方面还存在着以下问题：

（1）内置索引方法能保证检索结果可验证，不损失数据可信性，但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，在无关区块中产生了大量无效搜索过程，没有从根本上改善内容检索的效率。

（2）当目标数据不存在时，仅使用块内索引优化的方法检索响应慢，且无法向轻节点提供数据不存在证明，查询结果正确性无法验证，查询可信性不足。

1.2.2 区块链溯源查询研究现状

在作为区块链重要特性功能之一的数据溯源查询方面。在比特币系统中，由于其本质属于交易系统范畴，数据溯源指的是以比特币实体为粒度的溯源。而在 BCDB 模型 [39]中，将交易数据推广到更具一般化的数据格式后，在数据字段中设置了 PreHash 字段，指向同 Key 标识的上一版本数据记录，以此实现针对关键字为 Key 数据的修改轨迹溯源。PreHash 中仅存储着同 Key 标识的上一版本历史数据的哈希值，真实数据还需通过该哈希值向底层磁盘存储系统中得到，这种溯源方法在面对大量溯源请求的场景时，将

检索压力转移给了磁盘存储系统，在一段时间内，当存在大量对同一 Key 的溯源请求时，将产生大量低效而冗余的磁盘操作，降低溯源的效率，难以应对复杂的溯源场景。

为进一步增强区块链系统的溯源能力，You[42] 等人基于以太坊状态树结构，提出了一种针对交易数据的可追溯混合索引机制，即通过账户事务外接一个哈希指针指向上一个事务所在的区块，可以快速检索账户交易跟踪链(account transaction trance chain，ATTC)，在一定程度上提高了溯源查询的效率，但是对于一些交易活动范围较广的事务，仍需在区块内部遍历，还存在着优化的空间。为改善这一情况，Xing 等人 [43]在 ATTC 的基础上提出了一种基于子链账户交易的索引结构，将链中存在的交易根据事务划分为多个子链，再将传统区块链中的逐块查找模式转换为逐子链查询模式，缩短了查询路径，然而该方法在构建索引过程中链接到交易的过程繁琐，索引冗杂，且子链模式下全节点维护代价较高。

此外，研究[44]中提出冷热区块的概念并基于跳表设计了一种 hot_skiplist 索引结构，可根据溯源查询频率对索引进行动态调整，提高了溯源查询的效率。文献[45]设计了 MB+树结构，缓存表及临时表，溯源查询时须先查缓存表和查临时表，最后再查询 MB+树索引，改善了溯源查询的响应速度。值得肯定的是，两者在提升溯源查询效率的同时，增强了溯源查询的动态适应性，但查询过程较为复杂，引入的数据结构也需占用全节点较大的存储空间。LineageChain[46]同样致力于实现区块链系统中的高效数据溯源查询。用户可以在特定的智能合约函数中指定状态数据的依赖关系,每次调用智能合约都会更新状态的依赖关系，并将依赖关系记录在一个无环图中。当执行溯源查询时，可以通过无环图的拓扑排序进行搜索。为了提升查询特定版本数据的效率，LineageChain 提出 DASL (Deterministic Append-only Skip List)用于在不同版本的数据上建立索引，加速了溯源查询速度。但值得权衡的是，LineageChain 使用了无环图来记录状态依赖关系，其结构较为复杂，同样给区块链全节点带来了过大的维护负担，抢占了全节点用于共识写入机制中的计算和存储资源，对整体区块链数据库系统的性能影响较大，在实用性上有所欠缺。

综上可知，目前针对区块链数据溯源查询的研究还存在着以下问题。

（1）由数据 PreHash 依次向前溯出完整的数据各历史版本的方法，在面对大量溯源需求的场景中，可能会产生大量冗余重复的磁盘读写过程，降低溯源查询的效率。

（2）全节点参与网络共识时需耗费大量计算存储资源，在对溯源检索过程进行优化时，引入复杂的数据结构会对全节点资源进行抢占，影响共识写入的效率。

1.3 本文主要工作

综上所述，现有区块链系统在内容检索和溯源查询处理上还存在着不足。在数据内容检索方面，现有方法检索效率低下；在溯源查询方面，现有优化方法难以兼顾效率与资源消耗。针对这两个问题，本文展开了以下研究工作：

（1） 针对内容检索效率低下的问题。本文采用内置索引的方式，在保证查询结果可验证，不损失区块链数据可信性的前提下，提出了一种基于以太坊状态树的索引优化模型，避免了内容检索中遍历区块检索方式所产生的大量无效搜索和磁盘 IO 过程，结合块内索引结构，从根本上提升了内容检索的检索效率和稳定性。此外，基于该索引模型可同时为查询提供数据存在和不存在证明，进一步提升了查询的可信性。

（2） 针对溯源查询难以兼顾效率与资源消耗的问题。本文为全节点引入溯源查询缓存，以提高数据溯源的效率，并设计了多级缓存的优化结构，缩减了缓存所占内存空间，减轻了全节点维护缓存的内存空间开销，使全节点能够节省出更多的计算存储资源投入到网络共识中去，兼顾溯源效率提升与资源消耗情况。

1.4 本文组织结构

本文的内容共分为五章，其组织结构如图 1.1 所示。

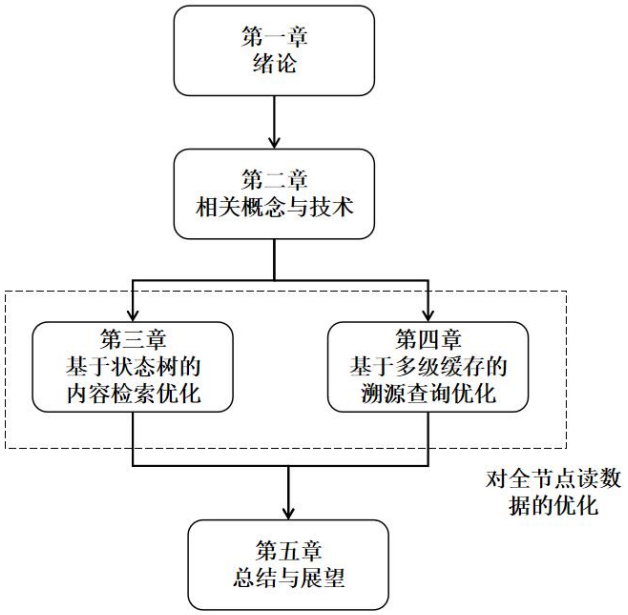

图 1.1 本文组织结构图

Fig. 1.1 The organization chart of this paper

章节安排分别为：

第一章，绪论。介绍了在区块链数据内容检索和溯源查询优化两个方面的相关研究背景及研究意义，国内外研究现状，并分析了当前存在的问题，最后简述了本文研究工作。

第二章，相关概念与技术。介绍了本文研究所涉及到的相关区块链基础概念和技术工作。

第三章，基于状态树的内容检索优化。针对区块链数据内容检索效率低下的问题。介绍了本文所提优化方案，即基于状态树的内容检索优化索引模型。对该索引模型结构，索引更新算法及内容检索算法进行了详细说明，并对该方案在内容检索上的效率进行了理论分析，最后设计实验验证方案的有效性。

第四章，基于多级缓存的溯源查询优化。针对在数据溯源请求频繁的场景中，如何在不给全节点带来过大负担的情况下，进一步提升区块链数据溯源查询的效率问题。介绍了本文所提优化方案，即基于多级缓存的溯源查询优化方案。对多级缓存结构，溯源查询流程和缓存更新方法进行了详细步骤说明，并对该优化方法中溯源查询的效率进行了理论分析，最后设计实验验证方案有效性。

第五章，总结与展望。对本文的研究成果及其应用情况等工作做了总结，并结合目前存在的不足，展望了区块链在数据查询检索优化方面的下一步研究方向。

2 相关概念与技术

上一章总结了本文的研究内容，即针对区块链数据的内容检索优化和溯源查询优化。本章首先介绍与之相关联的区块链基础概念，然后介绍相关技术工作，帮助进一步理解研究对象，并为后文介绍本文研究方案做下铺垫。

2.1 区块链基础

2.1.1 区块链概述

2008 年 11 月，化名为中本聪(Satoshi Nakamoto)的某学者发表了名为《比特币：一种点对点的电子现金系统》一文，提出一种被称为比特币的数字货币系统，由此区块链开始进入大众的视野。区块链是支撑比特币去中心化运行的核心技术，即在无中介介入的情况下，互不信任的双方可以直接使用比特币进行点对点支付。

在如今的互联网架构中，过于中心化的集中式架构带来的问题已日益严重[47]，如单点失效的网络安全问题对数据安全性构成威胁[48]。机构节点之间互相封闭，信息可信性和流通性受阻形成的信息孤岛问题等。为缓解这些问题耗费了大量的人力物力资源。而区块链这种去中心化的思想，正为解决这些问题提供了新的技术思路和启发，这也是其近些年区块链来备受各研究学者和业界关注的原因所在。

以数据治理角度[49][50][51]应用为例，现有的数据库普遍交由单一机构管理，数据的管理权掌握在中心机构手中，这种模式存在潜在的中心化安全问题：即用户必须无条件信任该机构，数据存储的安全性不得不和机构的信誉度捆绑在一起，这存在很大的潜在风险。更糟糕的是，若数据遭到第三方恶意入侵篡改，机构和用户都无法察觉，原因在于机构和用户对存储数据的可信性都缺乏自我验证机制。为了利用区块链技术解决中心化管理模式下数据可信性问题，很多学者对此展开了讨论[52][53][54][55][56]。其中利用区块链技术限制中心机构对数据的操作，将对数据的操作权交还给数据拥有者，即中心机构负责使用区块链数据库向用户提供数据存储服务，存储完整的区块链账本并运行共识算法对用户产生的数据进行写入，同时用户端轻节点存储区块头信息，完成对数据完整性及防篡改的验证，以此实现去中心化和防篡改的效果，这是一种有效的区块链可信数据治理应用思路。

区块链技术以其去中心化，多方共享，不可篡改，可验证以及可回溯等特性[57][58]为解决数据安全可信问题提供了新的可能，这也是其真正的价值所在。从本质上来说，区块链就是一个分布式的可信数据库，系统中的每个区块链全节点都拥有全量的区块链

账本。网络中产生的数据必须由多方经过共识协议确定才能写入，其网络基本架构如图2.1 所示。

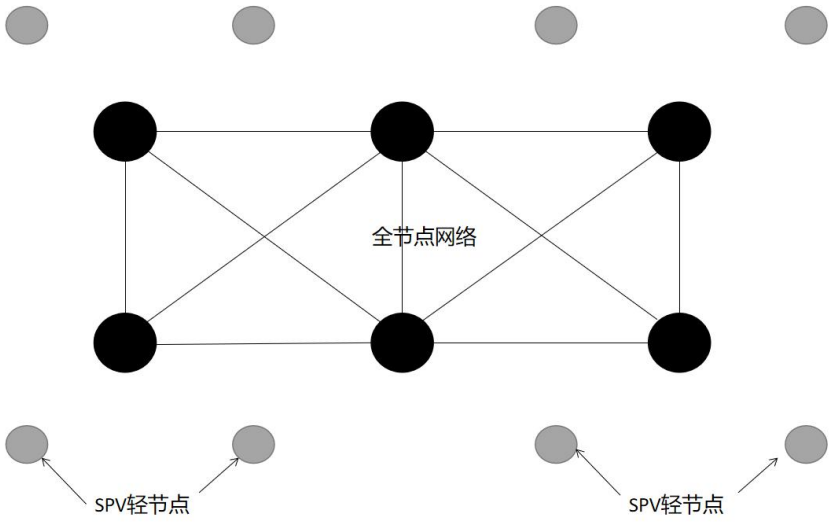

图 2.1 区块链网络架构图

Fig. 2.1 Blockchain network architecture diagram

在区块链网络中，每个全节点都存储全量的系统数据，用数据大量冗余备份存储解决单点失效的安全问题，数据多方共识写入机制来实现去中心化。此外，轻节点主要是一些存储计算资源有限的节点，在轻节点 SPV（Simplified Payment Verification）协议中，轻节点不需存储全量的区块链数据，只需存储区块头信息，以完成支付验证功能，即数据查询验证功能，保证数据可信性。这样的架构设计决定了区块链在效率上天然存在着短板，也决定了区块链只适合存储少量的关键数据，有价值的数据。以数据库视角来看，作为一个分布式可信数据库，解决区块链数据库的读写效率问题，是发挥其巨大潜在应用价值的关键。

（1）区块链数据结构

哈希指针是区块链中重要的基础数据结构，哈希指针借助 k-v 数据库实现，将数据哈希值作为 key，数据内容作为 value 存储于 k-v 数据库中，可以由数据哈希值从 k-v 数据库中取出数据并完成数据防篡改验证。其中数据哈希值既起到了数据哈希指针的作用，同时哈希指针又保证了分布式环境中节点数据地址的一致性和数据不可篡改性。

区块链是将数据存储于区块中，按照时间顺序通过哈希指针相连而形成的链式数据结构，通过网络节点共识机制，将数据以追加区块的方式写入其中，并以密码学方式保

证存储数据不可篡改和不可伪造。得益于其特殊的块内 Merkle 树数据组织结构，区块链可为数据查询方提供查询结果自我验证功能，这也是实现其数据可信特性的重要组成部分。此外，数据通过区块的追加方式进行写入，天然地为数据历史版本溯源提供了可能的基础和条件。相比于传统的集中式数据库，使用区块链构建的数据库具有去中心化、不可篡改、可追溯，查询可验证等特点，因此可以应用于多方不互信的应用场景，作为关键数据可信治理的角色。

如图 2.2 所示，区块链就是一个个区块根据哈希指针相连而成的数据结构（区块链中所有数据指针均为哈希指针）。每个区块由区块头和区块体构成，区块头中存储有指向前一区块的哈希值 Prehash，用于工作量证明共识的随机 Nonce 值，Merkle 树根哈希，区块产生时间戳和版本号等信息。由区块头中 Merkle 树根哈希所延伸出的 Merkle 树即为区块体，Merkle 树中交易数据存储于最底层叶子节点中，非叶子节点的值由叶子节点中交易数据计算哈希值而得，自底向上直至构造出 Merkle 树根。

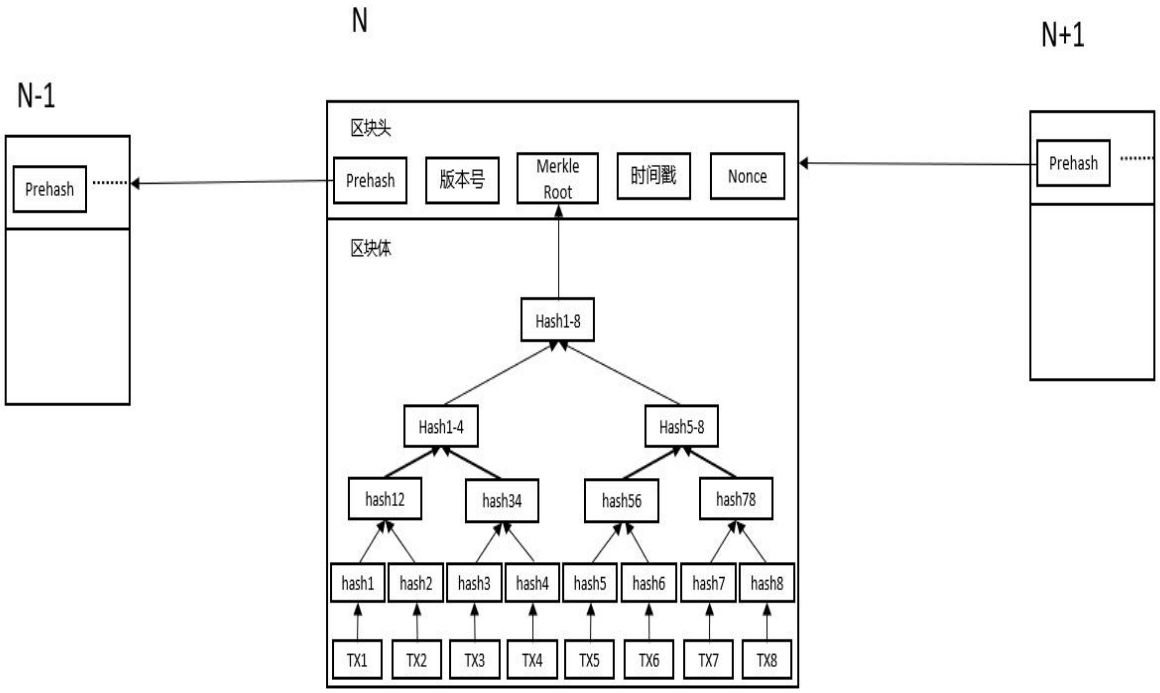

图 2.2 区块链结构图

Fig. 2.2 Blockchain structure diagram

在比特币系统中，轻节点存储了所有区块头，轻节点向全节点发送数据查询请求后，全节点在区块链账本中进行数据检索，并向轻节点提供目标数据及其 Merkle 验证路径，轻节点自身可通过此 Merkle 路径重新计算根哈希，将计算出的根哈希值与自身区块头中存储的 Merkle 根哈希对比，即完成数据存在性证明（Merkle Proof），即验证该数据确实存在于链上某个区块之中（Proof of Membership）。值得一提的是，Merkle Tree 由于其叶子节点的无序性及块内 Merkle 结构的局部性，只能完成数据存在性证明，而无法针对某一数据提供不存在证明（Proof of Non-membership），且 Merkle Tree 并无索引结构，无法提供高效的内容检索方法，导致内容检索只能采取依次遍历各区块中交易列表的形式进行搜索，效率低下。

（2）区块链系统分类

自比特币被提出以来，人们发现了其底层的区块链技术不仅能够应用于数字货币，还能应用在去中心化环境中实现可信的价值传输[59]。随后，出现了以以太坊为代表的一批实现数字资产交易的区块链系统应用。比特币、以太坊这一类系统, 允许所有人参加，即任何节点都可以参与区块链数据维护和读取，节点也可以随时加入或离开该网络，被称为公有链(Public Blockchain)。公有链允许节点自由进入、退出系统，难以适用于企业级的应用。对于企业级应用，通常只有企业联盟成员才能加入区块链网络，这种节点需要经过允许才能加入的系统通常被称为联盟链 (Consortium Blockchain),例如超级帐本[60]。公有链和联盟链之间的区别如表 2.1 所示。

表 2.1 公有链与联盟链对比表

Table 2.1 Comparison between public chain and alliance chain

对比项	公有链	联盟链
节点准入机制	节点自由加入退出	节点加入需得到许可
用户隐私	用户身份匿名	用户身份需审核
节点数量	无限制	有限
激励机制	代币机制	无激励机制
去中心化程度	完全去中心化	弱去中心化
共识机制	效率低、安全性较高的 PoW, PoS 等	效率适中、安全性适中的 PBFT, RAFT 等

（3）区块链系统逻辑层级

尽管各区块链系统的实现方式和分类各有不同，但是它们具有共同的系统逻辑层级。从数据管理的角度，给出区块链逻辑层级模型，如图 2.3 所示。

区块链系统逻辑上自底向上可划分为数据层、网络层、共识层、激励层、智能合约层、查询层和应用层。

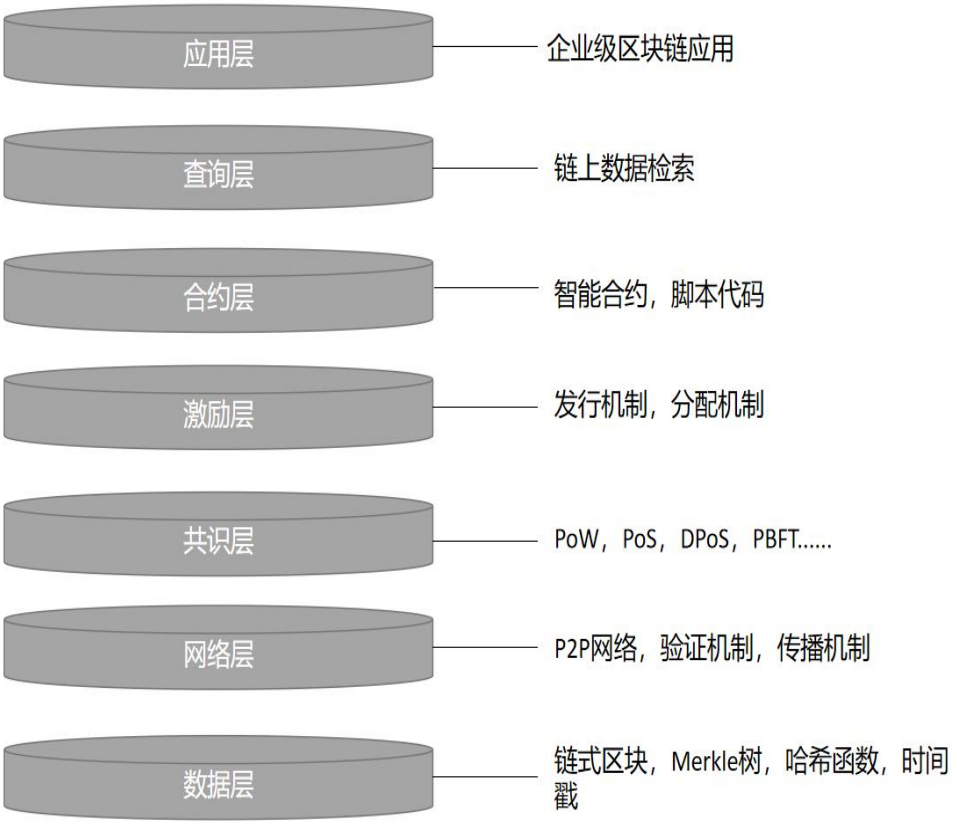

图 2.3 区块链逻辑层级图

Fig. 2.3 Blockchain logical hierarchy

①数据层：负责区块数据的存取、验证以及 Merkle 树的维护，其中，区块链链式结构保证了区块链数据的不可篡改性，而 Merkle 树则为交易验证提供了保证。

②网络层：负责系统的网络组网、消息传播以及数据验证。

③共识层：负责通过共识协议保证分布式环境中各节点帐本数据的一致性。

④激励层：负责代币发行与分配。

⑤合约层：提供编程机制以实现去中心化的交易逻辑。

⑥查询层：实现对交易账本数据的访问和验证，对账号状态的查询以及对区块数据的查询。

⑦应用层：基于区块链提供的基础服务，实现去中心化应用。已有很多工作[61][62][63][64][65][66]针对该层上的区块链技术应用做了讨论和综述。

2.1.2 以太坊状态树

本节从以太坊状态树结构和存储方式两个角度出发，介绍以太坊状态树的设计思路及特性，有助于更好理解后续第三章节中针对内容检索效率问题，本文所提基于状态树的内容检索优化方案。

（1）状态树结构

与比特币系统不同，以太坊采用基于账户的交易模型，系统显式地记录每个账户的状态，状态包括账户余额等全局信息。使用状态树 MPT（Merkle Patricia Trie）来完成由账户地址到账户状态之间的映射。MPT 是一种 Merkle 树与路径压缩前缀树（Patricia Trie）的结合，即将路径压缩前缀树中的普通指针换成了哈希指针，之所以选择基于前缀树（Trie）结构实现的数据结构有以下几点原因：

①比特币中使用 Merkle Tree 来组织块内交易以提供 Merkle 存在性证明，但 Merkle Tree 并没有提供高效的查询方法，检索只能通过遍历的方式进行。与单个区块所包含的交易数量相比，系统中所有账户的数量显然要大得多，使用前缀树组织账户状态，可以提供对账户的高效查询。

②若使用 Merkle Tree，为确保分布式环境中各节点状态树的一致，须对其叶节点排序使其变得有序，但在这种情况下新增账户时可能需要重构整棵 Merkle Tree，代价太大。而前缀树新增账户时，只需要在对应位置插入新增节点，再重构该分支上的节点即可，对其他分支并无影响，有更高的插入和更新效率。

③前缀树树中状态节点的位置仅由账户地址的值决定，而与节点插入顺序无关，该特性可以保证分布式环境中各节点存储状态树的全局一致性，且结合 Merkle Tree 可同时提供账户存在性证明和不存在性证明。

此外，以太坊为了避免账户地址的碰撞，采用了长为 160bit 的数据标识账户地址，由于账户地址比较稀疏，进一步使用了路径压缩前缀树（Patricia Trie）以减少存储空间并提高数据检索效率。状态树 MPT（Merkle Patricia Trie）结构如图 2.4 所示，MPT 是 Merkle 树与路径压缩前缀树 Patricia Trie 的结合，树中指针均为哈希指针，图中示例了 4 个 7 位账户地址到账户余额的映射过程，树中结点有如下三种类型：

• 扩展结点（Extension Node）：shared nibble（s）中保存了压缩的路径数据。

• 分支结点（Branch Node）：分支结点，无法压缩，value 中存储刚好在此分支节点结束的账户余额，若没有节点在此结束则 value 为空。

• 叶子结点（Leaf Node）：保存账户状态数据，value 中记录了账户的余额信息。

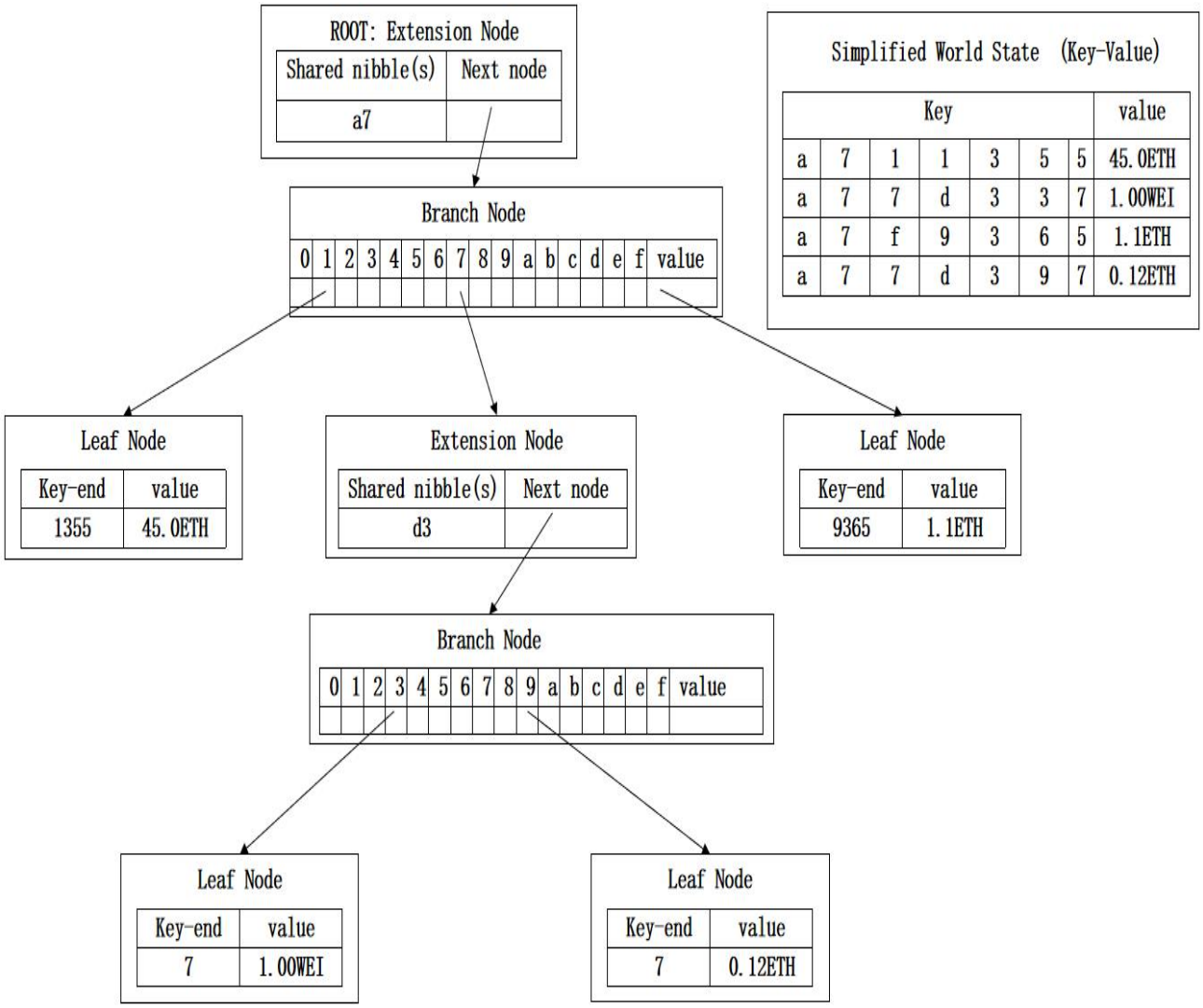

图 2.4　MPT 结构图

Fig. 2.4 MPT structure diagram

以太坊中除状态树之外还有交易树和收据树，同样都采用 MPT 的数据结构。交易树组织块内交易，每个交易执行完产生一个收据，记录这个交易的相关信息，收据用收据树组织，与交易树上的节点一一对应。与状态树所不同的是，交易树和收据树只与本

块内交易相关，独立于其他区块，即每个区块拥有完全属于本区块的交易树和收据树，一旦构建完毕便不再改变，而状态树维护的是全局账户状态信息，每产生一个区块都要进行更新，并将更新后的最新状态树根哈希会添加到新区块块头中，可以从最新区块中的状态树根获取到当前最新的全局账户状态树。

（2）状态树存储

由于每个区块拥有一棵当前时刻完整的状态树，为了避免存储空间的浪费，以太坊各区块间状态树采用共享节点的存储方式。如图 2.5 所示，当一个新区块发布时，新区块中会为变化的账户以重建分支的方式更新状态树，而与前序区块共享不变的状态分支。每更新一个账户状态，此账户状态叶节点的哈希值会改变，需对从该叶节点到根节点的所有中间节点的哈希值进行更新，最后得到新的根节点哈希。

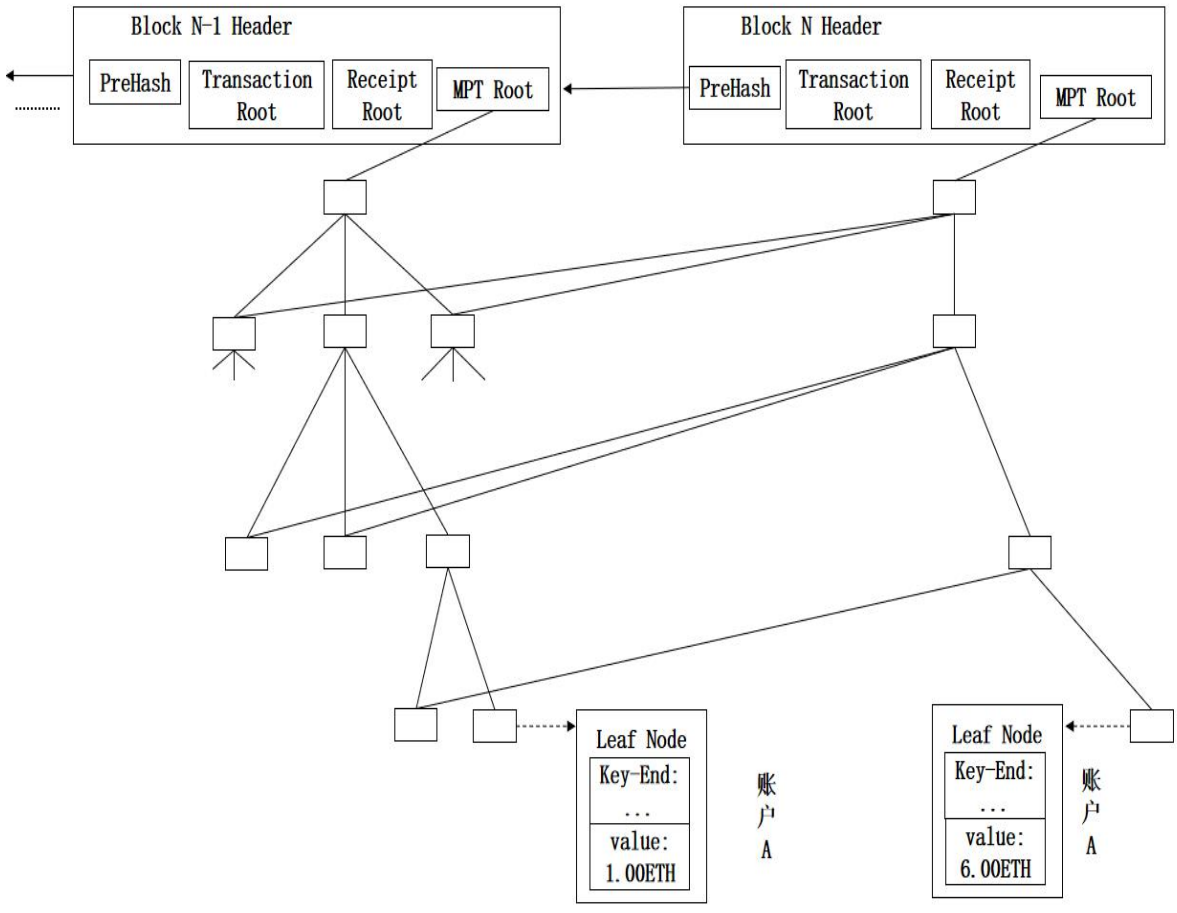

图 2.5　MPT 存储图

Fig. 2.5 MPT storage diagram

2.2 BCDB 模型

以比特币和以太坊为代表的区块链系统属于金融领域范畴，其数据格式单一，为改善这一情况，相关工作提出了 BCDB（Blockchain Database）模型[39]，对原先区块链交易数据格式进行了拓展，给出了更为一般化的区块链数据模型规范。本节首先对该数据模型进行介绍，并在该模型数据定义下，对现有方法在数据内容检索和溯源查询两方面的流程进行更详细的阐述分析，有助于进一步理解本文的研究问题和对象。

2.2.1 数据模型定义

现有区块链系统所存储的大都是固定格式的交易信息，数据格式单一，应用拓展性受到制约。针对此问题，BCDB 模型对原先比特币系统中的交易数据结构进行了修改，将交易数据结构扩展到任意记录格式，给出了应用拓展性更好的区块链数据格式规范。在该模型中数据和数据操作定义如下：

（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 2.6 所示，BCDB 模型中数据记录由数据头（Data Head）和数据（Data）组成。

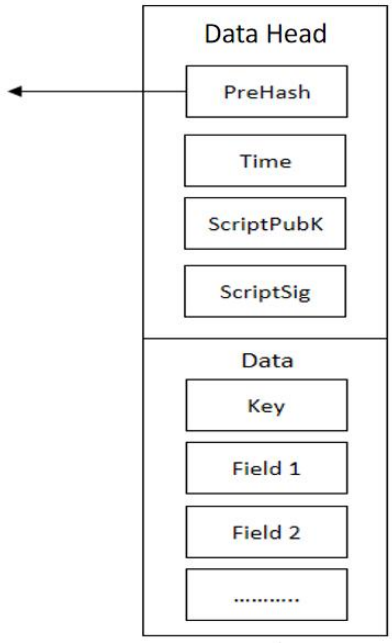

图 2.6 数据记录结构图

Fig. 2.6 Data logging structure diagram

其中 Data 部分为数据内容，Key 为关键字，是数据记录的唯一标识。Field 域为数据属性内容，可以为任意格式，以增强数据记录可拓展性。

数据头中包含 PreHash，指向相同 Key 标识数据记录的上一历史版本，实现数据修改溯源和不可篡改；Time 是该版本记录发布时间；ScriptPubK 指定拥有该数据记录写操作权限的下一拥有者公钥，ScriptSig 是拥有本数据记录写操作权限的拥有者的签名，用于证明本记录有效，即在写入数据的时候，先查询是否存在关键字为 Key 的数据记录，若存在，则检验当前数据记录的 ScriptSig 是否与前一数据记录的 ScriptPubK 相匹配，只有在匹配的情况下才可认为发布的该条数据记录有效。若 Key 是第一次出现，则将 Prehash 设置为 null。对某一 Key 数据记录的修改，则通过在新区块中写入新的相同 Key 数据记录，以区块追加的方式实现，所有历史记录版本存在于历史区块中，不可删除，查询数据记录时根据数据关键字 Key 进行内容检索，最后返回最新版本的数据记录。此外，可根据数据头中的 PreHash 进行数据溯源，追溯出特定 Key 数据记录的所有修改历史版本。

（2）数据操作

增加记录：添加某一新 Key 标识数据记录时，数据拥有者可以指定下一拥有对该条记录写操作权限的公钥，然后用自己的私钥对该记录签名发布。

修改记录：针对某一 Key 数据记录的修改通过在新区块中，发布一个新的相同 Key 记录以追加方式实现，这与区块链防篡改并不冲突。只有拥有对该 Key 数据记录写操作权限的操作者才可修改该 Key 数据记录，记录发布后需验证其私钥签名是否与父记录中公钥所匹配，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。针对关键字 Key 的内容检索，返回该 Key 标识数据记录的最新版本；而针对关键字 Key 的溯源查询，返回该 Key 标识数据记录的所有历史版本，即溯出其历史修改轨迹。

删除记录：数据记录一旦写入便无法删除，只能以追加最新版本的方式覆盖之前的历史版本。

2.2.2 内容检索流程

在区块链系统的查询层中，如何实现针对数据内容的检索功能，为区块链上层应用提供高效可信的内容检索服务，这一课题具有十分重要的研究意义和应用价值。而当前的大部分区块链系统实现中，在这方面还存在着比较明显的效率问题。

如在以比特币为典型代表的区块链系统中，各区块中存储的数据以 Merkle 树的形式组织，而 Merkle 树本身无索引结构，内容查询检索只能采用遍历区块并遍历块内数据的方式进行搜索，遍历过程中产生了大量的磁盘 IO，极大限制了检索的效率。

以图 2.7 为例，当检索目标数据为 TX1 时，全节点需从最近的区块 N+1 开始，依次遍历比对区块中的交易列表，检索完未发现目标数据，再继续向前遍历区块 N 中的交易列表，直至找到数据 TX1 为止。不难发现，这种依次向前遍历区块和各块内交易的方式，当目标区块 N 和最新区块之间有大量区块时，将会产生大量的磁盘 IO 和无效搜索过程，检索效率低下。

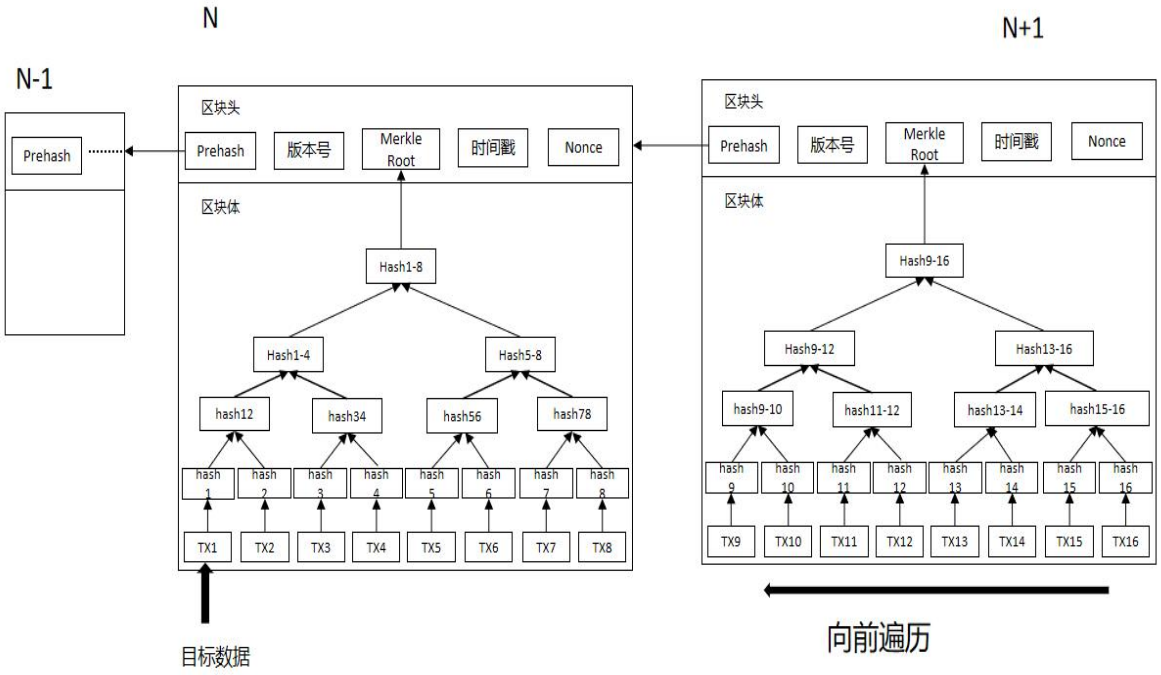

图 2. 7 内容检索示意图

Fig. 2.7 Content retrieval diagram

在 BCDB 模型中，为了改善这一情况，结合红黑树为区块内的 Merkle 树添加了针对数据关键字 Key 的内容索引，设计了新的块内索引结构 Merkle-RB，提高了块内数据内容检索的效率。同大部分内置索引的优化方法类似，对块内 Merkle 结构进行索引设计，能在保证检索结果可验证，数据可信性不受损失的情况下，提高在区块内的内容检索效率，但由于缺少全局性的索引结构，导致内容检索无法一次确定目标数据所在区块，

还需采用依次向前遍历区块的方式对目标数据进行搜索，内容检索的效率没有得到根本上的提升，还存在着很大的优化空间。

此外，优化后的块内 Merkle 结构本质上还是一种局部性索引，能为内容检索结果提供数据存在性证明，但无法提供数据不存在证明，查询可信性方面也存在着不足。

2.2.3 溯源查询流程

数据可溯源是区块链确保数据可信、可追溯的重要特性功能之一，其在金融、供应链、数字资产确权等领域都具有不可忽视的应用价值。

在 BCDB 模型中，数据溯源查询，指针对关键字 Key 的溯源查询，将根据数据中 PreHash 字段，依次递归地从底层存储系统中取出该数据记录的各历史版本，返回该 Key 标识数据记录完整的数据修改轨迹链路。

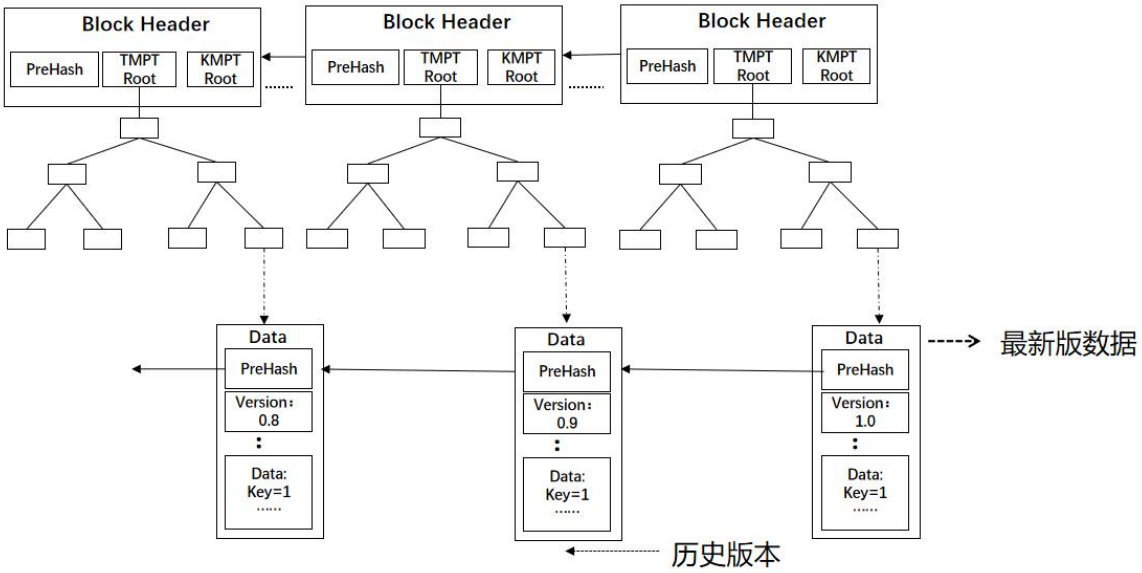

图 2.8 溯源检索示意图

Fig. 2.8 Schematic diagram of traceability retrieval

以图 2.8 为例，当要对关键字 Key=1 的数据做溯源查询时，首先需从当前最新区块开始，检索到 Key=1 对应数据记录的最新版本（Version=1.0），再根据该版本中 PreHash 字段，从底层存储系统（如 LevelDB）中获得上一版本的数据记录（Version=0.9），再由该版本中 PreHash 字段获取更上一版本数据记录（Version=0.8），重复该过程，直至遍历到数据记录的原始版本，以得到完整的溯源链，即该 Key 所标识数据记录的完整修改轨迹。

综上可知，溯源查询中产生了多次根据数据哈希值从底层存储系统中取数据的过程，而在当前大部分区块链系统中，这一过程的实现依赖于全节点底层的键值数据库系统（如 LevelDB），该模型单一地采用数据字段中的 PreHash 进行溯源检索，在溯源查询请求频繁的应用场景中可能产生大量重复低效的磁盘 IO 检索过程，降低溯源查询的响应速度，加重全节点负担，存在效率上缺陷。

此外，对溯源查询的技术方案做优化，应尽可能最大程度地减小全节点在溯源检索上的资源开销，这有利于让全节点分配更多的计算存储资源到消耗更大的网络共识计算中去，不影响区块链系统的整体效率和可用性。现有针对溯源查询的优化方法中，大都在全节点中为溯源查询引入新的数据结构，以提升溯源查询的检索效率，但多数方法设计的数据结构和溯源方法过于复杂，给全节点带来了较大的维护开销，对全节点整体系统效率造成了影响。

2.3 本章小结

在本章节中，首先介绍了与本文研究相关的区块链基础概念，然后以 BCDB 数据模型为例，介绍了相关技术并进一步对现有方法在数据内容检索和溯源查询两方面的流程及特性进行了更为详细的说明分析，有助于进一步理解明确本文研究问题和后续章节中将要介绍的研究方案。

3 基于状态树的内容检索优化

区块链在数据存储结构上与传统关系型数据库系统有很大差别，其特殊的数据组织方式保证了数据检索结果的可验证性，为数据可信提供了支撑。因此，在对其数据内容检索方式进行优化时必须保证检索结果的可验证性，即在不损失其数据可信性的前提下进行。如图 3.1 所示，网络中轻节点向全节点发送针对关键字 Key 的内容检索请求，全节点收到检索请求后，在自身所存储的完整区块链数据中根据该关键字进行内容检索。若检索成功，需向发起查询的轻节点返回目标数据记录及其验证路径（Path of Merkle Proof），轻节点收到检索结果后可由该验证路径完成数据存在性证明（Proof of Membership），对检索结果进行验证；若检索失败，则返回数据不存在信息。

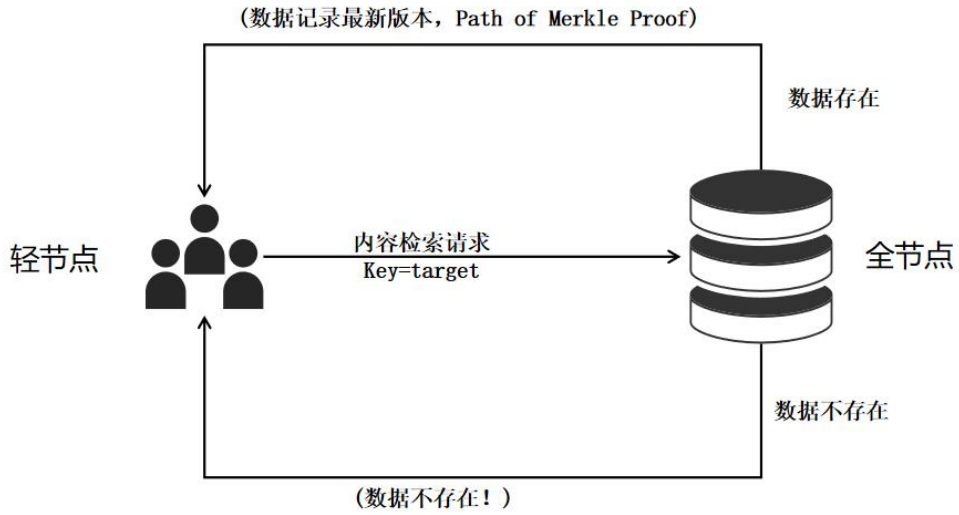

图 3.1 可验证内容检索示意图

Fig. 3.1 Verifiable content retrieval diagram

在目前对内容检索的优化方法中，外联数据库方法无法保证数据不可篡改性和检索结果可验证性，损失了区块链数据可信特性，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检索效率问题。因此，本研究采用内置索引的方法，旨在保证内容检索结果可验证的前提下，从根本上改善区块链全节点内容检索的效率。

现有内置索引优化方法多数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，检索的效率还存在着优化的空间。鉴于此，针对区块链数据内容检索效率低下的问题，本章节将介绍本文所提优化方案，即基于状态树的内容检索优化索引模型，将从索引结构、索引更新、基于该索引模型的内容检索算法等方面对该优化方案进行详细描述，并对其内容检索效率进行理论分析和实验验证。

3.1 索引结构

为解决区块链数据内容检索效率低下的问题，本文提出了基于状态树的索引优化模型。同传统区块链系统类似，模型中的指针都为哈希指针，借助 K-V 键值数据库（如 LevelDB）实现底层数据的存储。如图 3.2 所示，索引分为全局状态区块索引 KMPT（Key-MPT）和块内 TMPT（Transaction-MPT）索引，KMPT 与 TMPT 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 数据记录状态。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 TMPT 与 KMPT 都是根据数据记录 Key 值所构建而成，以实现高效内容检索。

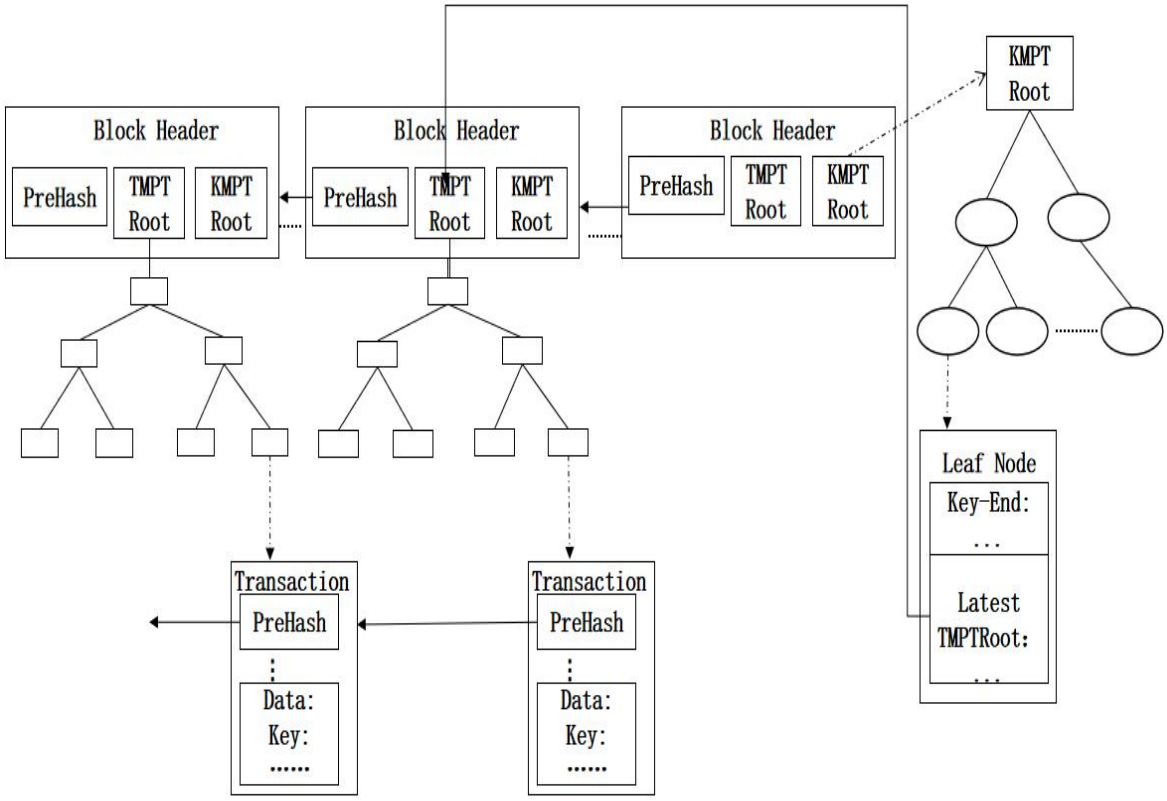

图 3.2 索引结构图

Fig. 3.2 Index structure diagram

如图 3.2 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，组织块内数据记录，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有更高的

检索效率。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，与图 3.1 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 标识数据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，避免遍历区块搜索目标数据的过程。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 结构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 数据记录状态，即其所在区块中存储的 TMPT 根哈希信息。与以太坊相同，每个区块都有一颗代表当前时刻的最新完整 KMPT，存储了到本区块时刻的全局 Key 记录状态信息。因而为了减轻全节点存储负担，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。

3.2 索引更新算法

（1）数据插入

插入算法是 TMPT 和 KMPT 索引构建的基础，以图 3.3 为例说明在 MPT 中插入 4 个 Key-Value 对的构建算法过程。

如图 3.3-①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 3.3-②所示，将<a77d337,value2>插入，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 3.3-③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 3.3-④所示，该 Key 与 a77d337 共享前缀 a7+d3，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：

Hash(Leaf Node)=Hash(Node.Key-End,Node.Value)

扩展节点：

Hash(Extension Node)=Hash(Node.Shared nibble(s),Node.Next Node)

分支节点：

Hash(Branch Node)=Hash(Node.Children[0],...,Node.Children[f],Node.Value)

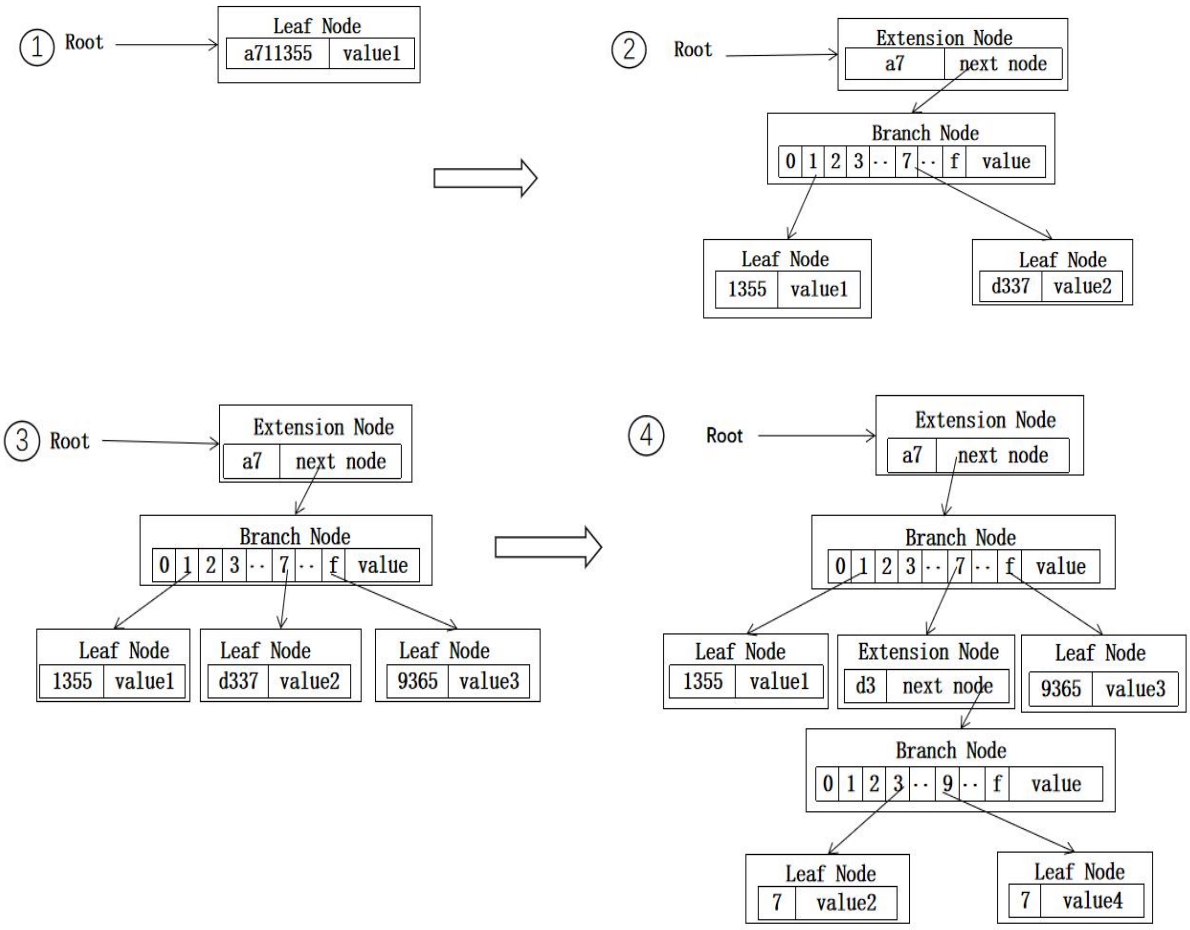

图 3.3 MPT 构建过程示意图

Fig. 3.3 Schematic diagram of the MPT build process

分支节点的哈希计算，即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。

在本文索引模型中，TMPT 与 KMPT 的构建都基于以上插入算法。所区别的是在 TMPT 维护的 Key－Value 对中，Key 为数据的 Key 值，Value 为具体数据记录的哈希值，而在 KMPT 中，Key 为数据 Key 值，Value 为 Key 记录最新版本所在目标区块中的 TMPT 根哈希值，即 LatestTMPTRoot。

（2）索引更新算法

索引更新以区块为单位，首先根据新区块内数据 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，以新建分支方式更新 KMPT，最后将 KMPT 根哈希添加到区块头中完成索引更新。相应的索引更新算法如下：

算法 1. 索引更新算法：

```
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 第 2-5 行为块内 TMPT 索引的构建，第 2 行首先初始化 TMPT 根；从第 3 行开始遍历交易集合，第 4 行将交易插入到块内 TMPT 中，具体插入细节如（1）中所述，并记录每次插入后的最新 TMPTRoot，第 5 行循环结束，此时块内 TMPT 索引构建完毕。

第 6-10 行为全局 KMPT 索引的构建，第 7 行首先从当前系统中最新块获取到最新 KMPTRoot，第 8 行开始遍历交易集合，第 9 行以当前 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，具体方法细节如（1）中所述，并记录插入后的 KMPT 根哈希。

最后循环结束，KMPT 索引更新完毕，11-12 行将构建完成后的最新 TMPT 及 KMPT 根哈希添加到区块头中，第 13 行将区块返回，即完成一个区块的索引构建及 KMPT 更新过程。

3.3 内容检索算法

由数据记录关键字 Key 值进行内容检索，分为两个检索过程，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 再开始进行块内内容检索。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的 Merkle 证明路径，用于对内容检索结果提供验证。相应的针对数据记录关键字 Key 的内容检索算法如下：

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 第 2 行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。

第 3-4 行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，第 5 行由此根哈希可以在 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。

第 6-7 行为该 Key 记录不存在的情况，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。

第 8-13 行对应该 Key 记录存在的情况，其中第 9 行从在 KMPT 中查询到的状态节点中获取到该 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，第 10 行将 path 清空，然后第 11 行根据获取到的 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 的最新版本存在性证明。

最后 12 行将检索到的目标 Key 记录和块内检索路径返回。

综上可知，采用本文的基于状态树的内容检索优化索引模型，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，随后直接开始块内内容检索，避免了遍历区块检索方式带来的大量无效搜索过程，节省了查询过程的时间消耗。

此外，在本文索引模型中，若 Key 对应数据记录存在，块内 TMPT 检索的过程即提供 Key 最新版本数据记录 Merkle 存在性证明路径的过程；若 Key 不存在，则在 KMPT 中的检索过程即为提供 Key 记录不存在证明路径的过程。即模型可同时提供基于 Key 值查询的数据存在性和不存在性证明。

针对查询最新版本数据记录的存在性证明与其他区块链系统中的 Merkle 存在性证明类似，这里仅针对不存在证明进行说明（Proof of Non-membership）。如图 3.4 所示，KMPT 中存储有 Key 为 a711355,a77d337,a7f9365,a77d397 的四个 Key 数据记录状态信息，对某个不存在的 Key 为 a77d367 的记录查询请求，需提供其数据不存在证明。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，轻节点接受到该分支路径 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 Nextnode 域中的哈希值比较，一致则 P2 有效，对 P2 进行匹配；P3，P4 同理，验证 P4 有效后，即说明该分支有效，对 P4 匹配失败，及证明 Key 为 a77d367 的记录确实不存在，即完成数据不存在证明的过程。

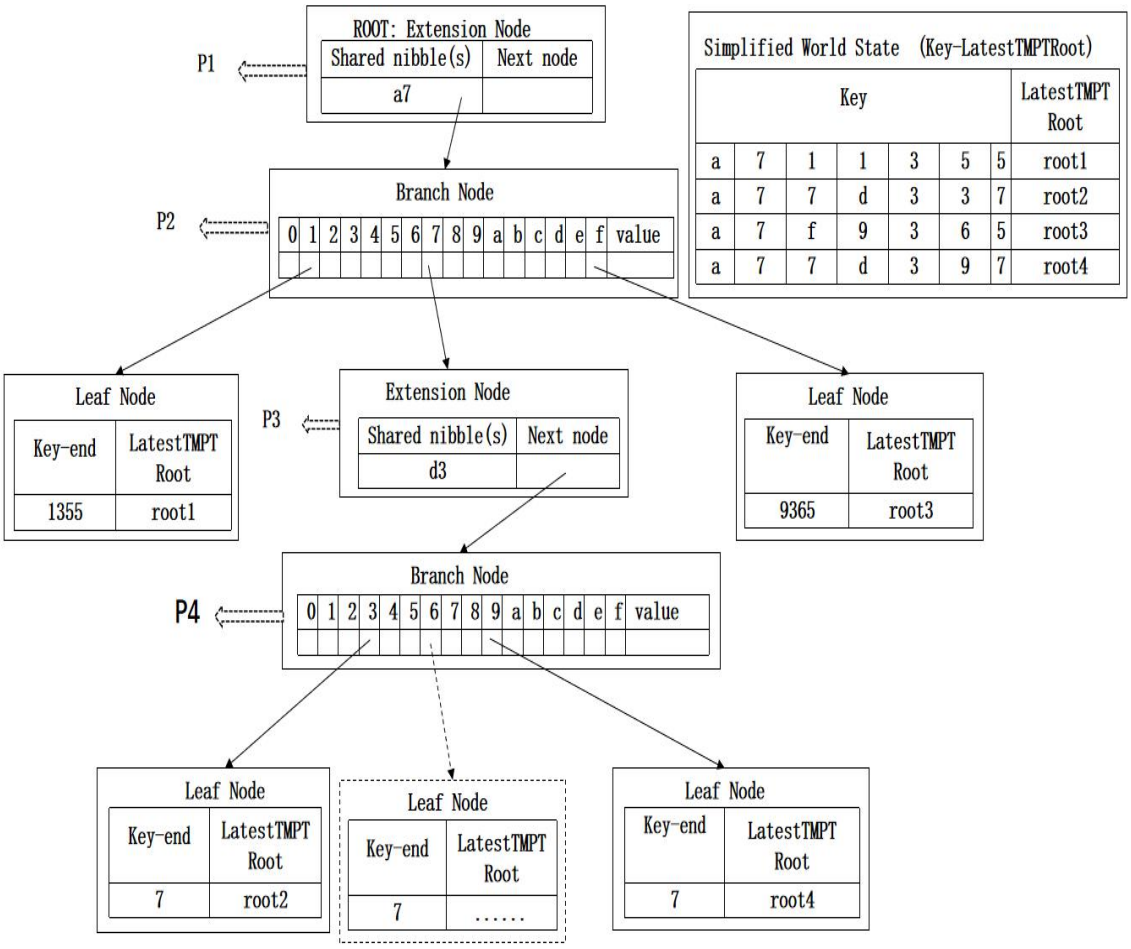

图 3.4 不存在证明路径

Fig. 3.4 No proof path diagram

此外，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，针对数据关键字 Key 的溯源查询可追溯出该 Key 标识数据记录的所有历史版本，具体溯源查询算法如下：

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 第 2 行首先创建用于保存溯源查询结果的交易数据记录列表.

第 3 行在系统中查询 Key 对应的最新版本记录。

第 4-5 行即查询 Key 对应记录不存在，则返回空值并结束程序。

第 6-14 行对应查询 Key 记录存在的情况，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。

最后第 15 行返回保存的记录列表，结束。

3.4 内容检索算法效率分析

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，故查询某一特定 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：

$$T_2 = O(N \cdot logM)$$

本文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，将两个过程的时间复杂度都降到了对数级别，即本文索引模型下检索时间复杂度T_3为：

$$T_3 = O(log\ NM)$$

显然：

$$T_3 < T_2 < T_1$$

即本文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。

另一方面，由于引入了基于状态树的全局性索引 KMPT，增加了额外的维护消耗，在单块索引构建上，构建时间复杂度开销为：

$$T_{cost} = O(log\ M)$$

在存储空间方面，该状态树为路径压缩前缀树，其空间消耗由整个区块链中不同 Key 值的数量 Num_{Key} 决定，故假设 R 为区块链数据中不同 Key 的占比，即：

$$R = Num_{Key}/NM$$

（$0 < R < 1$）

则以共享节点方式构建的该状态树索引存储空间复杂度 S 为：

$$S = O(R \cdot NM)$$

即本文索引模型方法在理论上以一定的时间 T_{cost} 和空间 S 消耗为代价，将内容检索时间随数据量线性增长降到了对数级别，改善了内容检索的效率。

3.5 实验对比与分析

由于本文研究重点属于区块链全节点中的数据查询检索优化范畴，不涉及区块链网络共识部分，故实验环境采用单机搭建模拟全节点完成各项测试验证，实验中所采用的单机硬件环境为 Intel(R) Core(TM) i5-7300HQ CPU（2.50 GHz），RAM（8GB），操作系统 Windows 10。实验底层数据库使用 LevelDB，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 语言进行搭建。参考 BCDB 模型数据定义，针对原区块链中交易格式，交易块内组织形式，状态树结构等底层数据结构进行了修改，实现本文基于以太坊状态树的索引优化模型。同时，为进行对比实验，实现了 BCDB 模型中的 Merkle RB Tree 结构，作为块内索引优化方法的代表，以及原始 Merkle Tree 结构，作为传统区块链系统检索方法的代表，与本文方法进行各项对比实验。

• 实验 1：区块深度对于查询时间的影响

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 Key 记录的查询需返回其最新版本。当系统中最新区块块高度为 h1，目标交易最新版本所在区块高度为 h2 时，定义区块深度为 h=h1-h2，本实验探究目标交易最新版本所在区块深度对查询时间的影响情况。实验顺序写入 100 万条 Key 升序的数据记录，每个区块设置存储 1000 笔数据记录，总共 1000 区块，当目标交易所在区块深度为 h=100，200，300....1000 时实验结果如图 3.5 所示。

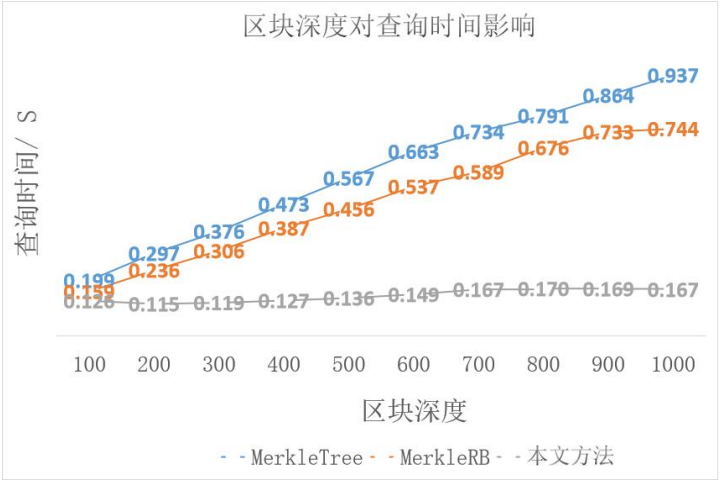

图 3.5 区块深度对查询时间影响

Fig. 3.5 Impact of block depth on query time

由实验结果可知，Merkle Tree 方法和 MerkleRB 方法查询时间随目标 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 MerkleTree 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，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 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，相比 Merkle Tree 方法有更好的检索效率。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，使查询时间仍随区块深度线性增加；而基于本文索引模型的方法，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，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，符合 3.4 章节中的理论预期。

• 实验 2：目标 Key 不存在情况下查询响应时间与区块数的关系

本实验探究极端情况下，若所查询目标 Key 不存在时，三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 1000 笔数据记录，当系统中总区块数为 100，200，300...1000 时，输入一个不存在的目标 Key 值，测试查询响应时间。

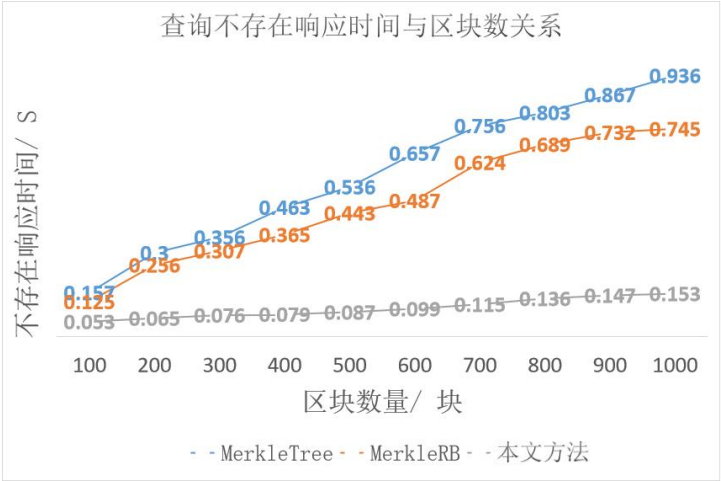

图 3.6 查询不存在响应时间与区块数关系

Fig. 3.6 Relationship between no-query response time and number of blocks

实验结果如图 3.6 所示，由于查询目标 Key 值不存在，Merkle Tree 和 Merkle RB 方法需遍历检索完全部区块数据才可返回不存在信息，两者查询响应时间都随系统区块数线性增长，其中 Merkle RB 方法由于加快了块内检索的速度，所以响应时间比 Merkle Tree 方法更短。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 树中检索即可返回不存在信息及不存在证明路径，不需遍历检索区块数据，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。

• 实验 3：数据溯源查询效率对比

在 BCDB 模型中，数据溯源指查询出某特定 Key 记录所有历史版本，以溯出该 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两个过程，因此溯源查询时间与最新版本的查询时间相关。在 Merkle RB 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 方法与本文 KMPT 方法在追溯历史版本上的效率，实验首先将目标数据最新版本所在区块深度固定为 100，测试历史版本数分别为 10，20，30...100 时溯源查询时间。

实验结果如图 3.7 所示，由图可知，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，因此具有相近的历史版本追溯效率，且历史版本数对溯源查询的时间影响较小。

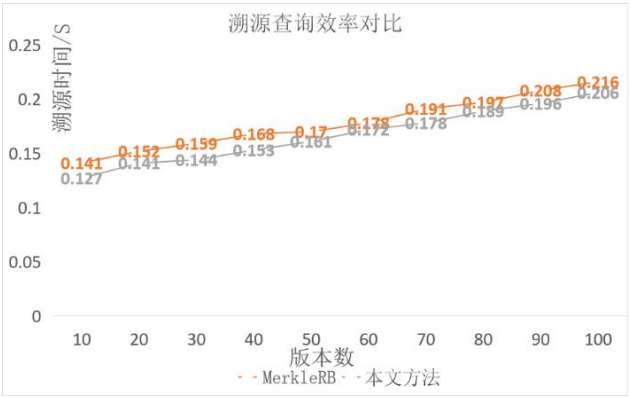

图 3.7 历史版本数对溯源查询效率影响

Fig. 3.7 Influence of historical version number on traceability query efficiency

由于溯源查询受最新版本查询时间影响，本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...1000 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 3.8 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，而本文方法溯源查询仍不受影响，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。

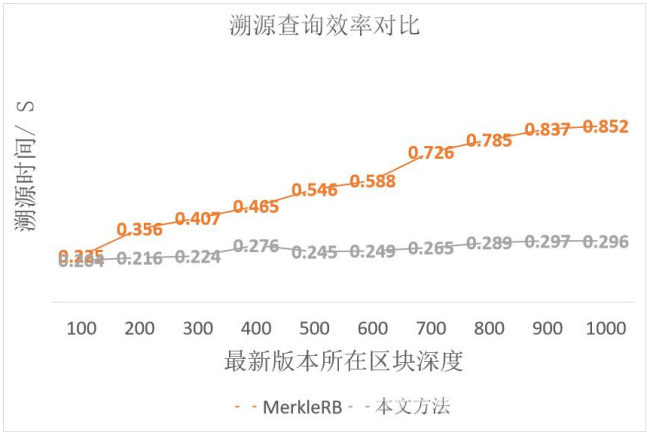

图 3.8 最新版本区块深度对溯源查询效率影响

Fig. 3.8 The impact of block depth of the latest version on the efficiency of traceability query

• 实验 4：索引构建代价对比

索引的更新以区块为单位，区块内包含的数据记录数量决定了索引的构建时间，本实验探究在不同数据数量下 Merkle RB 与本文索引模型构建的代价对比。

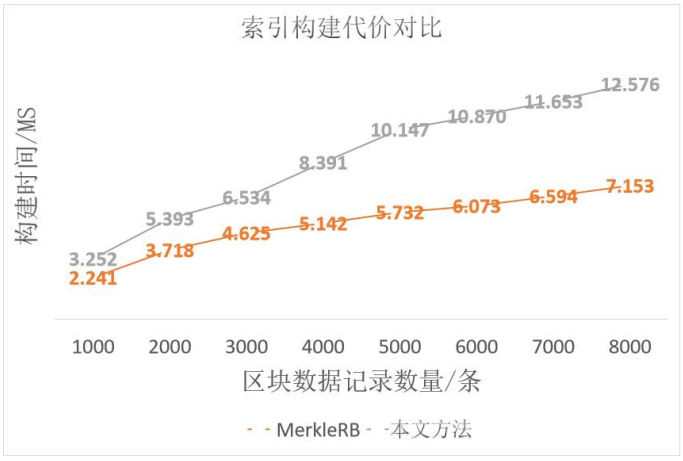

图 3.9 索引构建代价对比

Fig. 3.9 Index construction cost comparison

实验结果如图 3.9 所示，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长，但与查询效率及稳定性的提升相比，单块索引构建的时间代价维持在毫秒级别，没有为系统带来过大的额外负担。此外，由 3.4 节中分析可知，本文索引模型中 KMPT 采用以太坊中状态树的存储方式（共享分支节点）以节省空间，且在空间开销上受不同 Key 值数据占比 R 影响，而在实际环境中 R 的值是远小于 1 的（重复的 Key

值较多），进一步降低了模型的空间开销，综合性价比较高。即本文索引模型在可接受的代价范围内，将内容检索的时间由随数据量线性增长降到了对数级别，改善了内容检索的效率和稳定性，同时增强了查询可信性。

3.6 本章小结

在本章节中，针对区块链数据内容检索效率低下的问题，介绍了本研究所提的优化方案，即基于以太坊状态树的内容检索索引模型。基于以太坊状态树结构，设计了具有全局特性的区块索引结构 KMPT，实现了精准定位目标数据所在目标区块的功能，避免了内容检索过程中遍历无关区块产生的大量无效搜索过程，从根本上改善了内容检索的效率，结合块内索引 TMPT，实现了基于关键字 Key 的高效区块链数据内容检索。经实验验证，基于本文所提索引模型，内容检索的效率，稳定性得到了本质上的提升。此外，受益于状态树的全局性特征，基于该索引模型可同时为查询方提供数据存在和不存在性证明，进一步增强了内容检索的可信性。

值得一提的是，本研究在实现针对数据关键字 Key 的高效内容检索的同时，由于提升了 Key 标识数据记录最新版本的检索速度，对数据溯源检索的效率也做出了一定的贡献。但是，并没有改变数据溯源检索的基本方式，溯源仍需由数据中的 PreHash 字段向底层存储系统中依次遍历溯出数据的各历史版本，在一些对数据溯源有着更高效率要求的场景中，存在很大的缺陷。因此，本文的下一步研究工作是：如何进一步提升数据溯源查询的效率，以满足更高要求的溯源场景需要。

4 基于多级缓存的溯源查询优化

数据可溯源是区块链保证数据可信的重要特性之一，随着区块链技术应用不断深入，一些场景对溯源查询的效率有了更高的要求。现有区块链系统大都使用哈希指针，从底层存储系统（如 LevelDB）中依次递归遍历出完整的数据溯源链路。LevelDB 是一种针对写密集型应用而设计的键值对数据库，以牺牲读性能为代价换取写性能的提高，在传统区块链系统中写数据频率远高于数据查询的频率，因此 LevelDB 是适用的。随着区块链技术的不断深入应用，许多应用场景对区块链数据溯源查询的需求正在不断增加，在一些数据溯源请求频繁的场景中，由于 LevelDB 读性能的不足，可能在底层存储系统中产生大量冗余重复的低效磁盘 IO 检索过程，降低溯源查询的效率。

参考 BCDB 模型中的溯源规范，典型的溯源查询从检索逻辑上可划分为两个阶段：第一阶段，由溯源数据关键字 Key 确定结果的哈希值（哈希指针）集合；第二阶段，由第一阶段得到的哈希值集合在 levelDB 中查询检索出完整的溯源数据集合并返回。在研究[67]中对这两个阶段的检索耗时情况进行了分析，如图 4.1 所示。

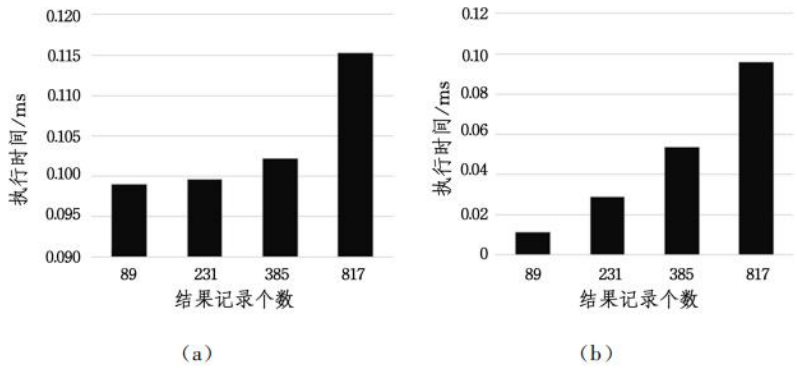

图 4.1 溯源查询一、二阶段检索耗时

Fig. 4.1 The first and second stages of traceability query take time to search

其中图 4.1-（a）为使用内置索引方法后溯源查询第一阶段检索的平均耗时情况，图 4.1-（b）为第二阶段平均耗时情况。由图可知，在一些溯源查询需求频繁的场景中，随着溯源检索请求数量的增长，溯源结果集合范围的不断增加，溯源一二阶段的耗时也同步增加，这导致了溯源查询效率的降低。

深入分析可知，本文第三章研究中针对内容检索问题提出的基于状态树索引优化方案属于内置索引优化的范畴，而内置索引方法仅能提高检索目标数据哈希值的效率，即

改善第一阶段的检索耗时情况，其对第二阶段并没有产生影响，底层存储系统 LevelDB 的数据读效率决定了溯源第二阶段的检索耗时。因此，改善溯源检索第二阶段对底层存储系统的依赖，成为提升溯源查询效率的关键。

为提升溯源查询的效率，很多研究工作为溯源查询设计了新的辅助优化结构，以提高全节点针对数据溯源的检索能力，但在这些方法中引入的数据结构普遍较为复杂，增加了全节点的维护开销，可能对全节点用于网络共识的计算存储资源进行抢占，影响区块链系统的整体性能。此外，这类方法并没有减轻溯源查询中第二阶段对底层存储系统的依赖，无法满足溯源请求频繁场景中的要求。

对此，本章节展开了研究。首先，参考在 2.2 章节中介绍的 BCDB 模型区块链数据模型规范，数据溯源指由数据关键字 Key 对其所标识的数据记录进行溯源查询，以追溯出该 Key 标识数据记录的所有历史版本，即完整的数据修改轨迹链路。在对数据溯源检索过程进行深入分析后发现，为得到完整的溯源链路，从底层存储系统中取出各历史版本数据是必要的，此过程优化的空间不大。因此，本文研究方案从提升网络中重复溯源查询请求的检索效率入手，为溯源查询引入缓存，旨在用高效的内存操作替代重复低效的磁盘 IO 检索，以降低对底层存储系统的依赖，改善溯源查询的效率。需考虑的是，如何为全节点设计内存空间更小，维护代价更低的溯源查询缓存结构，在提升数据溯源查询效率的同时，降低其维护过程的资源消耗，不影响区块链系统整体性能。

综上所述，针对在数据溯源请求频繁的场景中，如何在不给全节点带来过大负担的情况下，进一步提升区块链数据溯源查询的效率问题。本章节将介绍本文所提优化方案，即基于多级缓存的溯源查询优化方案，分别从多级缓存结构，溯源查询流程和缓存更新方法方面进行详细阐述，并对该优化方法中溯源查询的效率进行理论分析和实验验证。

4.1 多级缓存结构

为了克服现有技术方案中存在的不足，本文提供了一种区块链数据溯源查询优化方法，通过引入溯源查询缓存的方法，利用高效内存操作来减少溯源中低效的磁盘 IO 次数，缓解溯源查询对底层存储系统 Level DB 的依赖，提升溯源检索效率；同时设计了多级缓存的优化结构，实现在提升溯源检索效率的同时，尽可能减小全节点维护缓存的计算存储资源，兼顾溯源效率与资源消耗，减轻全节点负担。

如图 4.2 所示为本文所提多级缓存优化结构，全节点中的缓存为多级缓存结构，包括一级缓存和二级缓存，一级缓存中存储 key 标识数据的各历史版本哈希值和数据，即（Hash_n, Data_n）→（Hash_n-1, Data_n-1）...（Hash_0, Data_0），二级缓存只存储数据各历史版本的哈希值，即（Hash_n）→（Hash_n-1）...（Hash_0），以减小缓存数据

存储量。对应的缓存淘汰降级机制执行时机可采用定时更新或设置存储上限的策略完成，以设置存储上限阈值为例。当一级缓存达到设置存储上限时，触发淘汰降级机制，将查询频率较低的 key 数据缓存降为二级缓存。对应地，当二级缓存空间达到上限时，将剔除二级缓存中所有数据。两种策略的执行算法过程相同，不同的是执行的时机。

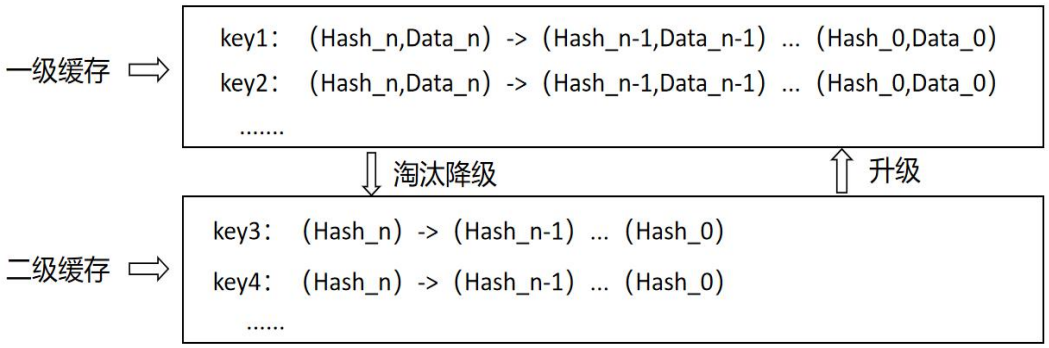

图 4.2 多级缓存结构图

Fig. 4.2 Multi-level cache structure diagram

多级缓存技术方案的关键在于将一段时间内查询过的数据溯源链路保存，全节点收到对重复 key 的溯源请求时，可借助高效的内存操作直接从缓存中取出数据，验证后返回溯源结果，减少溯源过程中对重复 Key 溯源的大量冗余磁盘操作，以此提升溯源效率，让全节点节省更多的计算资源投入到消耗更大的网络共识工作中去。

4.2 溯源查询流程

全节点引入多级缓存后，为利用多级缓存提升溯源查询效率和维护数据一致性，全节点处理溯源查询请求的流程发生了变化。如图 4.3 所示。以 Key 为关键字标识的溯源目标为例，全节点溯源处理过程总体上可分为三步，首先由第三章中所述内容检索方法检索出该 Key 对应的最新版本数据，记为（LatestHash，LatestData），其中 LatestData 为最新版本数据内容，LatestHash 为最新版本数据哈希值。然后由该 Key 查询多级缓存，取出的（Hash_n, Data_n）→（Hash_n-1, Data_n-1）...（Hash_0, Data_0）为多级缓存中存储的该 Key 所标识的数据溯源链，最后将（LatestHash，LatestData）与（Hash_n, Data_n）进行数据一致性校验处理，主要是对溯源链路的完整性进行校验，以保证溯源的链路在当前时刻区块链账本中是完整的，得到该 Key 标识的数据完整溯源链路（LatestHash，LatestData）→...（Hash_n, Data_n）→（Hash_n-1, Data_n-1）...（Hash_0, Data_0）。

在溯源检索过程中数据哈希值与数据成对出现，由此通过数据哈希值可方便地对数据进行防篡改和正确性校验，以确保溯源数据可信性，溯源的检索过程即为对应数据完整性和正确性的校验过程。

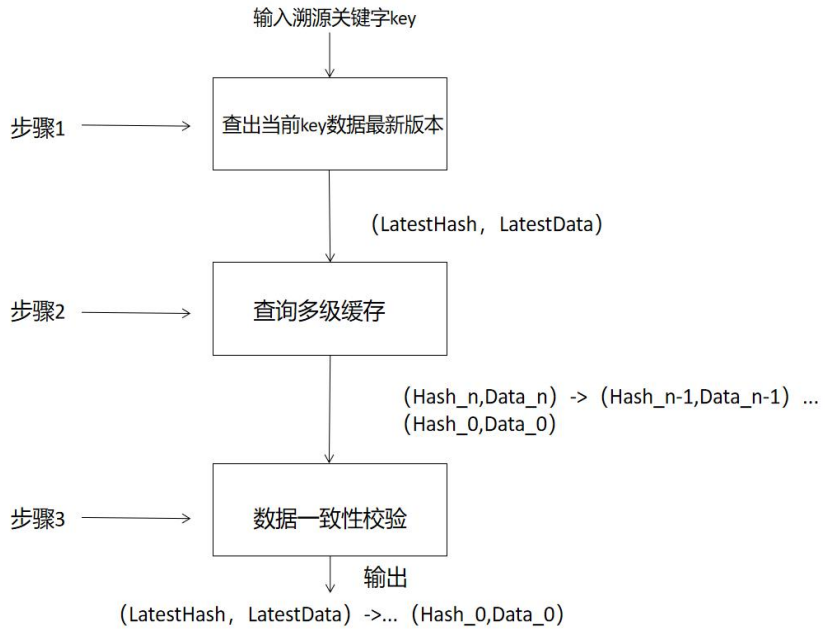

图 4.3 全节点溯源流程图

Fig. 4.3 Full node traceability flow chart

4.3 缓存更新方法

在本文的技术方案中，溯源查询检索算法流程与缓存更新维护算法同时进行，以保证多级缓存与区块链账本中数据一致性，全节点中具体的溯源检索和缓存更新算法流程如图 4.4 所示。

步骤 1，输入目标数据标识关键字 key，由网络中的节点向区块链全节点发起溯源查询请求，请求包含溯源目标数据标识关键字 key，全节点收到请求后，在当前区块链账本中查询出所述 key 对应数据的最新版本哈希 LatestHash 及最新数据 LatestData，记为（LatestHash，LatestData）；

步骤 2，在区块链全节点的缓存中查询步骤 1 中的 key；

步骤 2.1，若区块链全节点的缓存中不存在所述 key 对应的数据，则由（LatestHash，LatestData）在当前区块链账本中依次向前遍历出完整溯源链（LatestHash，LatestData）

→...（Hash_0, Data_0）并返回，最后将所述遍历得到的完整溯源链添加到所述 key 对应缓存中，结束对所述 key 的溯源查询；

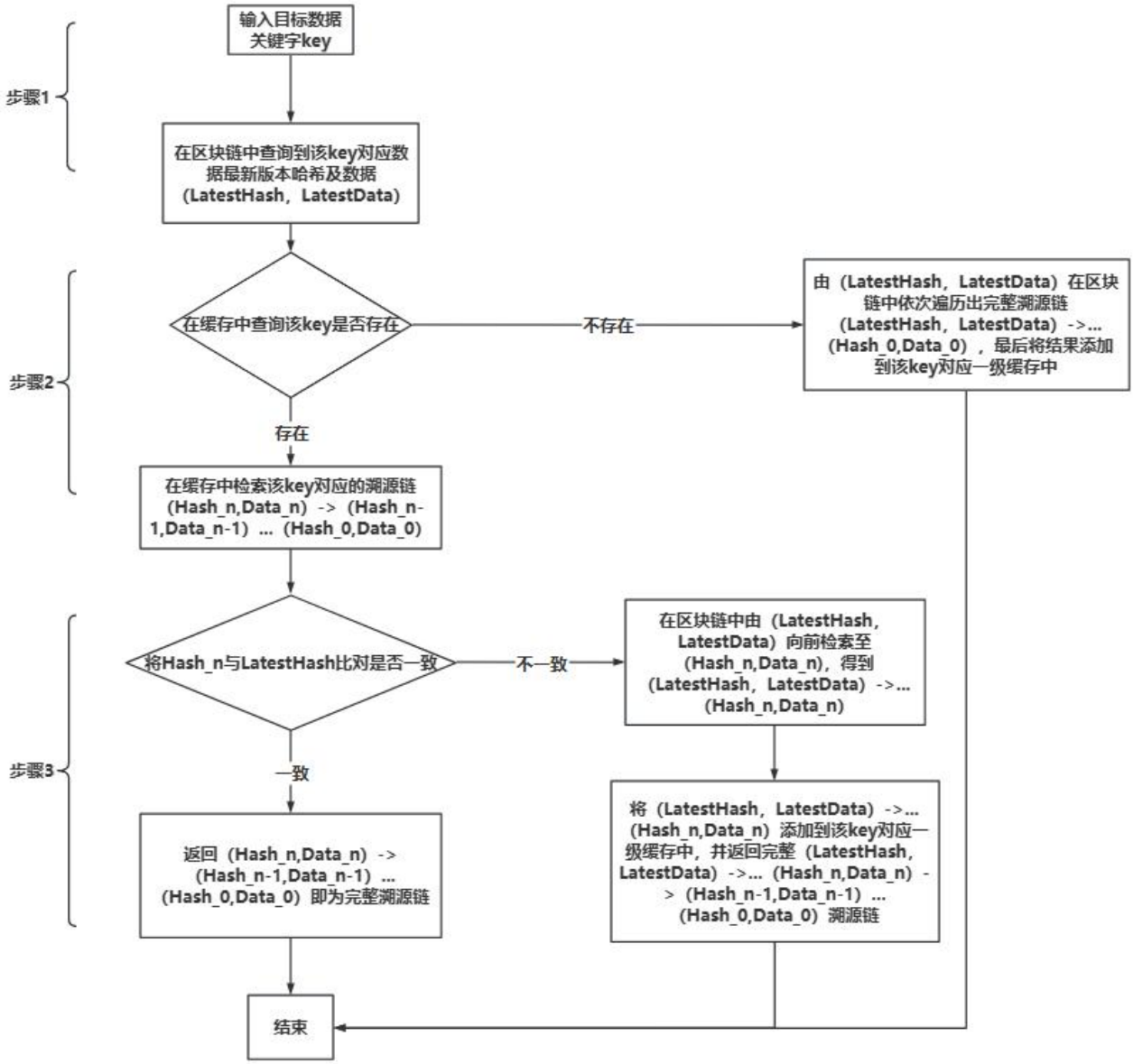

图 4.4 检索流程图

Fig. 4.4 Search flow chart

步骤 2.2，若缓存中存在 key 对应的数据，则在缓存中检索出 key 对应的完整溯源链（Hash_n, Data_n）→（Hash_n-1, Data_n-1）...（Hash_0, Data_0），执行步骤 3；

步骤 3，全节点进行一致性校验，将步骤 1 得到的 LatestHash 与步骤 2 在缓存中检索出的（Hash_n, Data_n）中的 Hash_n 比对，进行溯源完整性校验；

步骤 3.1，若 LatestHash 与 Hash_n 一致，则缓存中得到的（Hash_n, Data_n）→（Hash_n-1, Data_n-1）...（Hash_0, Data_0）即为完整溯源链，返回该结果，结束对所述 key 的溯源查询；

步骤 3.2，若 LatestHash 与 Hash_n 不一致，则在区块链中由（LatestHash，LatestData）向前检索至（Hash_n, Data_n)，得到（LatestHash，LatestData）→...（Hash_n, Data_n），将其添加到所述 key 对应的缓存中，以保证缓存与区块链账本中数据一致性，并与步骤 2 在区块链全节点的缓存中检索出的（Hash_n, Data_n）→（Hash_n-1, Data_n-1）...（Hash_0, Data_0）溯源链拼接成完整溯源链（LatestHash，LatestData）→...（Hash_n, Data_n）→（Hash_n-1, Data_n-1）...（Hash_0, Data_0），返回该结果，结束对所述 key 的溯源查询。

进一步地，所述步骤 2 全节点中的缓存为多级缓存结构，包括一级缓存和二级缓存，一级缓存中存储某 key 标识数据的各历史版本哈希值和数据，即（Hash_n, Data_n）→（Hash_n-1, Data_n-1）...（Hash_0, Data_0），二级缓存只存储数据各历史版本的哈希值，即（Hash_n）→（Hash_n-1）...（Hash_0），一级缓存达到存储上限时，触发淘汰降级机制，将查询频率较低的 key 数据缓存降为二级缓存，二级缓存空间达到上限时，将剔除二级缓存中所有数据，以节省内存空间占用并提高缓存命中率。

进一步地，所述步骤 2.1 中遍历得到的完整溯源链添加到所述 key 对应的一级缓存中。

进一步地，所述步骤 2.2 中，若缓存中存在该 key 对应的数据，且存在于一级缓存中，则从一级缓存中检索出缓存存储的完整溯源链，即（Hash_n, Data_n）→（Hash_n-1, Data_n-1）...（Hash_0, Data_0）；若数据存在于二级缓存中，则在二级缓存中取出（Hash_n）→（Hash_n-1）...（Hash_0），再从区块链底层数据库中批量取出各哈希值对应的数据，得到结果（Hash_n, Data_n）→（Hash_n-1, Data_n-1）...（Hash_0, Data_0），将其放入一级缓存中，进行缓存升级。

4.4 溯源查询效率分析

在本技术方案中，主要通过引入缓存，借助高效的内存操作替代磁盘操作，来提高区块链链上数据溯源检索的效率，同时通过设计多级缓存结构，节省了内存空间开销。

以溯源长度为 N 的目标溯源数据为例，$t_{磁盘}$为磁盘平均访问时间，$t_{内存处理}$为内存平均处理时间。在无缓存优化时，定位到数据最新版本后，所需溯源平均时间 $T_{原始溯源}$为：

$$T_{原始溯源} = N \cdot （t_{磁盘} + t_{内存处理}）$$

一般情况下，内存访问速度是纳秒级（10 的-9 次方），硬盘的访问速度是微秒级（10 的-3 次方）。在顺序访问情况下，内存访问速度仅仅是硬盘访问速度的 6~7 倍而在随机访问情况下，内存访问速度就要比硬盘访问速度快上 10 万倍以上，因此：

$$T_{原始溯源} \approx N \cdot t_{磁盘}$$

引入多级缓存后，其中 A 为缓存命中率（0<A<1）。假设区块链账本中数据和缓存数据已保持一致，即若缓存命中，则缓存中存储有完整的溯源链数据或完整溯源链哈希值。平均溯源时间需讨论：

当所需溯源链数据全在一级缓存中时，即多级缓存最好情况平均溯源时间为

$$T_{最好} = （1 - \mathrm{A}）\cdot \mathrm{N} \cdot \mathrm{t}_{磁盘} + \mathrm{A} \cdot \mathrm{N} \cdot \mathrm{t}_{内存处理} \approx （1 - \mathrm{A}）\cdot \mathrm{N} \cdot \mathrm{t}_{磁盘}$$

当所需溯源链数据全在二级缓存中时，即最坏情况，此时由于二级缓存中存有完整的溯源链哈希，可采取批量并发的磁盘操作以提高效率，因此多级缓存最坏情况平均溯源时间：

$$T_{最坏} = （1 - \mathrm{A}）\cdot \mathrm{N} \cdot \mathrm{t}_{磁盘} + \mathrm{A} \cdot \mathrm{N} \cdot \mathrm{t}_{批量磁盘}$$

可知：

$$T_{最坏} < \mathrm{N} \cdot \mathrm{t}_{磁盘} = \mathrm{T}_{原始溯源}$$

又有：

$$T_{最好} < \mathrm{T}_{多级} < \mathrm{T}_{最坏}$$

综上可得出：

$$T_{最好} < \mathrm{T}_{多级} < \mathrm{T}_{最坏} < \mathrm{T}_{原始溯源}$$

即本文所提多级缓存溯源技术方案在理论上溯源效率要优于原始溯源方法。此外，在保持相同的缓存命中率情况下，多级缓存由于引入了二级缓存结构，对部分查询频率较低的溯源数据只存储其溯源哈希链，因此存储空间占用对比缓存全量数据的普通缓存方法也有了明显的改善，缓解了区块链全节点的存储压力。

4.5 实验对比与分析

本实验环境仍采用单机搭建模拟全节点完成各项测试验证，实验中所采用的单机硬件环境为 Intel(R) Core(TM) i5-7300HQ CPU（2.50 GHz），RAM（8GB），操作系统 Windows 10，使用 Java 语言，借助 Redis 缓存组件实现所提多级缓存结构溯源方法，模拟在全节点中的溯源情况。此外，实现了无缓存时的原始溯源方法与引入普通单级缓存

的溯源方法，通过对比实验测试验证了引入多级缓存后溯源检索效率的提升情况，以及对多级缓存的空间节省和维护开销等情况进行了测试评估。

• 实验 1：溯源效率对比

为探究引入多级缓存后，对比原始溯源方法（即根据数据字段 PreHash 依次向前的溯源方法）区块链的溯源效率优化情况，本实验设置 1000 次溯源查询请求，其中在重复的溯源请求（请求同一 Key）占比分别为 0%，30%，60%情况下，计算出单次溯源耗时平均值，对比原始溯源方法与多级缓存溯源方法的耗时情况，如图 4.5 所示。

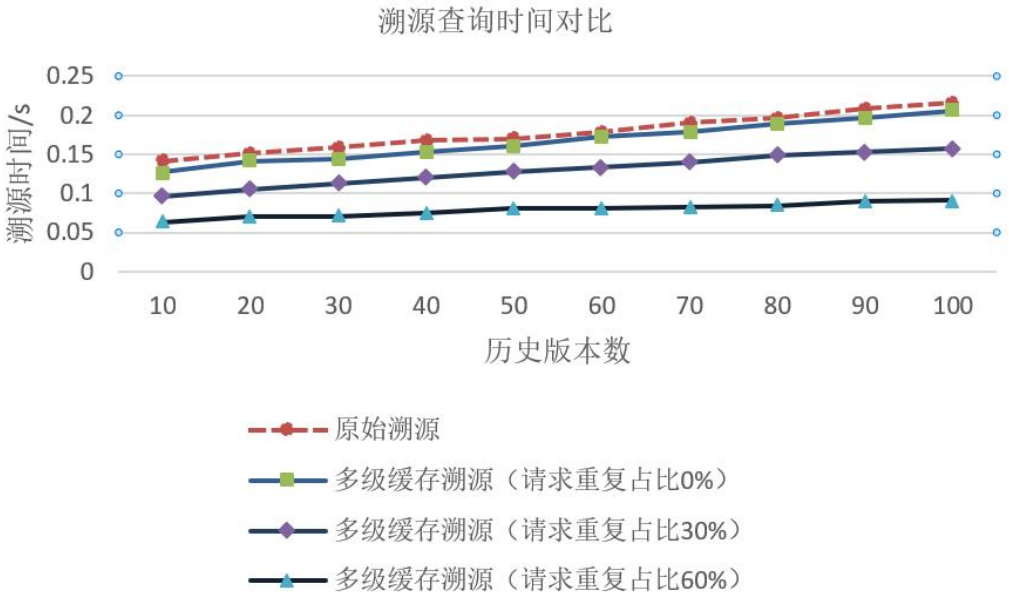

图 4. 5 溯源查询耗时对比图

Fig. 4.5 Comparison Chart of Traceability Query Time

由实验结果可知，当溯源请求重复占比为 0%时，多级缓存溯源方法中缓存无法发挥溯源作用，而是采用原始溯源方法溯源，故溯源效率与原始方法基本无差别，这也对应着多级缓存方法中的溯源最差情况。当请求重复占比上升时，多级缓存溯源发挥作用，溯源的效率明显优于原始溯源方法，且随着溯源请求重复占比的增加，这一优势愈发明显，且基本符合在 4.4 章节中的理论分析情况。验证了本文所提多级缓存方法对全节点溯源效率提升的有效性。

• 实验 2：单/多级结构溯源耗时对比

多级缓存的二级缓存中存储有溯源哈希链路，在节省空间的同时有利于最大限度发挥底层 Level DB 的读数据性能，一定程度上可以提高检索完整溯源链路的效率。本实验首先在多级缓存方案的基础上进行修改，将其中缓存降级部分用直接删除淘汰替换，

以实现普通单级缓存方法。实验设置比例 R 代表不同应用场景中缓存数据的分布情况，其中在多级缓存中，R=一级缓存数据量/数据总量；在单级缓存中，R=缓存数据量/数据总量。对 1000 条不同 Key 标识溯源数据（单条 Key 数据历史版本数为 100）分别进行溯源测试，并计算其单条 Key 数据的平均溯源时间，对比多级缓存方法与普通单级缓存方法的溯源效率情况。

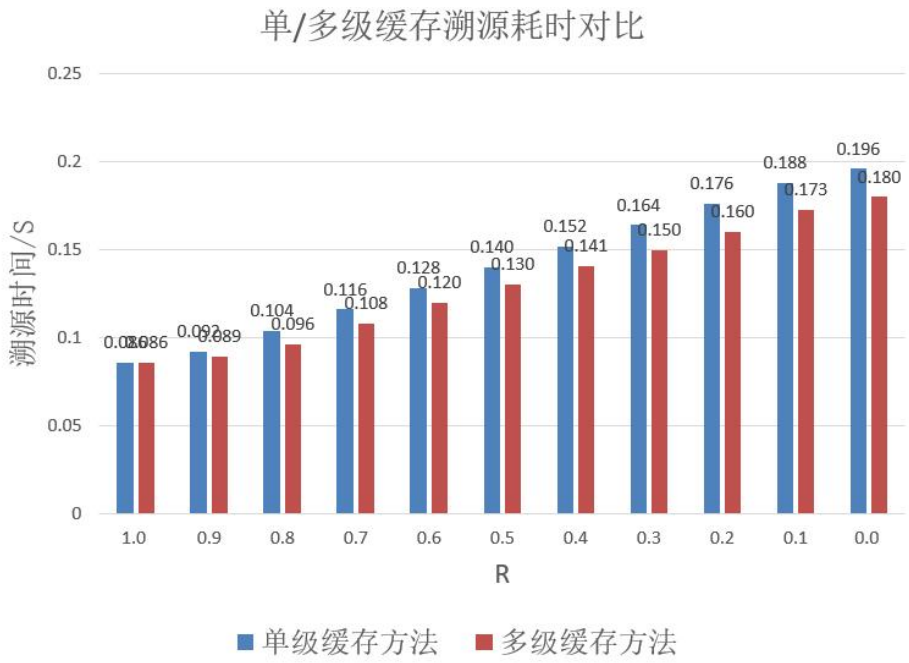

图 4.6 单/多级缓存溯源耗时对比

Fig. 4.6 Comparison of single/multi-level cache traceability time

实验结果如图 4.6 所示，可知，随着 R 的不断减小（即缓存数据占比不断减小），由于缓存命中率的下降导致两种方法溯源平均时间均升高，但多级缓存溯源平均时间上升趋势比普通单级缓存更加平缓，具有着更优的溯源效率，这得益于多级缓存的二级缓存中存储的溯源哈希链路。多级缓存在一级缓存未命中时，可依靠二级缓存中存储的溯源哈希链路，向 Level DB 中溯源数据进行批量查询，与单级缓存在缓存未命中时只能采用原始的依次递归方法查询出完整溯源链路相比，更大程度地发挥出了 Level DB 的读数据性能，缩短了溯源的平均耗时。因此，在不同应用场景中由于溯源查询请求分布不同影响，缓存数据的分布 R 也不尽相同，得益于多层级结构的加持，在相同 R 分布情况下，多级缓存方法相比普通单级缓存方法有着更优的溯源查询效率。

• 实验 3：存储空间开销对比

在比特币系统中，交易数据的平均大小在 250B 左右，使用 SHA-256 算法计算每笔交易哈希值，即数据哈希值大小为固定的 32 字节。因此为消除单笔数据大小对溯源测试的影响，且使实验结果更贴近真实场景，在本实验中设置单笔溯源数据大小为 250B，并使用 SHA-256 算法计算数据哈希值，分别测试对比多级缓存与普通单级缓存在存储不同溯源数据量下所占用存储空间情况，实验结果如图 4.7 所示。

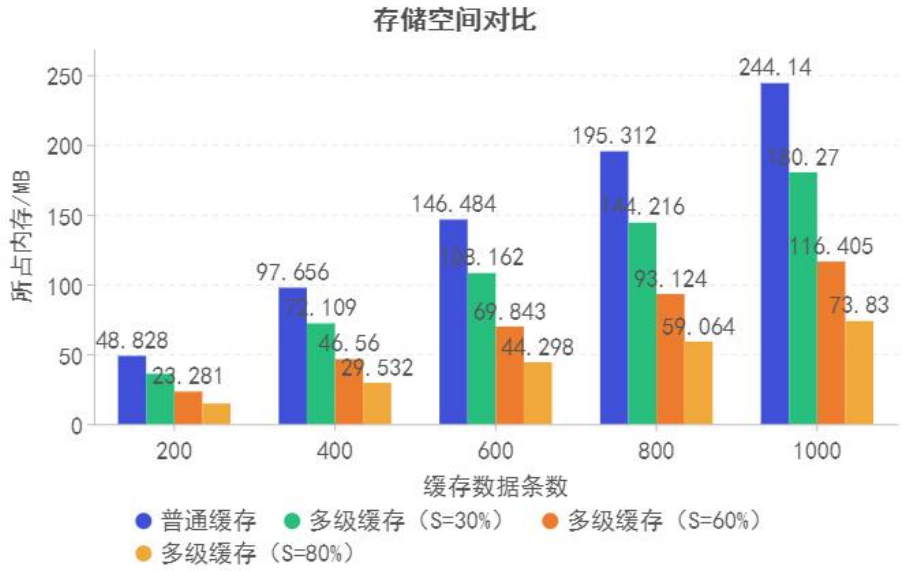

图 4. 7 存储空间消耗对比图

Fig. 4.7 Comparison of storage space consumption

由于多级缓存方法中的内存空间占用由存储数据量和数据分布情况（一，二级缓存占比情况）共同决定，故设置图 4.7 中 S 为多级缓存方案中二级缓存的占比率，即二级缓存中的溯源数据在整个多级缓存中的占比。图 4.7 显示了多级缓存中在数据不同分布 S 下的空间消耗情况，由该图可知，引入多级缓存结构后，由于能对缓存中的溯源数据及时进行淘汰降级处理，使得多级缓存所占内存空间要优于普通单级缓存方法，且随着淘汰降级为二级缓存的溯源数据占比增多，多级缓存方案节省内存占用的优势愈发明显。实验结果验证了多级缓存方法对比普通单级缓存方法在内存空间消耗上的优势，证明了本文所提多级缓存的溯源优化方案在提升溯源效率的同时兼顾了内存开销的降低，降低了全节点的内存存储负担。

• 实验 4：缓存维护代价对比

本技术方案中使用了多级缓存的优化策略，与普通单级缓存方法相比需额外对缓存内容进行定期更新，即缓存淘汰降级过程，为评估该过程所带来的额外消耗，设计本对比实验进一步探究其所耗计算资源情况。实验设计分别向普通单级缓存和本文多级缓存结构中存入 100，200，300...1000 条数据，且设置存入完毕后多级缓存启动一次淘汰降级算法，将一级缓存中的数据降到二级缓存中，以此测试多级缓存淘汰算法的耗时极限，对比二者的耗时情况，评估维护多级缓存代价消耗情况。

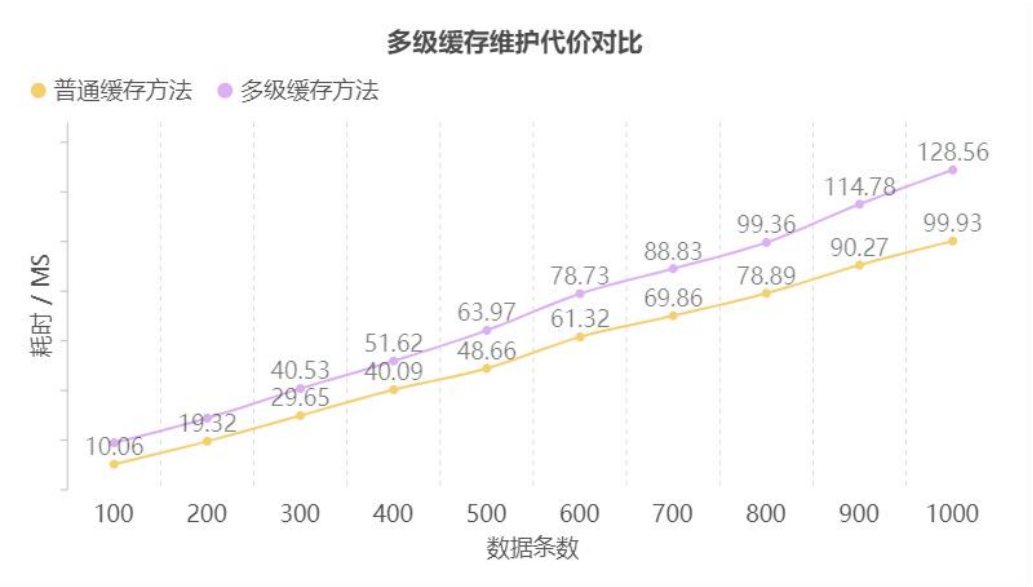

图 4.8 多级缓存维护耗时对比图

Fig. 4.8 Multi-level cache maintenance time comparison chart

实验结果如图 4.8 所示，由该结果可知，随着存入数据量的不断增长，两种方法耗时基本呈线性增长，且多级缓存方案需额外进行淘汰降级过程，因此耗时要高于普通单级缓存方法，但额外开销的耗时基本维持在维护单级缓存耗时的 1/3 以内。此外，本实验测试中将所有一级缓存降级为二级缓存以测试该过程耗时极限，在实际环境中并不是所有的一级缓存中数据都需要降级处理，因此实际耗时开销要更优于这一情况，总体来说，多级缓存结构的维护代价在可接受的范围内，对实际溯源场景具有一定的应用价值。

4.6 本章小结

在本章节中，针对在溯源请求频繁的场景中，如何在不给全节点造成太大负担下，进一步优化提升全节点溯源查询的效率问题，介绍了本文所提的技术方案，即基于多级

缓存的溯源查询优化方案，用高效的内存操作替代了重复低效的磁盘 IO 溯源检索，减轻了对底层存储系统的依赖，提升了溯源查询的效率；同时多级缓存结构降低了全节点维护缓存的内存存储开销，兼顾溯源效率与资源消耗，减轻了全节点负担。最后，经实验验证，本文提出的区块链数据多级缓存溯源查询优化方法能够在较低的内存消耗下减少溯源查询过程中的磁盘 IO 次数，提升数据溯源查询的效率。

5 总结与展望

5.1 研究总结

区块链因其去中心化，不可篡改，查询可验证及数据可溯源等特性，逐渐在数据可信治理领域得到广泛应用。随着应用的深入，如何提升其数据读效率成为了核心问题之一。其中数据内容检索和溯源查询处理是区块链的两个重要数据读取方式，很多工作针对其效率进行了优化，但仍存在着许多不足。在数据内容检索方面，现有的研究方法难以避免采用遍历区块的搜索方式，导致检索效率没有得到根本性的改善。在数据溯源查询方面，为减轻溯源请求频繁场景中底层存储系统的压力，提高溯源效率，现有研究普遍通过引入复杂数据结构的方式进行优化，给全节点带来了过大的维护负担，难以兼顾溯源效率与资源消耗。综上，在面向区块链全节点中内容检索和溯源查询处理两个方面，本文研究工作的贡献主要包括以下两点：

（1）针对内容检索效率低下的问题，本文提出了一种基于以太坊状态树的索引优化模型，使得检索可精准定位目标数据所在区块，避免了遍历区块的搜索过程，经实验验证，相比于现有的仅构建块内索引的优化方法，随着区块链数据量的增长，该索引模型在可接受的索引维护代价内，将内容检索的时间从线性增长降到了对数级别，内容检索在效率和稳定性上均得到了很大改善。此外，受益于状态树索引的全局性特征，该模型可同时提供查询数据存在或不存在证明，进一步提升了查询结果的可信性。

（2）针对在溯源查询请求频繁的场景中，难以兼顾全节点溯源查询效率与资源消耗的问题。本文设计了基于多级缓存的溯源查询优化方案，用高效的内存操作替代了重复低效的磁盘 IO 溯源检索，提升了溯源查询的效率，同时该多级缓存结构降低了全节点维护缓存的内存开销，兼顾溯源效率与资源消耗，减轻了该场景下全节点底层磁盘存储系统的压力。

综上所述，为了改善区块链系统在读数据方面的效率，即数据查询处理性能问题，针对内容检索和溯源查询两个课题，本研究提出了基于以太坊状态树的内容检索优化方案和基于多级缓存的溯源查询优化方案，并设计实验验证了方案的有效性。研究旨在不损失区块链基本特性的前提下，提升区块链全节点内容检索和溯源查询的能力，以拓展区块链在各类应用场景中的适应性，发挥其确保数据可信的核心业务价值。

5.2 应用总结

在本研究中，除设计对比实验对研究方案进行验证外，为进一步测试本文研究成果在真实应用场景中的有效性，我们将两个研究成果方案集成到新华社二级单位（中国图片社）的项目“中国图片区块链知识产权保护平台”中，旨在改善区块链模块图片数据内容检索和图片版权流转数据溯源查询的效率。

该项目借助区块链对图片版权信息等关键数据进行存储，以实现针对图片版权数据的可信查询（查询可验证），产权流转链路追溯等核心业务功能，属数字资产保护应用领域。

参考 BCDB 模型数据规范，项目中区块链所存储的图片版权信息的数据记录结构设计如图 5.1 所示。

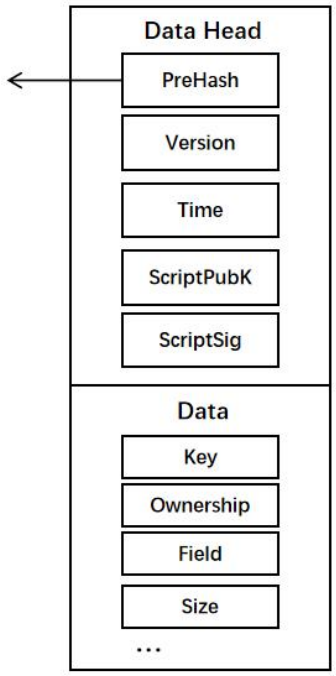

图 5.1 图片版权信息数据结构

Fig. 5.1 Image copyright information data structure

单条图片版权数据记录由 Data 和 Data Head 两部分组成，其中 Data 部分记录图片版权信息具体数据内容，Key 为图片关键字，是图片实体的唯一标识。其余域为图片数据的属性内容，如 Ownership 记录该图片当前所有权归属，Field 为图片所属领域，Size 为图片大小信息等。

数据头 Data Head 中记录该条图片数据记录的元信息，其中 PreHash 指向相同 Key 标识的上一历史版本图片数据记录，以实现图片数据修改轨迹溯源和防篡改验证；Time 是该版本图片记录发布时间；Version 为该条图片数据记录版本号；ScriptPubK 指定拥有该图片数据记录写操作权限的下一拥有者公钥，ScriptSig 是拥有本图片数据记录写操作权限的拥有者的签名，用于发布时验证本图片记录有效性，即在写入数据的时候，先查询是否存在关键字为 Key 的图片数据记录，若存在，则检验当前数据记录的 ScriptSig 是否与前一数据记录的 ScriptPubK 相匹配，只有在匹配的情况下才可认为新发布的该条图片数据记录有效。若 Key 是第一次出现，则将 Prehash 设置为 null。对某一 Key 标识图片数据记录的修改，则通过在新区块中写入新的相同 Key 的图片数据记录，以区块追加的方式实现，所有图片历史记录版本将存在于历史区块中，不可删除。

综上所述，在该项目中数据内容检索和溯源查询定义如下：

内容检索：根据图片数据记录标识关键字 Key 进行内容检索，最后返回该 Key 标识最新版本的图片数据记录。

溯源查询：输入图片数据记录关键字 Key，查询到最新版本后，根据数据头中的 PreHash 进行图片数据溯源，追溯出特定 Key 所标识图片数据记录的所有修改历史版本，即图片版权流转链路。

以上两个过程的具体实现方式均采用本文研究优化方案，因此不再赘述。

最后，经过一段时间的测试记录发现，本文基于以太坊状态树的索引优化模型在数据内容检索效率，查询响应速度，稳定性及查询可信性方面都表现出了更好的性能。另一方面，本文所提多级缓存溯源查询优化方案在以可接受的计算存储代价内，改善了图片版权流转链路溯源查询的效率，且在大量溯源查询请求并发时段，这一优势愈发明显，效率与经济性兼备。即本文所提优化方案在真实应用场景中的测试表现基本符合预期，有效性得到了进一步验证。

5.3 未来展望

最后，值得一提的是，虽然本文对全节点中数据内容检索和溯源查询进行了一定程度的优化，但受时间和自身水平等原因所限制，在本研究方案中还存在着很多不足之处和进一步的研究空间。例如，在溯源查询优化中，关于多级缓存淘汰更新的最佳时机及最佳方案的探究还存在着很大的研究空间，如尝试将定时更新策略与惰性更新策略应用到不同场景中去，设计动态调整机制，进一步合理分配全节点的计算内存资源。此外，本研究着重于优化全节点中的检索方案，在未来，可将研究重点转向如何将全节点功能

拆分，即分离出专门的查询节点以及共识节点，分工协作以实现能满足更高数据查询要求的区块链网络。

参 考 文 献

[1] Nakamoto S. Bitcoin: A peer-to-peer electronic cash system[J]. Decentralized Business Review, 2008: 21260.

[2] Buterin V. A next-generation smart contract and decentralized application platform[J]. white paper, 2014, 3(37).

[3] Tikhomirov S. Ethereum: state of knowledge and research perspectives[C]//International Symposium on Foundations and Practice of Security. Springer, Cham, 2017: 206-221.

[4] Cachin C. Architecture of the hyperledger blockchain fabric[C]. Workshop on distributed cryptocurrencies and consensus ledgers. 2016,310:4.

[5] Brown R G, Carlyle J, Grigg I, et al. Corda: An Introduction[R/OL]. (2016,08,01) [2021,04,07].dhttps://www.researchgate.net/publication/308636477_Corda_An_Introduction.

[6] LENG K, BI Y JTNG L, et al. Research on agricultural supply chain system with double chain architecture based on blockchain technology[J]. Future Generation Computer Systems, 2018, 86:641-649.

[7] OZTURK C, YILDIZBA§I A. Barriers to implementation of blockchain into supply chain management using an integrated multi-criteria decision-making method: a numerical example[J]. Soft Computing, 2020, 24:14771 14789.

[8] BERGER C, PENZENSTADLER B, DROGEHORN O. On using blockchains for safety-critical systems[C]//Proceedings of the 4th International Workshop on Software Engineering for Smart Cyber-Physical Systems. 2018: 30-36.

[9] SANI A S, YUAN D, BAO W, et al. Xyreum: A high-performance and scalable blockchain for iiot security and privacy[C]//39th IEEE International Conference on Distributed Computing Systems. 2019: 1920-1930.

[10]JIN H, XU C, LUO Y, et al. Toward secure, privacy-preserving, and interoperable medical data sharing via blockchain[C]//25th IEEE International Conference on Parallel and Distributed Systems. 2019: 852-861.

[11]XIAO Z, LI Z, LIU Y, et ai. Emrshare: A cross-organizational medical data sharing and management framework using permissioned blockchain[C]//24th IEEE International Conference on Parallel and Distributed Systems. 2018: 998-1003.

[12]MCBEE M P, WILCOX C. Blockchain technology: principles and applications in medical imaging[J]. Journal of Digital Imaging (Online), 2020.

[13]ZHAO J, ZONG T, XIANG Y, et al. Robust blockchain-based cross-platform audio copyright protection system using content-based fIngerprint[C]//International Conference on Web Information Systems Engineering. 2020: 201-212.

[14]MENG Z, MORIZUMIT, MIYATA S, et al. Design scheme of copyright management system based on digital watermarking and blockchain[C]//42nd IEEE Annual Computer Software and Applications Conference. 2018: 359-364.

[15]Seyed Mojtaba Hosseini Bamakan,Amirhossein Motavali,Alireza Babaei Bondarti. A survey of blockchain consensus algorithms performance evaluation criteria[J]. Expert Systems With Applications,2020,154:

[16]Kiayias A, Russell A, David B, et al. PPCoin: Peer-to-Peer Crypto-Currency with Proof-of-Stake[J]. self-published paper, 2012, 19:1.

[17]Vasin P. BlackCoin’s Proof-of-Stake Protocol v2[EB/OL]. (2014,01,08)[2021,03,25]. https://blackcoin.org/blackcoin-pos-protocol-v2-whitepaper.pdf.

[18]Larimer D.Delegated proof-of-stake[EB/OL]. (2014,04,03) [2021,03,25]. https://wenku.baidu.com/view/5c3d6112336c1eb91a375d9b.html.

[19]Sompolinsky Y, Zohar A. Secure High-Rate Transaction Processing in Bitcoin[C]. Conference on Financial Cryptography and Data Security, Springer: Berlin/Heidelberg, Germany, 2015, 8975: 507-527.

[20]刘志杰,张方国,田海博.基于ECDLP的工作量证明方案设计[J]. 密码学报, 2020, 7(04):95-105.

[21]Eyal I, Gencer A E, Sirer E G, et al. Bitcoin-NG: a scalable blockchain protocol[C]. Symposium on Networked Systems Design and Implementation, Santa Clara, CA, USA, 2016: 45-59.

[22]Yu B, Liu J K, Nepal S, et al. Proof-of-QoS: QoS based blockchain consensus protocol[J]. Computers & Security, 2019, 87:101580.1-101580.13.

[23]Li Y, Wang Z, Fan J, et al. An Extensible Consensus Algorithm Based on PBFT[C]. 2019 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery, Guilin, China, 2019:17-23.

[24]Tong W, Dong X and Zheng J. Trust-PBFT: A PeerTrust-Based Practical Byzantine Consensus Algorithm[C]. 2019 International Conference on Networking and Network Applications (NaNA), Daegu, Korea (South). 2019:344-349.

[25]涂园超,陈玉玲,李涛,等.基于信誉投票的PBFT改进方案[J].应用科学学报,2021,39(01):79-89.

[26]Bentov I, Lee C, Mizrahi A, et al. Proof of Activity: Extending Bitcoin’s Proof of Work via Proof of Stake[J]. IACR Cryptology ePrint Archive, 2014, 42(3): 34-37.

[27]吴梦宇,朱国胜,吴善超.基于工作量证明和权益证明改进的区块链共识算法[J].计算机应用, 2020, 40(08):2274-2278.

[28]LI Y ZHENG K, YAN Y et al. Etherql: a query layer for blockchain system[C]// 22nd International Conference on Database Systems for Advanced Applications. 2017: 556-567.
[29]Mongodb[EB/OL]. https://www.mongodb.org.
[30]Chainsql[EB/OL]. http://www.chainsql.net/.
[31]Postgresql[EB/OL]. https://www.postgresql.org/.
[32]PENG Y DU M, LI F, et al, Falcondb: Blockchain-based collaborative database [C]//Proceedings of the 2020 ACM International Conference on Management of Data. 2020: 637-652.
[33]SCHWARTZ D, YOUNGS N, BRITTO A, et al. The ripple protocol consensus algorithm[R]. White Paper, 2014.
[34]Mysql 8.0 reference manual[EB/OL], https://dev.mysq 1.com/doc/refman/8.0/en/.
[35]ZHANG Y KATZ J, PAPAMANTHOUC. Integridb: Verifiable sql for outsourced databases[C]//Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security. 2015: 1480-1491.
[36]BUCHMAN E, KWON J, MILOSEVIC Z. The latest gossip on bft consensus[Z], arXiv preprint arXiv: 1807.04938, 2018.
[37]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]. Journal of Software, 2019, 9: 2655-2670.
[38]Zhang C, Xu C, Xu J, 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). IEEE, 2019: 842-853.
[39]焦通,申德荣,聂铁铮,寇月,李晓华,于戈.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(09):2671-2685.
[40]郑浩瀚,申德荣,聂铁铮,寇月.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.
[41]Abdennebi A, Kaya K. A Bloom Filter Survey: Variants for Different Domain Applications[J]. arXiv preprint arXiv:2106.12189, 2021.
[42]You Y，Kong L，Xiao z，et al. Hybrid indexing scheme supporting blockchain transaction tracing [J]. Computer Intergrated Manufacturing System8，2019，25(4)：978-984.
[43]Xing Xiaogang,Chen Yuling,Li Tao,Xin Yang,Sun Hongwei. A blockchain index structure based on subchain query[J]. Journal of Cloud Computing,2021,10(1).
[44]刘炜,王栋,佘维,潘恒,宋轩,田钊.一种面向区块链溯源的高效查询方法[J].应用科学学报,2022,40(04):623-638.
[45]宋杰,那美玉,张彭奕,郭朝鹏. 一种基于区块索引结构的数据溯源查询方法[P]. 中国：CN112765181B,2021-12-03.

[46]RUAN P, CHEN G, et al. Fine-grained, secure and efficient data provenance on blockchain systems[J]. Proceedings of the VLDB Endowment, 2019, 12(9):975- 988.

[47]Xu Li Da,Lu Yang,Li Ling. Embedding Blockchain Technology Into IoT for Security: A Survey[J]. IEEE INTERNET OF THINGS JOURNAL,2021,8(13).

[48]Kreß Fabian,Sidorenko Vladimir,Schmidt Patrick,Hoefer Julian,Hotfilter Tim,Walter Iris,Harbaum Tanja,Becker Jürgen. CNNParted: An open source framework for efficient Convolutional Neural Network inference partitioning in embedded systems[J]. Computer Networks,2023,229.

[49]Nadal Sergi,Jovanovic Petar,Bilalli Besim,Romero Oscar. Operationalizing and automating Data Governance[J]. Journal of Big Data,2022,9(1).

[50]McDermott Kevin. Allied Systems: Data governance challenges and the opioid crisis[J]. Journal of Information Technology Teaching Cases,2022,12(1).

[51]Zhihong Liang, Yuxiang Huang, Zechun Cao, Tiancheng Liu,Yuehua Wang. Creativity in Trusted Data: Research on Application of Blockchain in Supply Chain[J]. International Journal of Performability Engineering,2019,15(2).

[52]Kumi Sandra,Lomotey Richard K.,Deters Ralph. A Blockchain-based platform for data management and sharing[J]. Procedia Computer Science,2022,203.

[53]American CryptoCoupon Inc.; "Data Management System Using Multiple Blockchains" in Patent Application Approval Process (USPTO 20200136829)[J]. Information Technology Newsweekly,2020.

[54]Alibaba Group Holding Limited; Patent Application Titled "System And Method For Blockchain-Based Data Management" Published Online (USPTO 20200120084)[J]. Internet Business Newsweekly,2020.

[55]Services Computing; Investigators from Peking University Release New Data on Services Computing (Decentralized Services Computing Paradigm for Blockchain-based Data Governance: Programmability, Interoperability, and Intelligence)[J]. Computer Technology Journal,2020.

[56]ALTR; ALTR Partners with Sirius to Deliver Blockchain-based Data Governance and Security as a Managed Service[J]. Journal of Engineering,2019.

[57]Wattana Viriyasitavat,Danupol Hoonsopon. Blockchain characteristics and consensus in modern business processes[J]. Journal of Industrial Information Integration,2019,13.

[58]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(04):481-494.

[59]Zutshi Aneesh,Grilo Antonio,Nodehi Tahereh. The Value Proposition of Blockchain Technologies and its impact on Digital Platforms[J]. Computers & Industrial Engineering,2021(prepublish).

[60]Diyong Wu,Wei Wei,Cheng Li,Yuan Tang. Hyperledger Fabric Transaction Processing Process Optimization[J]. Journal of Research in Science and Engineering,2023,5(3).

[61]Stodt Fatemeh,Reich Christoph. Introducing a Fair Tax Method to Harden Industrial Blockchain Applications against Network Attacks: A Game Theory Approach[J]. Computers,2023,12(3).

[62]Chen Chao,Huang Hao,Zhao Bin,Shu Desheng,Wang Yu. The Research of AHP-Based Credit Rating System on a Blockchain Application[J]. Electronics,2023,12(4).

[63]Sethaput Vijak,Innet Supachate. Blockchain application for central bank digital currencies (CBDC).[J]. Cluster computing,2023.

[64]Kuanchin Chen, Damodar Y. Golhar, Snehamay Banerjee. Blockchain applications and challenges for supply chain and Industry 4.0: a literature review[J]. International Journal of Applied Decision Sciences,2023,16(1).

[65]Kiu M. S.,Chia F. C.,Wong P. F.. Exploring the potentials of blockchain application in construction industry: a systematic review[J]. International Journal of Construction Management,2022,22(15).

[66]Kalajdjieski Jovan,Raikwar Mayank,Arsov Nino,Velinov Goran,Gligoroski Danilo. Databases fit for blockchain technology: A complete overview[J]. Blockchain: Research and Applications,2023,4(1).

[67]王千阁,何蒲,聂铁铮,申德荣,于戈.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.

攻读硕士学位期间发表学术论文及专利情况

1　基于状态树的链上数据高效可信查询索引模型及方法．XX，**黄笠煌**，陈志奎，于硕. 重庆大学学报（自然科学版）. 主办单位：重庆大学。（本硕士学位论文第三章）

2　一种区块链数据高效可信索引方法与流程. XX，**黄笠煌**，李强，李昕欣. 2022-02-25，中国，专利公开号：CN114020737A（实质审查，本硕士学位论文三章）

3　一种区块链数据溯源查询优化方法．XX，**黄笠煌**，秦昌媛，邹寅星．2023-01-06，中国，专利公开号：CN115952195A（实质审查，本硕士学位论文第四章）

4　一种 CNFS 协议中区块链节点的分组多链异步共识方法．XX，王国良，黄笠煌，陈志奎．2022-04-12，中国，专利公开号：CN113141414B（已授权，本硕士学位论文第一章）

致　　谢

三年时光如白驹过隙，一转眼，研究生生涯即将结束。回首过去的三年，也是目前为止我人生中最重要的三年。过程中，不管是在自身专业上还是心态上，都产生了巨大的转变与成熟。记得刚入校时，陈志奎老师对我们说，他自己理解的幸福就是不断做好自身的积累，无论是在工作上还是生活上，都能时刻做到游刃有余。这句话我一直谨记着，提醒自己在平时空闲时间中不要虚度光阴，耐下心来雕琢自己，终身学习。在过去三年中，身边的老师，同学，家人给予了我许多关键的帮助，因此，在论文完成之际，在此向所有帮助过我的人致以由衷的感谢。

首先，要感谢我的导师 XX 老师，实验室的陈志奎老师和于硕师姐。X 老师总是能够在我科研受挫，心情沮丧时给予我充分的肯定和关键的建议，这给了我极大的继续下去的自信心，同时尊重我的想法，让我在科研的道路上能自由地寻找到自己的热爱和所长，这一点我十分珍惜和感恩。陈老师生活中平易近人，但从他的问候中能感受到他心系每一个学生，也时常在实验室紧张的科研氛围中感受到他带来的温暖。科研刚起步时，是于硕师姐不断和我们分享自己的观点见解，带着我们逐一点评各典型文章的研究思路，这帮助我们小白在茫茫学海中慢慢找到了一些方向，她对我论文写作上的指导让我少走了很多弯路，过程中她严谨科学的学术精神和勇于向上的精神，同样使我敬佩不已，我也会将这份精神带到之后的生活中去。在此，由衷地感谢我的导师 XX 老师，实验室的陈志奎老师和于硕师姐对我的关怀与耐心指导。

其次，我要感谢身边的同学和朋友，平日里的每一次闲谈嬉戏，都给我带来了印象深刻的欢乐，这些点滴的记忆碎片支撑着我克服了生活学习中的困难。不管今后是否聚少离多，我都会清楚地记得这份友谊，感恩并珍惜着那些我们互相陪伴着的日子。

此外，我要感谢远在千里的父母亲人，在我读书期间不断嘘寒问暖，是你们一直默默地付出，才可能有今天茁壮成长的我，谢谢你们。

最后，感谢大连理工大学，值此疫情期间，为我们创造了最安全的环境，让我们得以专心学习，祝愿大连理工大学再创辉煌，软件学院蒸蒸日上！

大连理工大学学位论文版权使用授权书

本人完全了解学校有关学位论文知识产权的规定，在校攻读学位期间论文工作的知识产权属于大连理工大学，允许论文被查阅和借阅。学校有权保留论文并向国家有关部门或机构送交论文的复印件和电子版，可以将本学位论文的全部或部分内容编入有关数据库进行检索，可以采用影印、缩印、或扫描等复制手段保存和汇编本学位论文。

学位论文题目：_____

作 者 签 名 ：_____ 日期：_____年____月____日

导 师 签 名 ：_____ 日期：_____年____月____日