溯源系统区块链数据快照生成与高速检索算法

汪继辉[a]，余宏杰[*b]
（安徽科技学院 a. 智能制造学院，安徽 凤阳 233100；b. 信息与网络工程学院，安徽 蚌埠 233000）

摘 要：食品安全的快速溯源是减少食品危害的重要手段，目前针对该问题的研究仍不能满足高速且准确的溯源需求。本文将传统数据库技术和现代区块链技术相结合，建立区块链快照的结构化数据，既保持了区块链的数据安全特性，又能利用关系数据库具备快速检索的技术框架，从而让检索数据的性能提高2～10倍。遴选百万级的商品数据进行模拟溯源和性能评测，实验结果表明了区块链快照技术和高速检索算法的合理性和可行性。

关键词：溯源；区块链；快照；高速检索
中图分类号：TP312 **文献标志码**：A **文章编号**：1673－0143(2025)01－0070－12
DOI：10.16389/j.cnki.cn42－1737/n.2025.01.009

Blockchain Data Snapshot Generation and High-speed Retrieval Algorithm for Traceability System

WANG Jihui[a], YU Hongjie[*b]
(a. College of Intelligent Manufacturing, Anhui Science and Technology University, Fengyang 233100, Anhui, China; b. College of Information and Network Engineering, Anhui Science and Technology University, Bengbu 233000, Anhui, China)

Abstract: Rapid traceability of food safety is an important means to reduce food hazards. Current research on this issue still cannot meet the demand for high－speed and accurate traceability. In this paper, traditional database technology and new blockchain technology were combined to establish structural data of blockchain snapshots, which not only maintained the data security characteristics of blockchain but also utilized the technical framework of relational databases for high－speed retrieval, it improved the performance of querying blockchain data by 2 to 10 times. A million－level commodity data was selected for simulation and performance evaluation. The experimental results showed the rationality and feasibility of snapshot technology and high－speed retrieval algorithms.

Key words: traceability; blockchain; snapshot; high－speed retrieval

收稿日期：2023－09－01
基金项目：安徽省科学技术厅重点研究与开发计划项目(202204c06020065)
作者简介：汪继辉(1976—)，男，硕士生，研究方向：区块链应用。
*通信作者：余宏杰(1970—)，男，教授，博士，研究方向：智能计算、模式识别。E－mail：yhj70@mail.ustc.edu.cn

0　引言

目前，在食品溯源领域有传统的基于RFID硬件相结合的溯源技术[1]和基于物联网的中央式溯源系统等常规方法[2-3]，其中采用RFID电子标签对食品中的关键物料进行追踪，具有技术门槛高、数据加密后不易被篡改等优点，但部署成本高，技术难度大，不易被大范围采用。近年来，有厂商自研基于中央数据库的溯源本地化方案，但如何做到数据公正性，让消费者相信数据的真实性仍有待解决[4]。近些年国内外围绕区块链框架的溯源技术研究发展迅速，一些区块链技术溯源科研成果投入应用。2016年6月，国内基于区块链打造的溯源平台“码上放心”上线，被广泛用于药品溯源。同时，基于蚂蚁链（商业化区块链平台）的食品安全溯源商业化方案也逐渐被采用。在国际上，IBM于2018年研发基于区块链的IBM Food Trust溯源商用产品，沃尔玛等大型商超已经部署该产品的应用[5]。区块链技术溯源方案因具备不易被篡改的特性，在食品溯源方面逐渐被广泛接受，但区块链技术溯源方案也存在查询性能和速度等方面的问题。

由于区块链链表结构的分布式数据存储架构[6]，导致交易性能较传统中央式存储架构低。目前的研究主要集中于采用微调共识机制[5,7]以提高上链数据的性能，即通过提高吞吐量从而不断突破区块链的交易性能瓶颈。区块链技术框架的特性，即在高并发交易时数据查询较传统业务会更快进入性能瓶颈，且高度依赖于节点的计算性能。在实际应用中，溯源方案除了能够提供良好安全的技术框架之外，应更加关注终端消费者使用体验，能让终端消费者快速查询到商品的源头，同时能证实数据的公正性[8-9]。所以有必要在现有基于区块链技术架构的溯源方案上，最大幅度提高区块链数据查询速度，以减少溯源查询响应时间。

针对区块链数据量增加导致查询效率低的问题，国内外都有不少研究[10-11]。Morishima等[12]利用GPU较高的计算能力提出通过GPU加速区块链搜索，改善了区块链搜索性能，但是GPU价格昂贵且能耗高。国内有基于混合索引的区块链系统的查询优化的方法[13-15]等，从一定程度提高了区块链查询速度。本文结合新旧技术的特长，利用数据库查询速度快的优点，通过足够的冗余数据来存储关键信息，克服区块链数据检索慢的缺点，减少查询响应时间，使基于区块链技术的溯源系统更容易落地，也更容易被终端消费者接受。

1　快照存储与高速查询溯源原理

结合关系数据库查询效率高、易于优化的特点，基于关系数据库技术建立区块链的关键数据本体，即为区块链关键数据快照。快照既保存了区块链关键数据的完整性，又能存储数据之间的链式关系。利用冗余数据将区块链多级深度数据的关系链转换为一维数据，即随机存储格式。理论上基于区块链数据查询，如果从起始点到目标数据需要经过n个节点查询，其时间复杂度为n。经过快照变化后的数据查询，理论上只需要1次查询即可，时间复杂度为1，查询效率提高n倍，且查询性能改善幅度与经过的节点数成正比。

如图1所示的商品A，经过层层溯源得出A的原料包含B0、B1，B1的原料包含C0、C1，B0、C0和C1为不包含其他原材料的材料。A为第一层，B为第二层，C为第三层，依此类推。

以图1为例，A为被溯源的商品，定义为根节点，其深度定义为1；B、C为原料商品，B0和B1的深度为2；C0和C1的深度为3。B1定义为普通节点可被进一步溯源，B0、C0、C1定义为叶子节点（即终节点，不能进一步溯源）。同时，定义B1为C1的叶节点，C1为B1的子节点。

区块链的存储容量在几KB到1M之间，且数据需要打包存储。溯源商品关系链存储于区块链时需考虑以下因素：（1）只存储关键数据，如商品ID；（2）溯源商品的底层商品（即原料）需打包

后存储,满足防篡改需求;(3)可利用区块之间链式关系来查询被溯源商品的所有底层原料商品。将图1中所示的逻辑关系存储于区块链,至少需要3个区块来存储并记录逻辑关系,即为区块1:A/B1、A/B0;区块2:B1/C1、B1/C0;区块3:C1、C0。因区块链系统出块的随机性,区块1、区块2、区块3实际上不连续可能性较大,所以查询A的溯源关系链需遍历区块数不小于3。在实际业务场景中,因技术上无法直接跳跃到目标区块,需遍历经过的全部区块,耗时较多。

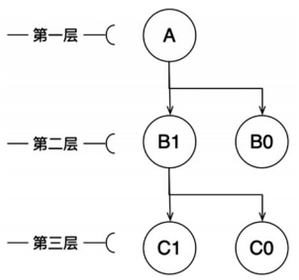

图1 溯源商品示意图

Fig. 1 Traceability product schematic diagram

1.1 链表快照数据

为降低查询响应时间,考虑将链上数据的逻辑关系映射成关系数据并存储于数据库的二维表中。本研究中将数据做二维变换存储于关系数据库中,通过创建表索引,可明显提高查询速度,即为链表快照数据,见表1。

表1 链表快照示例

Tab. 1 Linked list snapshot example

Key	Value	Next Key
1	A	2, 3
2	B0	
3	B1	4, 5
4	C0	
5	C1	

分析表1中的二维数据结构可以发现,链表快照数据有下列优点:(1)查找下一个节点可直接到达,无需经过无用的区块链节点;(2)通过增加更多字段用于存储商品的附加信息(例如序列号、生成日期、有效期等),可节省跨表查询附加信息的时间;(3)创建表索引能大幅度提高查询速度。

1.2 随机存储快照数据

链表适用范围是大量插入数据和删除数据的场景。对于区块链数据,存储后并无插入和删除数据的需求,故二维化的链表结构不是最佳。

针对高频数据查询的应用场景,随机存储的数据结构是最快的。例如,一维、多维数组和Hash map等都是随机数据结构。因数据库中没有数组、Hash map等直接对应的字段类型,需对图1中的关系再进行数据变换,使用"/"分割溯源商品链的溯源商品和原料商品,越靠近顶层的溯源关系越靠前排列,经变换后得到数据结构:

A/B0、A/B1、A/B1/C0、A/B1/C1。

因底层(原料)商品先上链存储,在后续数据上链时,能自动形成并保存溯源关系,经适当变换后生成溯源关系链数据存储于数据库中。使用JSON格式存储数据,理论上可保存到各个子节点的溯源关系链数据,当顶点的商品上链时就能自动生成包含所有子节点的溯源关系链数据并存储。对于A(根节点)存储全部的静态溯源链关系,本文定义为随机存储快照,见表2。表2中所示的二维数据有下列优点:(1)A节点(根节点)包含了所有的溯源关系,查询A的全部溯源

关系仅需1次查询；(2)采用JSON数据格式的数组结构来模拟一维数组存储溯源关系，无需递归或深度查询；(3)关系链使用“/”作为分割域，前后关系即为溯源关系，可快速分解。

表2　随机存储快照示例

Tab. 2　Random access memory snapshot example

Key	Value	Next Key	Childs
1	A	2，3	[{A/B0}，{A/B1}，{A/B1/C0}，{A/B1/C1}]
2	B0		[{B0}]
3	B1	4，5	[{B1/C1}，{B1/C0}]
4	C0		[{C0}]
5	C1		[{C1}]

2　链表和随机存储快照数据生成

通过模拟数据上链的过程，预生成商品数据和溯源关系链存储于关系数据库中，基于模拟数据生成链表和随机存储数据快照，并比较2种快照数据生成方式在时间复杂度和空间复杂度等关键指标上的差异。

2.1　商品和原料商品数据生成

因溯源系统中商品溯源深度一般不超过10层，本文将商品溯源深度最多定义为10层作为前提条件，以满足实际溯源业务的需求。定义如表3所示商品的基本属性，用于商品和原料商品生成。

表3　商品属性定义

Tab. 3　Product table definition

属性名称	类型	用途
标识符/id	整数型	主键
商品名称/Name	字符串型	商品、原料商品
深度/Depth	整数型	1表示顶层商品；≥1表示原料商品
序列号	字符串型	商品序列号，长度为10个字符，不区分大小写英文和数字组合

注：深度(Depth)指用于不同深度溯源的原料商品，例如Depth＝2的商品，用于Depth＝1的商品的第二层溯源的原料商品。

2.2　链表快照数据生成

模拟溯源商品上链[9]的数据解析，上链数据示意如图2所示，数据上链后可转换为商品溯源树，结构如图3所示。

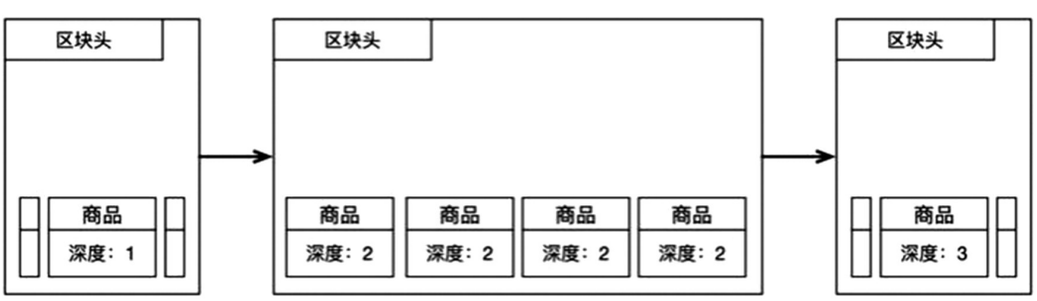

图2　上链数据示意图

Fig. 2　Chain data diagram

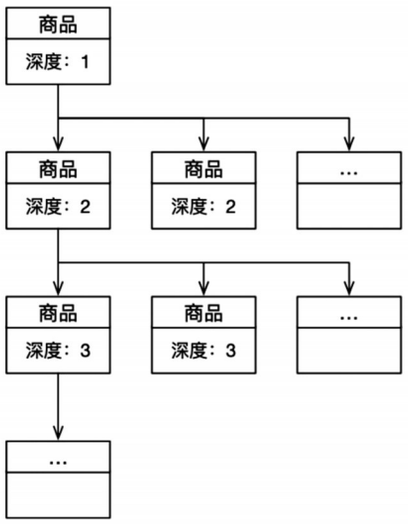

图3 商品溯源树示意图

Fig. 3 Product traceability tree diagram

对商品溯源树进行变换,假定每个普通节点只包含一个叶结点和子结点,得到如图4所示的链表结构。该链表结构可变换为用于存储的链表快照表结构,见表4。

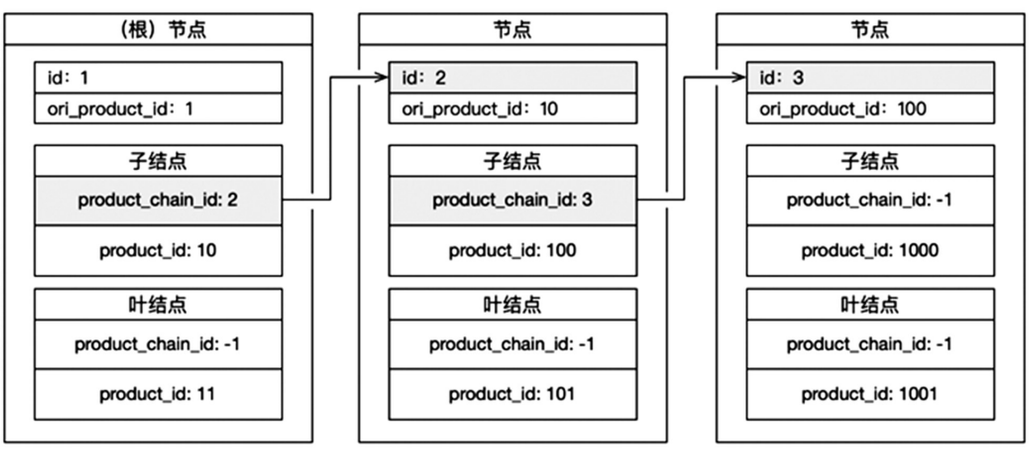

图4 链表结构图

Fig. 4 Linked list structure diagram

表4 链表快照表结构

Tab. 4 Linked list snapshot table structure

属性名称	类型	用途
标识符/id	整数型	主键
被溯源商品标识符/ori_product_id	整数型	
溯源链表	字符串型	下一层链表定义

溯源链表是由商品标识符组成的集合,使用JSON格式,每个单元包含2个元素,即商品链标识符和商品标识符。其中,商品链标识符(product_chain_id)为下一层溯源链的标识符,当product_chain_id = −1时,表示本溯源商品为叶子节点,无需继续溯源;当product_chain_id > 0时,表示本溯源商品为普通节点,需继续溯源。商品标识符(product_id)为Product表的标识符,可根据id到Product表中查询对应的商品信息。

根据表4中链表快照结构可以得到对应的链表快照表,见表5。其中,parent_product_id_set

字段采用JSON格式封装，链表快照数据有下列优点：① 元素逻辑结构清晰，易读性高；② 原生语言支持快速解析JSON格式数据，代码实现更加简洁。

表5　链表快照表

Tab. 5　Linked list snapshot table

id	ori_product_id	parent_product_id_set
1	1	[{"product_chain_id": 2, "product_id": 10}, {"product_chain_id": −1, "product_id": 11}]
2	10	[{"product_chain_id": 3, "product_id": 100}, {"product_chain_id": −1, "product_id": 101}]
3	100	[{"product_chain_id": −1, "product_id": 1000}, {"product_chain_id": −1, "product_id": 1001}]

生成链表快照数据步骤（递归算法[10]）及伪代码如下：

1）将生成根节点的全部溯源链表关系分解为生成每个节点和下层的原料商品，该过程可归纳为一个递归过程，选用递归算法更容易实现。

2）当前节点的深度 ≥ 溯源深度时，即满足终止条件，生成叶子节点；否则调用递归算法并等待返回后生成普通节点。

```
1.  Function generateProductChainListSnapShot ( Product,  MaxNodeNum, MaxLeafNum, maxDepth, Prod-
cutListByDepth)
2.    If  Product. Depth ≤ maxDepth Then return
3.    Get nextDepthProcuctList From ProdcutListByDepth Where Product. Depth = Product. Depth + 1
4.    Make MetaProductList entity
5.    For step to from 1 to MaxNodeNum + MaxLeafNum
6.      Get random Product From nextDepthProcuctList into NewProduct
7.      Make MetaProduct entity
8.      Set MetaProduct. [product_chain_id, product_id] = [-1, NewProduct. id]
9.      If Step  less than or equal MaxNodeNum Then
10.         Set ProductChain = generateProductChain(NewProduct, maxDepth, ProdcutListByDepth)
11.         If ProductChain NotEqual NULL ProdcutChain. parent_prodcut_id_set NotEqual NULL Then
12.           Set MetaProduct. [product_chain_id, product_id]= ProductChain. [id, ori_product_id]
13.         End If
14.     End If
15.     Add MetaProduct into MetaProductList
16.   End For
17.   Make ProductChain entity
18.   Set Product. id into ProductChain. ori_product_id
19.   Set MetaProductList into ProductChain. parent_product_id_set
20.   return ProductChain
21. End Function
```

设置下列前提条件进行分析：① 每个顶层商品的溯源深度为n；② 除顶层外每次溯源节点包含1个叶子节点和1个普通节点。可推导出链表快照数据生成时间复杂度为

$$T(n)\begin{cases}1 & n=1,\\ T(n-1)+1 & n>1。\end{cases} \tag{1}$$

进一步推导可得

$$T(n)=2^{n-1}-1=O(2^n)。 \tag{2}$$

每个节点存储2个节点信息(1个普通节点和1个叶子节点),可推导出空间复杂度,即

$$S(n)=2n=O(n)。 \tag{3}$$

2.3 随机存储快照数据生成

随机快照数据需解决节点id能直接查询到所有溯源链信息,其数据结构见表6,根据表6中的数据结构生成随机存储快照表,见表7。

表6 随机存储快照表结构

Tab. 6 Random access snapshot table structure

属性名称	类型	用途
标识符/id	整数型	主键
被溯源商品标识符/ori_product_id	整数型	
溯源随机存储	字符串型	

表7 随机存储快照表

Tab. 7 Random access snapshot table

id	ori_product_id	parent_product_id_set
1	1	[{"product_id": "1/10"}, {"product_id": "1/10/100“}, {"product_id": "1/10/100/1000"}, {"product_id": "1/10/100/1001"}, {"product_id": "1/10/101"}, {"product_id": "1/11"}]
2	10	[{"product_id": "10/100“}, {"product_id": "10/100/1000"}, {"product_id": "10/100/1001"}, {"product_id": "10/101"}]
3	100	[{"product_id": "100/1000"}, {"product_id": "100/1001"}]

生成随机存储快照仍采用递归算法实现,伪代码如下:

```
1. Function generateProductChainArraySnapshot (Product, maxDepth, ProdcutListByDepth)
2.    If Product. Depth ≤ maxDepth Then return
3.    Get nextDepthProcuctList From ProdcutListByDepth Where Product. Depth = Product. Depth + 1
4.    Make MetaProductList entity
5.    For step from 1 to 2
6.       Get random Product From nextDepthProcuctList into NewProduct
7.       Make MetaProduct entity
8.       Set MetaProduct. [product_chain_id, product_id] = [-1, NewProduct. id]
9.       If step Equal 1 Then
```

```
10.        Set ProductChain = generateProductChain（NewProduct，maxDepth，ProdcutListByDepth）
11.        If ProductChain NotEqual NULL ProdcutChain. parent_prodcut_id_set NotEqual NULL Then
12.          Set MetaProduct.［product_chain_id，product_id］= ProductChain.［id，ori_product_id］
13.        End If
14.      End If
15.      Add MetaProduct into MetaProductList
16.    End For
17.    For MetaProduct in MetaProductList
18.      Make HashMap<String，Object> mapProductId entity with Key，Value =“product_id”，Product. id+
“/”+MataProdcut. product_id
19.      Make ArrayList arrayList entity with mapProductId
20.      If MetaProduct. product_chain_id NotEqual -1 The
21.        Get ProductChain By Key：MetaProduct. product_chain_id
22.        Convert ProductChain. parent_product_id_array to ProductIdList<List>
23.        For ProductId in ProductIdList
24.          Add arrayList with Key，Value =“product_id”，Product. id+“/”+ProductId. product_id in-
to mapProductId
25.        End For
26.      End If
27.    End For
28.    Make ProductChain entity
29.    Set jsonArray into ProductChain. parent_product_id_array
30.    return ProductChain
31. End Function
```

随机存储快照数据生成时间复杂度为

$$T(n)=\begin{cases}1 & n=1,\\ T(n-1)+1+(n-1) & n>1,\end{cases} \tag{4}$$

将式(4)进行计算变换，即

$$T(n)=\lim_{n\to\infty}\frac{2^{n-1}-1+(n-1)}{2^n}=\frac{1}{2}+\lim_{n\to\infty}\frac{n-1}{2^n}=\frac{1}{2}, \tag{5}$$

可得该算法的时间复杂度为

$$T(n)=O(2^n)。 \tag{6}$$

因每个节点需存储前面所有节点的信息，故随机存储快照数据生成算法的空间复杂度为

$$S(n)=2(1+2+3+\cdots+n)=n^2+n=O(n^2)。 \tag{7}$$

3　两种快照检索算法与对比分析

3.1　链表快照数据检索

链表快照数据检索的算法逻辑是从根节点开始检索下层所有的节点，将其分解为检索每个节点和下层的原料商品，可归纳为一个递归过程，故选用递归算法实现。递归到当前节点的没有

子节点时,即满足终止条件,否则调用递归算法后并等待返回。伪代码如下:

```
1. Function searchProductParsentFromList( ProductChainId)
2.    Get ProdcutChain from product chain list with id:ProductChainId
3.    For MetaProduct in ProductChina. MetaProductList
4.       If MetaProdcut. product_chain_id Not Equal-1 Then
5.          searchProductParsentFromList ( MetaProdcut. product_chain_id )
6.       End If
7.       Get MetaProdcut. product_id
8.    End For
9. End Function
```

当溯源链在2个节点(1个普通节点和1个叶子节点时)条件下,时间复杂度为

$$T(n)=\begin{Bmatrix}1 & n=1,\\ T(n-1)+1 & n>1。\end{Bmatrix}=O(2^n)。\tag{8}$$

3.2 随机存储快照数据检索

因随机存储快照根节点的快照存储了全部溯源链的信息,故直接分解溯源链即等同于遍历全部目标节点,可实现所有节点的数据检索,伪代码如下:

```
1. Function searchProductParsentFromArray( ProductChainId)
2.    Get ProdcutChain from product chain list with id:ProductChainId
3.    For MetaProduct in ProductChina. MetaProductArray
4.       Get parentProduct with value:MetaProduct. product_id
5.    End For
6. End Function
```

每个节点(包含1个普通节点和1个叶子节点时)的时间复杂度为

$$T(n)=2(n-1)=O(n)。\tag{9}$$

4 实验结果及数据分析

因存储资源具备随时可扩充且成本低的特点,故用于存储快照冗余数据的存储器不会成为制约条件。在商品溯源业务场景更应关注查询溯源商品的时间,表8展示了链表快照模型和随机存储模型两种快照的生成和检索时间复杂度对比。

表8 2种快照生成和检索时间复杂度对比

Tab. 8 Comparison of time complexity of two snapshot generation and retrieval methods

项目	链表快照模型	随机存储模型
生成溯源商品时间	$O(2^n)$	$O(2^n)$
检索溯源商品时间	$O(2^n)$	$O(n)$

由表8可得,使用随机存储模型在检索溯源商品时效率更高,能大幅提升溯源商品的查询速度。为了验证这个结论,利用表9所示的测试数据模拟生成溯源商品上链和溯源商品检索。测试软硬件配置见表10。

表9　模拟用实验数据规模

Tab. 9　Experimental data scale

项目	数量/万	用途
顶层商品数	10	被溯源商品的数量
原料商品数	90	作为被溯源商品的原料商品、每层10万，共9层
溯源链表数	90	1）每个商品包含自身，其生成9条溯源记录 2）链表和随机存储的快照记录各由不同的字段保存

表10　实验软硬件配置

Tab. 10　Experimental software and hardware configuration

项目	规格
服务器	Apple M1 、10核CPU、32G内存
数据库	Ver 8.0.32 for macos13 on arm64（MySQL Community Server－GPL）
测试程序	Java、spring－boot 2.7.1、mybatis 2.2.2

4.1　快照数据生成与分析

模拟数据上链过程采用以下步骤：

1）对商品不断追索原料，生成溯源链中各层种子商品样本。

2）收集各层原料的产品规格、厂家数量等信息不少于100个。

3）随机选取商品名、规格、厂家等关键信息组合生成溯源链中各层商品，共达到100万条。

4）通过算法模拟生成溯源链，使溯源链记录达到90万条。

通过步骤1）～4）理论上模拟了区块链上链过程，解决了构建大量区块链节点的高昂硬件成本问题和需要不断挖矿的费时问题，再通过编码实现模拟上链、生成2种快照数据，生成时间对比见图5，生成空间对比见图6。其中，图5的X坐标为快照数据生成第几次，Y坐标为快照数据生成时间（单位：s）；图6的Y坐标为快照数据占用磁盘空间（单位：MB）。

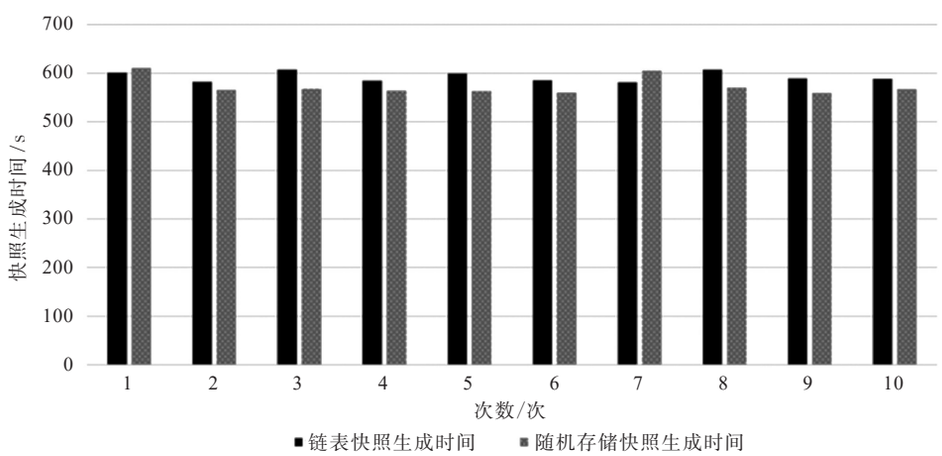

图5　2种快照数据生成时间对比图

Fig. 5　Comparison chart of the time taken to generate two types of snapshot data

根据图5可得，链表和随机存储快照用时接近，符合表8中时间复杂度相同的结论。

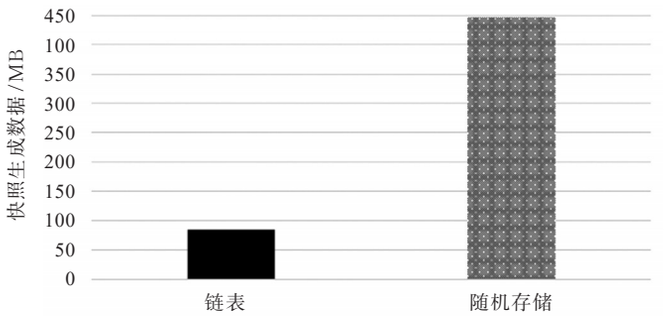

图6 2种快照数据生成空间对比图

Fig. 6 Comparison chart of the space taken to generate two types of snapshot data

根据图6可得,随机存储快照的空间占用是链表快照数据的5.3倍左右,符合随机存储快照的空间复杂度大于链表快照的空间复杂度的结论。

4.2 溯源商品检索与分析

分别使用模拟链上溯源数据检索(使用了混合索引算法[13])、基于链表快照溯源数据检索和基于随机存储快照检索并记录检索时间,重复10次实验后得出3种检索方法的运行时间对比图,见图7。其中,图7的Y坐标为每次数据检索时间(单位:s),X坐标表示3种数据检索算法进行了10次实验。

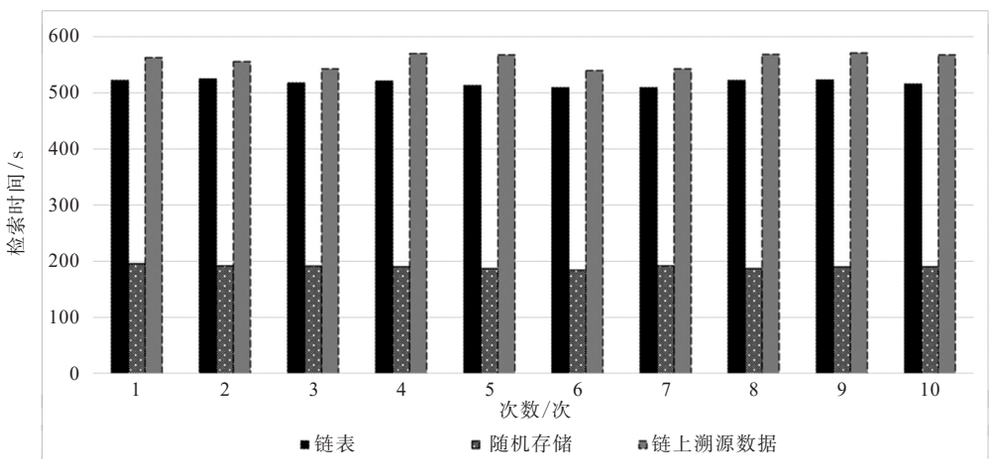

图7 数据检索时间对比图

Fig. 7 Comparison chart of the time taken to retrieve

根据图7可得,链表快照数据检索平均时间是随机存储快照检索平均时间的2.7倍,符合链表快照的时间复杂度大于随机存储快照的时间复杂度的结论,链上数据检索的时间最慢,长于链表快照数据检索平均时间。上述实验结果表明,随机存储的快照结构占用存储空间虽大,但对应的区块链数据检索算法时间最少。因存储硬件资源费用低廉,对于实际区块链的溯源系统,应选择检索时间更短的随机存储的快照结构,能最大限度提高消费者查询溯源信息的速度,提高系统的可用性,同时相对于混合索引更能提高区块链数据检索效率。

5 结语

随着区块链技术在食品溯源领域应用的日益增长,区块链提供的技术框架理论上解决了数据不可篡改的技术难题。但区块链作为一个分布式的存储系统,底层的技术框架决定了在处理集中式的高并发时更容易出现瓶颈和性能缺陷,同时受限于区块大小限制不能存储更多商品属

性值，不能实现随机查询功能等。结合到实际业务中，即如何解决让消费者能快速查询到商品的溯源信息。本研究在充分认识到区块链技术的基础上，结合传统数据库技术和利用溯源信息的生成即固定不变的特性，充分利用生成溯源信息的同时将已有的链式关系作为静态数据存储，使用冗余数据存储前后依赖关系，虽占用一定的存储空间，却能大幅度提高目标数据的查询速度。

通过模拟数据生成区块链的快照信息，充分对比了两种快照存储技术框架带来性能上的差异，论证了利用随机存储更有益于改善和提高查询相应速度，也从实验上证实了理论的正确性。由于模拟数据和真实数据仍然存在一定的差异性，希望将来能进一步优化区块链关键数据的快照存储技术，并应用于真实的业务场景。同时也希望能给从事溯源技术相关技术人员提供思路，改善区块链查询中存在的其他问题，提供最有利于用户的解决方案。

参考文献（References）

[1] 蔡文青，倪向东．乳制品生产信息追溯的过程与实现[J]．湖北农业科学，2011，50(15)：3184－3185，3190.

[2] 幸小刚．面向区块链的可搜索加密数据查询优化研究[D]．贵阳：贵州大学，2023.

[3] KAMATH R．Food traceability on blockchain：walmart's pork and mango pilots with IBM [J]．The Journal of British Blockchain Association，2018，1(1)：1－12.

[4] 刘雅东．基于区块链的溯源信息存储平台的研究与实现[D]．北京：北京邮电大学，2019.

[5] 傅威．溯源区块链共识机制研究[J]．信息系统工程，2019(5)：141－142.

[6] 陈飞，叶春明，陈涛．基于区块链的食品溯源系统设计[J]．计算机工程与应用，2021，57(2)：60－69.

[7] 刘怀愚，朱昌杰，李璟．时间复杂度的几种计算方法[J]．电脑知识与技术：学术交流，2011，7(7)：4636－4638.

[8] 卞立平，孙爱东，孙晓明，等．基于区块链技术的农产品深度溯源系统建设思考和设计方案[J]．江苏农业学报，2022，38(4)：1092－1098.

[9] 吴晓彤．基于区块链的农产品可信溯源系统研究与实现[D]．泰安：山东农业大学，2020.

[10] 王防修，周康．基于单链表的二叉树非递归遍历算法[J]．武汉工业学院学报，2012，31(4)：59－63.

[11] 李明佳，汪登，曾小珊，等．基于区块链的食品安全溯源体系设计[J]．食品科学，2019，40(3)：279－285.

[12] MORISHIMA S，MATSUTANI H．Accelerating blockchain search of full nodes using GPUs [C]．2018 26th Euromicro International Conference on Parallel，Distributed and Network－based Processing (PDP)．IEEE，2018：244－248.

[13] 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.

[14] 段冉阳，周文辉，魏骁，等．基于Hyperledger Fabric的食品溯源系统设计与实现[J]．电子技术应用，2021，47(3)：55－60.

[15] 刘佳琦，游新冬，吕学强，等．区块链技术在食品溯源行业的研究[J]．食品工业，2021，42(11)：273－277.

（责任编辑：胡燕梅，冯　舸）