Article

An Efficient Blockchain Transaction Retrieval System

Hangwei Feng [1,2], Jinlin Wang [1,2] and Yang Li [1,2,*]

1 National Network New Media Engineering Research Center, Institute of Acoustics, Chinese Academy of Sciences, No. 21, North Fourth Ring Road, Haidian District, Beijing 100190, China
2 School of Electronic, Electrical and Communication Engineering, University of Chinese Academy of Sciences, No. 19(A), Yuquan Road, Shijingshan District, Beijing 100049, China
* Correspondence: liyang@dsp.ac.cn

Abstract: In the era of the digital economy, blockchain has developed well in various fields, such as finance and digital copyright, due to its unique decentralization and traceability characteristics. However, blockchain gradually exposes the storage problem, and the current blockchain stores the block data in third-party storage systems to reduce the node storage pressure. The new blockchain storage method brings the blockchain transaction retrieval problem. The problem is that when unable to locate the block containing this transaction, the user must fetch the entire blockchain ledger data from the third-party storage system, resulting in huge communication overhead. For this problem, we exploit the semi-structured data in the blockchain and extract the universal blockchain transaction characteristics, such as account address and time. Then we establish a blockchain transaction retrieval system. Responding to the lacking efficient retrieval data structure, we propose a scalable secondary search data structure BB+ tree for account address and introduce the I2B+ tree for time. Finally, we analyze the proposed scheme's performance through experiments. The experiment results prove that our system is superior to the existing methods in single-feature retrieval, concurrent retrieval, and multi-feature hybrid retrieval. The retrieval time under single feature retrieval is reduced by 40.54%, and the retrieval time is decreased by 43.16% under the multi-feature hybrid retrieval. It has better stability in different block sizes and concurrent retrieval scales.

Keywords: blockchain retrieval; B+ tree; BB+ tree; I2B+ tree

check for updates

Citation: Feng, H.; Wang, J.; Li, Y. An Efficient Blockchain Transaction Retrieval System. *Future Internet* **2022**, *14*, 267. https://doi.org/10.3390/fi14090267

Academic Editor: Filipe Portela

Received: 22 August 2022
Accepted: 14 September 2022
Published: 15 September 2022

1. Introduction

Blockchain technology originated from the Bitcoin project [1] invented by Satoshi Nakamoto. This novel decentralized ledger technology has been used and explored in many fields, such as finance [2], digital copyright [3], supply chain management [4], and edge computing [5], and has achieved remarkable results.

With the rapid development of blockchain, the blockchain has also exposed some problems. Because the blockchain data have the append-only characteristic, under continuous accumulation, they cause enormous storage pressure to the blockchain nodes [6]. According to the data analysis provided by the blockchain explorer website, there are currently 670,000 blocks in Bitcoin, a total amount of data in the ledger close to 400 GB, involving 40 million account addresses. Ethereum has 12.24 million blocks. The total amount of data in the ledger has exceeded 1 TB, affecting 55 million account addresses [1,7]. The impact of a large blockchain ledger is as follows [6].

Each blockchain node needs to store a complete copy of the blockchain ledger data, and the amount of ledger data puts too much pressure on the node's storage. At the same time, thousands of blockchain nodes all need to store ledger data, which can lead to a huge waste of storage resources.

Too much pressure on a single point of storage can constrain new blockchain nodes from joining the blockchain network, leading to the centralization of the blockchain system, destroying the decentralized nature of the blockchain and directly affecting the scaling of the blockchain system.

In response to the storage problems arising from blockchain nodes, academics have proposed storing blockchain data in third-party decentralized storage networks, such as IPFS, to reduce the storage pressure [8–11]. The core idea of these blockchain storage solutions is to store the entire block data file to the IPFS network in order to reduce the storage strain on the blockchain nodes. The blockchain node is only responsible for storing the block hash and a summary of the transaction records. For acquisition of the block from the storage network, the blockchain node will utilize this hash and summary to verify the block data, ensuring that the blockchain is tamper proof.

However, blockchain is a ledger, made up of a combination of blocks, which contains a portion of the ledger's transaction records [12,13]. The transaction record is the smallest unit of the ledger, and the current blockchain transaction can support digital currency transfer, voucher record, logistics traceability and other kinds of information recording [14]. When the ledger data are stored in a third-party storage network, it raises the blockchain transaction retrieval problems.

The blockchain transaction retrieval problem is because blockchain nodes no longer have the complete blockchain ledger data when storing the blockchain ledger in third-party storage networks [15,16]. When blockchain ledgers are stored in third-party storage networks, blockchain nodes no longer have the entire blockchain ledger locally. Consequently, retrieving blockchain transactions becomes problematic. When users attempt to access the target transaction record, the blockchain node cannot locate the target block containing the target transaction. It requires blockchain nodes to regularly obtain the entire blockchain ledger from the storage network in order to access the transaction, resulting in considerable communication costs and poor user experience. Therefore, how to retrieve transaction in a large number of blocks based on the transaction records features becomes the most important aspect of the blockchain transaction retrieval challenge.

The traditional blockchain transaction retrieval approach [1,6] acquires the target transaction by traversing the blockchain ledger in local database by transaction hash, which is inefficient and cannot meet the requirements of transaction retrieval in many blocks in this scenario. Blockchain retrieval is a novel problem, and academia has also conducted extensive research on this. The current study mainly includes research on efficiency optimization of existing blockchain systems, query semantic retrieval function, and blockchain retrieval method research.

Research on efficiency optimization of the existing blockchain system and query semantic retrieval function includes the following: Li [17] proposes the etherQL system, a query layer designed on the outer layer of the blockchain. The main idea of the system is to copy the blockchain data to the external database MongoDB and create the query layer with the help of the functional interface provided by the external database to realize the blockchain data query. VQL [18] stores blockchain data based on the external database and makes the intermediate results verifiable. In response to the shortcomings of hyperledger fabric in processing temporal data, the processing efficiency is improved by adding copies and inserting redundant data [19]. By combining the Merkle tree and B+ tree characteristics, the blockchain is upgraded from the traditional hash-based query to the key-based query [20]. However, this type of research mainly considers the introduction of external databases or inserting redundant data to improve the retrieval efficiency of the blockchain. While increasing the extra cost of the system, it does not explicitly enhance the ability of blockchain retrieval. It only carries out optimization at the application layer and is not suitable for the needs of this scenario.

Blockchain retrieval method research includes Refs. [21,22]. Ren [21] proposes a DCOMB (dual combination bloom filter) method that combines the data stream of the Internet of Things with the timestamp of the blockchain to improve the versatility. Tu [22] proposes a set of solutions including extracting the account address, establishing an inverted index between the account address and the block hash, and using the B+ tree to manage it.

The program achieves good results. However, with the rapid expansion of the number of account addresses, retrieval efficiency is affected to a certain extent.

In this work, we conducted a study on blockchain transaction retrieval based on past studies. We focus on analyzing the characteristics of block data, extracting universal retrieval features, proposing an improved retrieval data structure, and effectively managing retrieval features to boost retrieval efficiency. The contributions include the following:

- For the blockchain transaction retrieval problem, based on the blockchain data semi-structured data features, this paper extracts the universal blockchain transaction characteristics, including account address and time, and establishes an efficient blockchain transaction retrieval system.
- In response to the lack of an efficient search structure, this paper proposes a scalable and flexible secondary search structure BB+ tree based on bloom filters and B+ trees, which can effectively manage account addresses. The I2B+ tree is introduced to establish time retrieval structure, which can efficiently manage the time characteristics and increase the attributes of blockchain retrieval.
- The experiment results prove that our system is superior to the existing methods in single feature retrieval, concurrent retrieval and multi-feature hybrid retrieval. The retrieval time under single feature retrieval is reduced by 40.54%, and the retrieval time is reduced by 43.16% under the multi-feature hybrid retrieval. It has better stability in the face of different block sizes and concurrent retrieval scales.

The content of this article is arranged as follows: Section 2 introduces the development of retrieval data structure, the bloom filter and I2B+ tree. Section 3 introduces the overall architecture of the blockchain transaction retrieval system, the establishment and update of the retrieval structure by the new data structure BB+ tree and I2B+ tree, the single-feature retrieval process, and the multi-feature hybrid retrieval process. Section 4 shows the performance evaluation results and analysis. In Section 5, the conclusions and future research are discussed.

2. Literature Review

2.1. Retrieval Structure

Although blockchain retrieval is a brand-new blockchain research direction, the academic community has conducted a lot of work on the research of the retrieval structure. We will briefly introduce the current research results in this section.

Retrieval structure, or index structure, academia has conducted a lot of research work, including the development of retrieval methods and retrieval data structure. The current retrieval method is mainly established by selecting a suitable data structure for the retrieval field, establishing a retrieval structure, quickly determining the offset of the retrieved field through the retrieval structure, and obtaining the complete data through the offset. Currently, the most widely used retrieval methods include signature files, suffix arrays, and inverted files [23–26].

- Signature file: Zheng and Faloutsos propose a retrieval technology for text information retrieval. They use a fixed-width signature and assign bit string to each item in the text information retrieval system. It requires ample space. So the cost of construction and update is high.
- Suffix tree and suffix array: Gonnet proposes the suffix tree, and Manber and Myers offer the suffix array. The suffix array uses less space than the suffix tree, but it takes a longer time to build. The main advantage of suffix array and suffix tree is that they support certain searches challenging better than inverted indexes in search phrases and regular expression searches.
- Inverted files: Inverted indexing is the most commonly used indexing method in modern search engines [27], whose core idea is to convert document–word information flow into word–document information. The inverted index table comprises two parts, namely the index item list and the event table of each index item itself. The index item contains the collection of all the words that have ever appeared in the document

collection. The event table of each index item itself includes a list of all documents in which a particular word has occurred or a pointer to actual data. When retrieving, according to the user's needs, the corresponding terms are retrieved in the index item list, and the information in the event table is obtained. For the realization of the inverted index, currently, the primary data structures used include hash linked list, B+ tree [26].

The central part of the hash linked list is the hash table. Each hash table entry stores a pointer, which points to the conflict-linked list. In the conflict-linked list, words with the same hash value form a linked list. The reason for the conflict-linked list is that two different words obtain the same hash value. If so, it is called a conflict in the hash method—the linked list stores word with the same hash value. The hash table is directly accessed according to the key value, which can quickly support random operations, such as adding, deleting, updating, checking. And the time complexity is O(1). However, the inverted index based on this method does not support sequential reading and scanning for the precise hash index. It is only suitable for a few specific occasions.

B tree is a very classical data structure. The B+ tree is an improvement of the B tree, which stores all data in leaf nodes and has indexes in non-leaf nodes. It has the advantages of fast full node scan, stable query speed, and natural sortability, which is widely used by databases [28]. It supports the deletion, modification, and lookup operation of a single record, and sequential scanning. To make B+ tree have better retrieval performance and apply for more scenarios, various experts and scholars have also proposed many improvement schemes for B+ tree. For example, Rao proposes CSSB+ tree [29]. The improved data structure makes more effective use of the CPU cache and speed up retrieval. Kim proposes a new data structure combining CSS and K-ary search [30], which can provide optimal search. Jin proposes a hybrid storage system index based on the B+ tree [31]. Some studies propose distributed indexes based on B+ trees, which are highly scalable and fault tolerant. Wang [32] proposes a tree structure MLB+ tree for fast multi-dimensional range retrieval, which is composed of several independent B+ trees. This structure can collect multi-dimensional features. In the scene of blockchain retrieval, the main retrieval features are account address and time characteristics. Tu [22] realizes a blockchain retrieval system based on the B+ tree. Compared with the traditional sequential retrieval and redis-based scheme, the retrieval performance is greatly improved. However, with the rapid increase in the number of account addresses, the direct use of B+ tree management will result in too many leaf nodes and a too large retrieval space, which will lead to a significant decline in retrieval efficiency. At the same time, because of the particularity of the time, it is not easy to manage the time feature in blockchain retrieval based on the B+ tree.

2.2. Bloom Filter

Bloom proposed the Bloom filter [33] in 1970. It is a probabilistic approximate retrieval structure that can be used for large-scale data. It can use a small amount of memory space to quickly and efficiently check whether an object is likely to exist in the set. With the explosion of data growth, Bloom filters have been widely used in many research fields, such as databases, distributed systems, and web caches [34], due to their high efficiency, low utilization of resources, and high flexibility between the bloom filter bits and the false positives probability. Specific optimization works include network packet filtering [35], p2p lookup [36], and metadata lookup [37], and many derivative types of Bloom filters, including counting Bloom filters [38], 2D Bloom filter, multidimensional Bloom filter [39], and so on.

The Bloom filter's core structure contains a bit array whose length is m with k hash functions. All bits in an empty Bloom filter are 0. When adding an element to the Bloom filter, each hash function maps the element to the bit array, and the bit array corresponding position is set to 1. When checking whether an element exists, the Bloom filter passes the element through k hash functions. If any related k positions is 0, the test element is not in the Bloom filter. As shown in Figure 1, item A sets positions 1, 5, and 6 to 1 through three different hash functions, hash1, hash2, and hash3. Item B sets positions 3, 5, and 6 to 1 through hash1, hash2, and hash3. When checking the new item C, the bit array is obtained through hash1, hash2, and hash3. When the positions set to 1 are consistent, it proves that the element is stored.

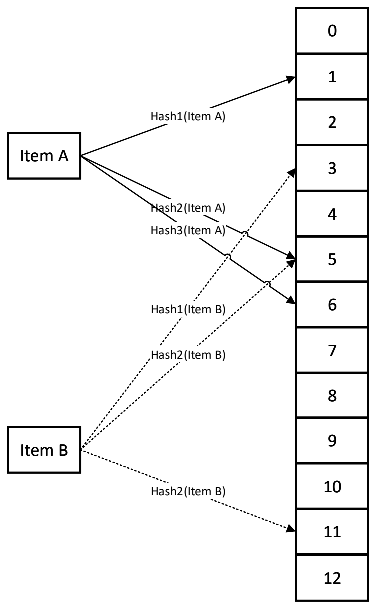

Figure 1. The Bloom filter.

The Bloom filter cannot produce false negatives but may produce false positives. False positives mean that the Bloom filter seems to store a item that does not actually exist in the Bloom filter. The k positions indicate whether an element is in the set as described above. Still, this method may cause the algorithm to mistakenly assume that an element that is not originally in the set is detected as being in the set. There is a mutually restrictive relationship between false positives and the Bloom filter bit array. We can see a connection between the two, whereby increasing the size of bit array in the Bloom filter reduces the false positives rate:

$$\mathrm{k} = \frac{m}{2^{\lceil \log_2 n \rceil}} * \ln 2 \tag{1}$$

$$\epsilon = \left(1 - \left(1 - \frac{1}{m}\right)^{kn}\right)^{k} \tag{2}$$

where ϵ is the false positives rate, k is the hash functions number, n is the resolved features number in the Bloom filter, and m is the Bloom filter bits number.

2.3. I2B+ Tree

I2B+ tree is an improved IB+ tree proposed by Edgar in 2020 [40]. It is an efficient index structure for time retrieval. It combines B+ tree and interval tree, which can effectively manage time interval information and retrieve time characteristics. It has good results and supports range retrieval. Specifically, the I2B+ tree is formed by the expansion of the B+ tree (n-ary tree). The I2B+ tree includes two types of nodes: internal nodes (its child nodes are leaf nodes) and leaf nodes (its father nodes are interval nodes). The internal node stores three tables: one table stores the addresses of its child nodes, the second stores the ordered node keys, and the last table stores the maximum value. Leaf nodes store ordered node key pairs and corresponding values. Similar to B+ tree, front and back pointers are set between leaf nodes and middle layer nodes to support range retrieval see Figure 2.

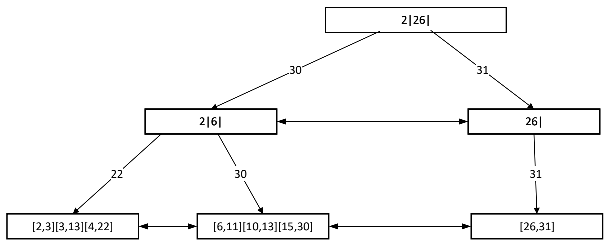

Figure 2. The I2B+ tree.

At the same time, to enhance the management of time, the I2B+ tree has two unique parameters: *order* and *alpha*. The *order* defines the child nodes maximum number that a node can have. The *alpha* represents an empirical factor. It affects selecting the separation point of the leaf node's child nodes. This parameter adjusts the balance of space and query time. The bigger *alpha* value results in a higher split point value. Therefore, fewer time splits occur, resulting in less storage and lower query efficiency. Compared with IB+, the biggest difference of the I2B+ tree is that the I2B+ tree increases the front and back pointers between the non-leaf nodes, which can reduce the spatial fragmentation rate during tree splitting and enhance the retrieval ability.

3. System Implementation

Blockchain retrieval is a complex retrieval scenario, which requires the retrieval system to have the characteristics of dynamic updating, fast and stable retrieval. In addition to a data structure with excellent performance, achieving good retrieval results needs a set of matching systems. This section introduces the overall architecture of the blockchain transaction retrieval system, retrieval feature extraction processing, retrieval structure generation, and retrieval process.

3.1. Blockchain Transaction Retrieval System Architecture

In this section, we introduce the blockchain retrieval system architecture. As shown in Figure 3, it mainly includes four modules: collection module, retrieval module, blockchain storage module and retrieval structure maintenance module.

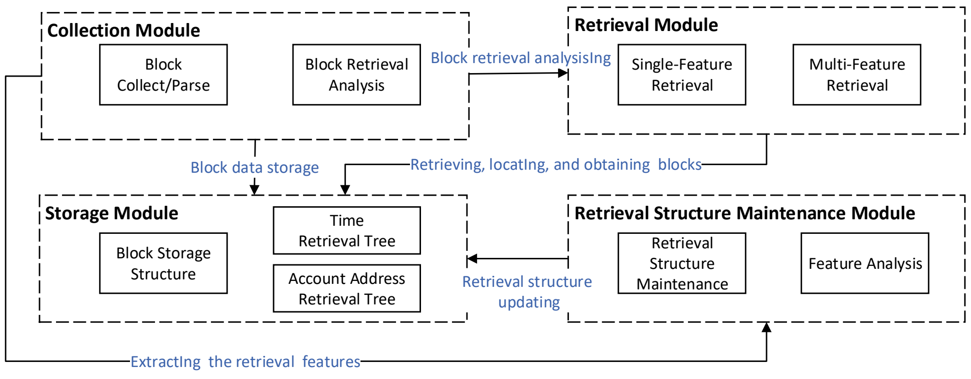
Collection Module
Block Collect/Parse
Block Retrieval Analysis
Block retrieval analysisIng
Retrieval Module
Single-Feature Retrieval
Multi-Feature Retrieval
Block data storage
Retrieving, locatIng, and obtaining blocks
Storage Module
Block Storage Structure
Time Retrieval Tree
Account Address Retrieval Tree
Retrieval structure updating
Retrieval Structure Maintenance Module
Retrieval Structure Maintenance
Feature Analysis
ExtractIng the retrieval features

Figure 3. Blockchain retrieval system architecture.

Collection Module : The primary responsibility is block collection/parsing and blocks retrieval analysis. The collection module obtains the block after the upper consensus module of the blockchain node, verifying that the block and sends the complete block data to the storage module for storage. At the same time, the block content is analyzed to extract the characteristic information needed for retrieval. The collection module sends it to the maintenance module of the retrieval structure to establish or update the retrieval structure. Block retrieval analysis extracts the retrieval targets and sends the retrieval features, including the account address and time stamp, to the retrieval module.

Storage Module: The primary responsibility is to store/obtain the block, and store/update the retrieval structure. The block data received from the collection module are stored in a file, which contains a mapping table between block hash and block offset. Block files can be stored locally or on IPFS. After the retrieval module retrieves the block's hash, which contains the transaction record, the system can obtain the block through the storage module. At the same time, the time retrieval structure and account address retrieval structure are stored for retrieval by the retrieval module.

Retrieval Structure Maintenance Module: The primary responsibility is to update and maintain various retrieval structures: update and keep the acquired retrieval feature information, including account address and timestamp, for the corresponding retrieval structure.

Retrieval Module: The primary responsibility is to perform single-feature retrieval or multi-feature hybrid retrieval based on the retrieval feature information to determine the block that contains the transaction record.

3.2. Retrieval Feature Extraction Processing

First of all, the blockchain retrieval system needs to extract the retrieval features from the block. Blockchain links block together through hash pointers. The data in the blocks are semi-structured, and each block consists of a block header and block body. The block header comprises consensus-related information, including version number, parent block hash, timestamp, Merkle root, block size, and other information.

The first step is to extract the characteristics, including the transaction timestamp, sender account address, and receiver account address, and generate the block summary file shown in Table 1. The block hash is the hash name for the entire block, followed by the transaction hash for each transaction, the account address of the sender participating in the transaction, the account address of the recipient participating in the transaction, and the time when the transaction was created.

The second step is to process these retrieval characteristics. The sender and receiver account addresses are processed separately for the accuracy of retrieval. The key–value pairs of the sender's transaction address and block hash and the key–value pairs of the receiver's transaction address and block hash are generated, respectively. These are sent to the retrieval structure maintenance module to update the account address retrieval structure. Then, the transaction timestamp in the block and block hash are sent to the time retrieval structure for updating.

Table 1. Block summary document.

Block Hash			
transaction1 hash	sender addresses	receiver addresses	transaction1 creation time
transaction2 hash	sender addresses	receiver addresses	transaction2 creation time
transaction3 hash	sender addresses	receiver addresses	transaction3 creation time
transaction4 hash	sender addresses	receiver addresses	transaction4 creation time

3.3. Retrieval Structure Generation and Analysis

In reviewing previous studies, we have seen a lot of research on optimizing the retrieval structure, including different retrieval methods and improvements to the B+ tree. However, these research schemes are not suitable for blockchain retrieval scenarios for the following reasons: First, the B+ tree structure lacks scalability with the rapid increase in the number of account addresses. Second, due to the random distribution of block time, it is difficult for the traditional B+ tree to establish an efficient retrieval structure for the time.

In this section, we propose a novel data structure BB+ tree. This efficient data structure combines the advantages of the B+ tree and Bloom filter. It also has scalability and fast retrieval speed. Next, we will sequentially introduce a time and space complexity analysis of the BB+ tree structure, account address retrieval tree based on the BB+ tree, and time-retrieval tree based on the I2B+ tree.

3.3.1. BB+ Tree

The B+ tree is a very classic retrieval structure with the advantages of efficient retrieval speed and supporting range retrieval. In the blockchain retrieval scenario, account addresses and block hashes are treated as key–value pairs and managed through the B+ tree. During blockchain transaction retrieval, the account address is retrieved as the key value in the B+ tree to obtain the block hash and the corresponding block for the complete transaction. However, with the rapid growth of the account address, the B+ tree is directly used to manage the account address retrieval items, resulting in the too-large tree structure and too-large retrieval space, directly affecting the retrieval time.

Given the deficiency of the B+ tree, we propose a BB+ tree. The BB+ tree has two core elements: Bloom filter and B+ tree. Then, we will introduce the BB+ tree in detail. We use the Bloom filter to optimize the B+ tree, mainly because the Bloom filter uses less space and has fast calculation speed. It can have better scalability and retrieval efficiency when facing a large number of account address retrieval items.

As shown in Figure 4, the BB+ tree is composed of a two-level structure, where the first level is the Bloom filter, and the second level is the B+ tree. The first level structure is used to quickly classify key values and reduce the search space: when a new pair of key–value enters, the key value is first input into the Bloom filter for calculation. The Bloom filter will generate an m-bit array. According to the bit array, the key–value pairs are managed by the subB+ tree corresponding to the input bit array. The second level structure is that after receiving the key–value pair of the first level structure, the BB+ tree will construct the subB+ tree according to the key value of the key–value pair to facilitate the key–value retrieval item. Similar to the B+ tree, the front and back pointers in the BB+ tree are set between leaf nodes to support range retrieval.

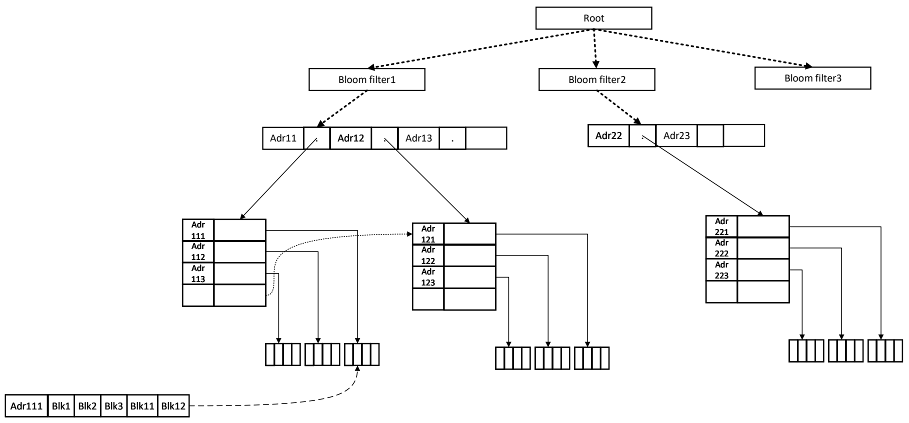

Figure 4. The BB+ tree.

The following is the pseudo-code analysis of the BB+ tree when inserting key–value pairs in Algorithm 1 and retrieving the key value in Algorithm 2:

3.3.2. Performance Optimization Analysis of BB+ Tree

This section will analyze the time and space complexity of the B+ tree and BB+ tree.

The data structure features of the B+ tree are as follows: internal nodes store corporate index relations, while leaf nodes store actual data. The retrieval of a key value often depends on the height of the tree, so the time complexity is

$$\text{Read}_{B+\text{ Tree}} = O(\log_d n) + s \tag{3}$$

where d is the order of the B+ tree, the total number of key–value pairs (number of leaf nodes) is n, and s is the additional query time cost. Although the logarithmic time complexity is excellent, with the rapid expansion of the search space, even logarithmic complexity cannot guarantee the search speed. Faced with the problem of retrieval space expansion, we propose a new structure, BB+ tree, whose time complexity consists of two parts: the time complexity of the Bloom filter and the time complexity of the subB+ tree, as follows:

$$\text{Read}_{BB+\text{ Tree}} = O(k) + O(\log_d n/2^m) + s \tag{4}$$

where k is the time required to calculate k hash functions, and m is the number of bit array. According to formula k in the Bloom filter, there is a relationship between m and n:

$$\text{k} = \frac{m}{2^{\lceil \log_2 n \rceil}} * \ln 2 \tag{5}$$

Substituting the expression of the above equation k into the formula, we can obtain

$$\text{Read}_{BB+\text{ Tree}} = \frac{m}{2^{\lceil \log_2 n \rceil}} * \ln 2 + \log_d \frac{n}{2^m} + s \approx \frac{m}{n} * \ln 2 + \log_d \frac{n}{2^m} + s \tag{6}$$

Taking n as the variable, we take the derivative of the time function of BB+ and compare it with the time function of B+. It can be found that the magnitude of the change of the BB+ tree is smaller than the derivative result of the B+ tree with the increase in n, which indicates that the growth rate of the BB+ retrieval time is slower than B+ tree's time with the increase in retrieval space.

$$\text{Read}'_{B+\text{ Tree}} = \frac{1}{n \ln d} \tag{7}$$

$$\text{Read}'_{BB+\text{ Tree}} = -\frac{m}{n^2} \ln 2 + \frac{1}{n \ln d} \tag{8}$$

The space consumption of the B+ tree depends on the number of nodes. In actual operation, the node's size matches the size of the disk page (usually 4, 8, or 16 K), so the size of the B+ tree is multiplied by the B+ tree nodes number and the disk page size. In addition to the space consumption of the B+ tree, the space consumption of the BB+ tree structure also includes the space consumption related to the Bloom filter.

Algorithm 1 Insert [k,v] in the BB+ tree

Input: [k,v] a pair of K-V value, a matrix $BF_{[i][j]}$ for bloom filters, $i \in \{0, 1, 2, \ldots, I-1\}$ and $j \in \{0, 1, 2, \ldots, m-1\}$, where m is the bloom filter bits size, H is hash functions number, three seed hash functions $h_1(x)$, $h_2(x)$ and $h_3(x)$

Output: true or false;

```
1:  $h_1 \leftarrow h_1(k), h_2 \leftarrow h_2(k), h_3 \leftarrow h_3(k)$
2:  $H_x = 0$;
3:  for $x = 0$ to $H-1$ do
4:      $H_x(k) = (h_1 + x * h_2 + x^2 * h_3) mod\, m$
5:      $H_x = H_x \,||\, H_x(k)$
6:  end for
7:  for $i = 0$ to $I-1$ do
8:      if $\exists i, H_x = BF_{[i][]}$ then
9:          access $subB + tree_i$ in BB+ tree
10:         if reach a leaf node k in $subB + tree_i$ then
11:             update leaf node k with [k,v]
12:         else insert a new leaf node k in $subB + tree_i$
13:         end if
14:     else
15:         insert $H_x$ in matrix $BF_{[i][]}$
16:     end if
17: end for
```

Algorithm 2 Query [k,v] in the BB+ tree

```
Input: $k$, the target key, a matrix $BF_{[i][j]}$ for bloom filters, $i \in \{0,1,2,\ldots,I-1\}$ and $j \in \{0,1,2,\ldots,m-1\}$, where $m$ is the Bloom filter bits size, $H$ is hash functions number, three seed hash functions $h_1(x)$, $h_2(x)$ and $h_3(x)$
Output: a set of value in [k,v]
1:  $h_1 \leftarrow h_1(k), h_2 \leftarrow h_2(k), h_3 \leftarrow h_3(k)$
2:  $H_x(k) = 0$;
3:  for $x = 0$ to $H-1$ do
4:      $H_x(k) = (h_1 + x * h_2 + x^2 * h_3) mod\, m$
5:      $H_x = H_x \mid\mid H_x(k)$
6:  end for
7:  for $i = 0$ to $I-1$ do
8:      if $\exists i, H_x = BF_{[i][]}$ then
9:          access $subB + tree_i$ in BB+ tree
10:         if reach a leaf node k in $subB + tree_i$ then
11:             return leaf node k
12:         else
13:             return k does not exist in $subB + tree_i$
14:         end if
15:     end if
16: end for
```

$$\text{Size}_{B+\text{ tree}} = \text{Size}_{\text{Page}} * \left(\sum_{\text{level}=0}^{h} \text{Page}_{\text{ level}} + \text{Page}_{\text{ leaves}} \right) \tag{9}$$

$$\text{Size}_{BB+\text{ tree}} = \text{Size}_{\text{ Page}} * \left(\sum_{\text{level}=0}^{h} \text{Page}_{\text{ level}} + \text{Page}_{\text{ leaves}} \right) + n * 1.44 \log_2(1/\epsilon) \tag{10}$$

$$\epsilon = \left(1 - \left(1 - \frac{1}{m}\right)^{kn}\right)^{k} \tag{11}$$

where $Page_{level}$ represents this level nodes, and $Page_{leaves}$ represents the number of leaf nodes. From the analysis of the above equation, we find that BB+ trees, compared with B+ trees, provide better retrieval performance in the face of many key values, at the cost of providing additional storage space to maintain the Bloom filters. However, the Bloom filter has the advantages of not needing to save original data, occupying less data, etc. One element in the B+ tree may occupy 4 KB, while one element in the Bloom filter may occupy 4 bits. So the space consumption of the Bloom filter can be ignored when comparing with the space-occupying capacity of the B+ tree. According to the theoretical analysis of time consumption and space consumption, the BB+ tree structure has better performance than the B+ tree structure.

3.3.3. Account Address Retrieval Tree Based on BB+ Tree

In [22], it uses the account address and block hash to establish an inverted index and introduces a B+ tree to manage an inverted index. However, with the significant increase in account addresses (the current scale of account addresses can reach millions), facing the search space rapid expansion, the B+ tree lacks flexibility and expansibility, leading to the decline in retrieval efficiency. Therefore, this paper proposes a new retrieval structure BB+ tree. In this section, the blockchain retrieval builds an account address retrieval tree based on the BB+ tree.

Establish the account address retrieval structure: As shown in Figure 5, when the retrieved items from the collection module, such as adr111 (corresponding to block1 hash, block2 hash, and block3 hash), adr112 (corresponding to block4 hash, block5 hash, and block6 hash) and adr113 (corresponding to block7 hash and block8 hash), appear, the blockchain retrieval system enters these account addresses into the Bloom filter. For example, when adr111 appears, the Bloom filter will computer and classify adr111. If the result after computing in Bloom filter is 0111, it will enter the subB+ tree corresponding to the Bloom filter 0111 and establish the retrieval item about this account address in subB+ tree. If the result after computing in the Bloom filter is 0011, there is no subB+ tree corresponding to the result after computing in the Bloom filter, and the retrieval system will update the Bloom filter and establish new subB+ tree. The follow-up work is described in the previous step.

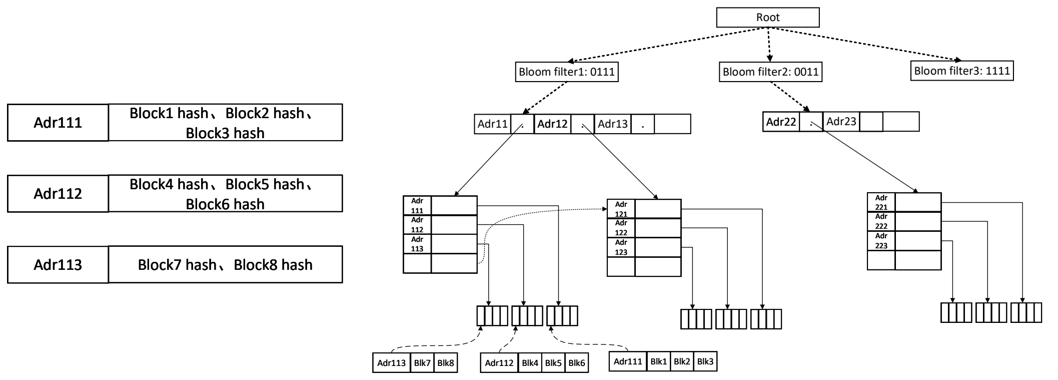

Figure 5. The account address retrieval structure based on BB+ tree.

Update the account address retrieval structure: When an update of the original account address is received, the retrieval items related to an account address in the account address retrieval structure are updated as shown in the following Figure 6. When the account address of the account address adr111 is updated, the block containing the account address is expanded from three to five. Read the retrieval item of adr111 from the retrieval structure, and add and update the newly added block hash block11 hash and block12 hash to the retrieval item.

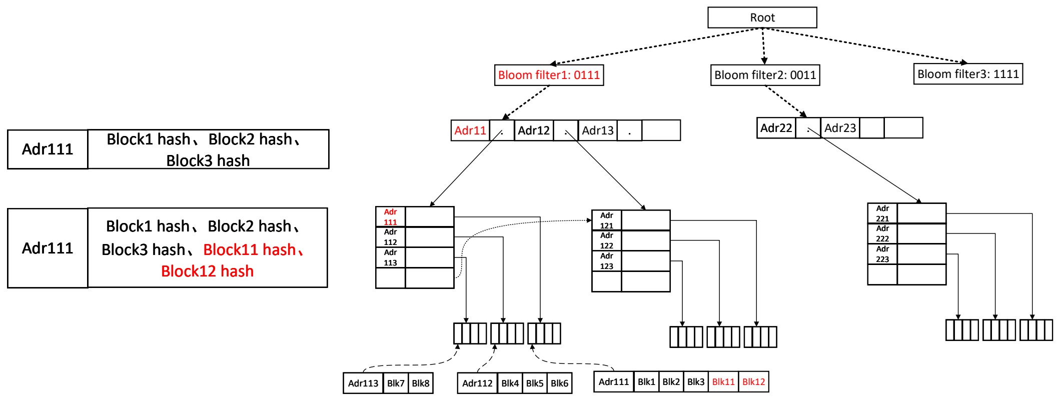

Figure 6. Updating the account address retrieval structure.

Retrieve in the account address retrieval structure: When the account address retrieval is performed, the retrieval module analyzes the target account account address. The system first uses the Bloom filter to distinguish and classify the account address, and enters the corresponding subB+ tree. Secondly, the retrieval item of the connected account address is obtained under the subB+ tree, and the block hash containing the account address is obtained.

3.3.4. Time Retrieval Structure Based on I2B+ Tree

Due to the complexity of time management, it is difficult for the B+ tree to manage and merge the time feature effectively. However, the blockchain retrieval scenarios need the B+ tree's range retrieval. This paper introduces the I2B+ tree—an improved B+ tree for the time index. The I2B+ tree combines the characteristics of the B+ tree and interval tree, which can store the time period, and support range retrieval. Based on the I2B+ tree, key–value pairs which contain block time and block hash are obtained from feature extraction and processing. The core idea is that the timestamp of a block which represents the generated time can be transformed into a time period (from the creation time of a block to the creation time of the next block). The I2B+ tree can merge the time period, and improve the management and retrieval efficiency of the time retrieval structure. According to the block information parsed by the collection module, the block hash and the corresponding time period can be obtained through analysis. The time period is input into the I2B+ tree as the key value, and the block hash as the key value.

Establish time retrieval structure: As is shown in Figure 7, convert the time the time period into value from 0 to 10,000, and enter the I2B+ tree. For example, the corresponds period to block1 hash is "2–3". The corresponding periods to block2 hash, block3 hash, block4 hash, block5 hash and block6 hash are "4–10", "11–20", "21–23", "24–30" and "31–35". The retrieval system will enter these period to the I2B+tree.

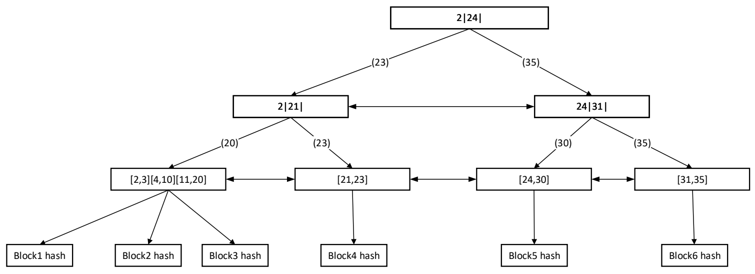

Figure 7. Time retrieval structure based on I2B+ tree.

Update time retrieval structure: As shown in Figure 8, when the block7 hash and block8 hashes appear, the corresponding periods are "36–40" and "41–48". Hash7 and Hash8 are updated to the time retrieval structure. The internal node 24|31 in the I2B + tree is updated to 24|36. At the same time, merge (31,35) with (24,30), add two time periods of (36,40) (41,48), and also add block hashes of block7 and block8.

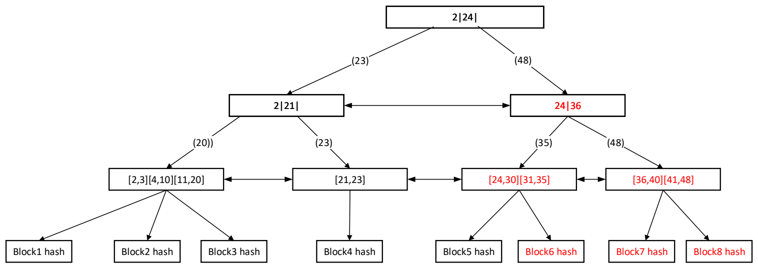

Figure 8. Updating the time retrieval structure.

Retrieve in time retrieval structure: When the retrieval target time is "9–12", the I2B+ tree traverses its nodes and selects the node involving the target period and the block hash values of the corresponding leaf nodes, block2 hash and block3 hash, to return.

3.4. Blockchain Retrieval Processing

Blockchain transaction retrieval is a complex scenario, and users will have some feature information when retrieving. The blockchain retrieval system proposed in this article can select single-feature retrieval or multiple-feature retrieval based on feature information.

3.4.1. Single-Feature Retrieval

The retrieval module carries out retrieval based on one characteristic of time, sender account address, and receiver account address, and obtains the complete block data from the storage module.

Based on the time characteristics, the retrieval module extracts the time of the retrieved target. The retrieved the time by using the time retrieval tree, and the set of block hashes in the time is output. Complete block data are obtained through the block storage module and analyzed for verification.

Based on the account address characteristics, the retrieval module extracts the receiver's address or sender's address of the retrieved target. The account address retrieval tree is used to retrieve the account address, and the set of block hashes related to the account address is output. Complete block data are obtained through the block storage module and analyzed for verification.

3.4.2. Multi-Feature Retrieval

As shown in Figure 9, the process is similar to single-feature retrieval. The retrieval module will analyze various features contained in the retrieval target. The retrieval based on time and account address features is carried out simultaneously in parallel to obtain the block hash set. The retrieval module takes the intersection operation based on three kinds of block hash sets obtained by different retrieval structures. According to the hash of the block, the system obtains the hash of the target block and receives the corresponding block in the storage module to obtain the related complete transaction records.

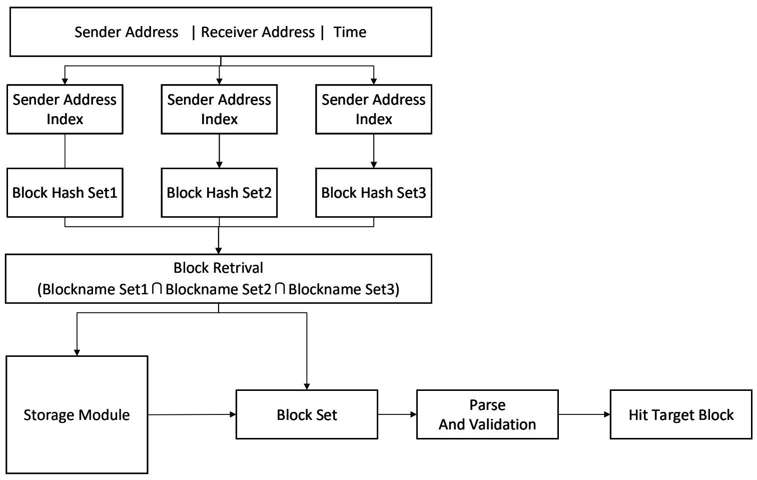

Figure 9. Multi-feature retrieval process.

4. Performance Evaluation

In this section, we evaluate the performance of the proposed blockchain retrieval system. The experimental platform used in this experiment is a server with Intel Xeon(R) CPU E5-26090 @ 2.40 GHz, 16 GB memory and 2TB HDD, and the operating system is Centos 7.9. Part of the blockchain data used in the experiment comes from xblock [41]. This website provides blockchain data for research and exploration, including actual data from currently influential blockchain projects, such as Bitcoin and Ethereum. This research uses a data set of actual transactions on the Bitcoin network from 2014 to 2016. Bitcoin is the model of most current blockchain projects, with distinctive blockchain characteristics, and its data are more representative. A portion of the data is randomly generated on demand, whose data structure is consistent with that of Bitcoin data. As shown in Table 2, Bitcoin block data for experiments include some features, including the following: **BlockID**, which represents the height of the block; **BlockHash**, which represents the hash of the block; **Btime**, which represents the time when block was created; and **Txs**, which represents the number of transactions in the block; In Bitcoin, each block contains 200 transactions on average. In Table 3, the presented results are transaction information in block 327872. **Transaction** represents the hash of the transaction. **Sender Address** represents the account address of sender in the transaction. **Receiver Address** represents the account address of the receiver. **Time** represents the time when the transaction was created.

Table 2. Block data.

BlockID	BlockHash	Btime	Txs
327872	00000000000000001BECC903BF44978DEDE7C0429B0	1414771239	288
327873	000000000000000008F1C9DC87C18D50D6AAC82923	1414771715	256
327874	00000000000000001A80CF302D79ED1322AFB14Fsq1	1414771623	176

Table 3. Transaction data in block 327872.

Transaction Hash	Sender Address	Receiver Address	Time
18755B2319DA1E1A 7F4432AF8C505D881	1111111111111111 1111BZbvjrl12xc	11114bV2T9cgCnj 8EffSHZtvop66atyv	1414771239
9ADE39C5D8CB3ED1 B91D702FF57A0FD9w	11111p3u5wS3s2DM P3LMRhoqcCmVFte	1114UGwDgKvF5dT eBzHBN6iN5BS5ao1JB	1414771239
12AD8088D7ECA878 CC7FBAAFAB9B2F5B2	11117APqhd39TNQL pVQhSLBXo3vA8KCb	111dTWEm8zcGJNT M5XBPLKjf3yRqxLWAv	1414771239

For comparative analysis, we implement BRBR [22], which is a novel and representative B+ tree transaction retrieval data structure based on extracting the sender transaction addresses for transaction retrieval. At the same time, this paper proposes the BB+ tree, in which we can dynamically adjust the bits array number in the Bloom filter. This experiment compares N = 2, N = 4, and N = 6 to observe the influence of the Bloom filter bits, where N is the bits number.

The experiment mainly has two parts. The first experiment part analyzes the time and space consumption of the BB+ tree proposed in this paper. The content of the investigation is the construction time and the amount of space occupied by the retrieval structure based on the BB+ tree under different block sizes. The second part analyzes the retrieval efficiency of the BB+ tree single feature in different scenarios and the retrieval efficiency of mixed multi-feature retrieval.

4.1. Time and Space Consumption for BB+ Tree

The experiment mainly evaluates the time and space consumption of the BB+ tree in different block sizes. Firstly, the blockchain retrieval system extracts the account address and block hash in the Bitcoin data provided by xblock and inserts the key–value pairs to BB+ tree and BRBR. Then, we analyze the comparison results of the establishment time and the retrieval file size when the block sizes are 1000, 3000, and 5000.

As can be seen from Figure 10a, we obtain the comparison results including BRBR, BB+ tree (N = 2), BB+ tree (N = 4), and BB+ tree (N = 6) to establish the retrieval structure in different block sizes. When the block sizes are 1000, 3000, and 5000, the time consumption of BRBR is 11.82, 40.26, and 71.52 s, respectively. The time consumption of the BB+ tree (N = 2) is 11.55, 37.02, and 60.74 s. We can see that the time consumption results of BB+ are better than BRBR in different situations. This is because the BB+ tree reduces the retrieval space of each subB+ tree by splitting the retrieval space so that it can be faster when inserting, updating, and deleting the retrieval items. When the Bloom filter's bits number N is different, the retrieval structure establishment's time is also different. When the block size is small, the performance of the BB+ tree is not outstanding after introducing the Bloom filter. However, as the block size increases, its advantages gradually manifest. As can be seen from Figure 10a, the larger the N in the BB+ tree, the shorter the establishment time of the retrieval structure. The reason is that the number of bits N of the Bloom filter increases so that the search items can be maintained in more subB+ trees. The subB+ tree has fewer items, which makes inserting, updating, and other maintenance efficient.

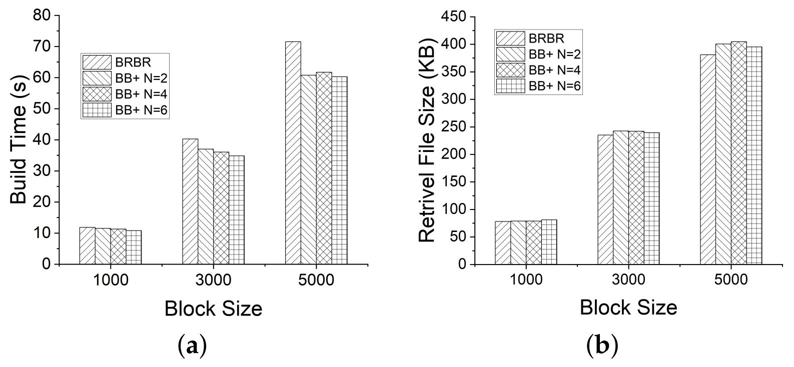

Figure 10. A comparison of time and space consumption. (**a**) Retrieval file build time between B+ and BB+; (**b**) retrieval file size between B+ and BB+.

As can be seen from Figure 10b, we obtain the space consumption comparison results of BRBR, BB+ tree (N = 2), BB+ tree (N = 4), and BB+ tree (N = 6) under the block sizes of 1000, 3000, and 5000, and the space consumption of BRBR is 78.15, 235.01 and 380.88 KB. The space consumption of the BB+ tree (N = 2) is 78.82, 242.59, and 404.76 KB. It can be seen that in terms of space consumption, BRBR is slightly better than BB+. The reason is that the BB+ tree introduces the Bloom filter, which will cause extra bitmap and other space occupation compared to BRBR. However, the Bloom filter has the advantages of occupying less space and having good performance, so the space consumption of the BB+ tree is not increased significantly compared with that of BRBR.

4.2. Block Retrieval Performance Test and Analysis

The performance evaluation of retrieval structure mainly includes three experiments: Experiment 1 investigates the retrieval performance of the account address retrieval structure based on the BB+ tree proposed in this paper under different block sizes. In experiment 2, blockchain retrieval often faces concurrent scenarios. This experiment investigates the retrieval performance of the BB+ tree under different concurrent retrieval scales. In experiment 3, we add time features to examine the performance of the proposed multi-feature retrieval.

4.2.1. Single-Feature Retrieval Based on the BB+ Tree

This experiment mainly focuses on the performance of the retrieval structure in different block sizes. Firstly, the blockchain retrieval system extracts the account address and block hash in the Bitcoin data provided by xblock and inserts the key–value pairs to BB+ tree and BRBR. Then, we conduct the blockchain single-feature retrieval experiments. The retrieval target is shown in Table 4. The retrieval structure contains the target, and the number of target key–value pairs is fixed. Finally, we analyze the comparison results of retrieval time in different block sizes, and the block sizes are 500, 1000, 1500, 2000, 2500, 3000, 3500, 4000, 4500, and 5000.

Table 4. Single-feature retrieval example.

Type	Description	Example
Single-Feature retrieval	Account address retrieval	address = "111113cvaEeMfzNNASCkn49qddBTdX1y4"

As shown in Figure 11, the abscissa is the block size, and the ordinate is the retrieval time. It can be observed from the figure that as the block size increases, the increase in the account address retrieval space will cause the retrieval time of BRBR and BB+ to increase, but the retrieval time of BB+ is better than the BRBR in the overall retrieval time. Especially in the small block size of 500 to 1000, the retrieval time of the BRBR can see a very significant increase, and then the increasing trend becomes flat. When BB+ is in the block size of 500 to 5000, the retrieval time increases with the increase in the block number. As a whole, compared with BRBR, the retrieval time increased more gently with the increase in the block size without a spike, indicating that the BB+ retrieval structure has good scalability in the face of the rapid increase in the block size. Secondly, when the bits number N in the Bloom filter bit array is different, the BB+ performance is also different. Compared with BB+ (N = 2), the overall performance of BB+ (N = 4) is better, the retrieval time is shorter, and the growth trend is more gradual. It indicates that the Bloom filter enables the BB+ tree to effectively reduce the size of retrieval space and improve the retrieval efficiency. However, compared with BB+ (N = 4) and BB+ (N = 6), even in 1500–2000, the retrieval time of the BB+ (N = 6) fluctuated. This means that as the Bloom filter bit array increases, additional computational complexity may be introduced, thus affecting the overall retrieval time. In 4000–5000, BB+ (N = 6) is gradually better than BB+ (N = 4), indicating that with the increase in block size, the retrieval time based on the BB+ retrieval structure begins to change again. It shows that the increase in N will not necessarily improve the retrieval efficiency under the block size, and the size of N should be dynamically adjusted according to the block size and the retrieval requirements of the transaction.

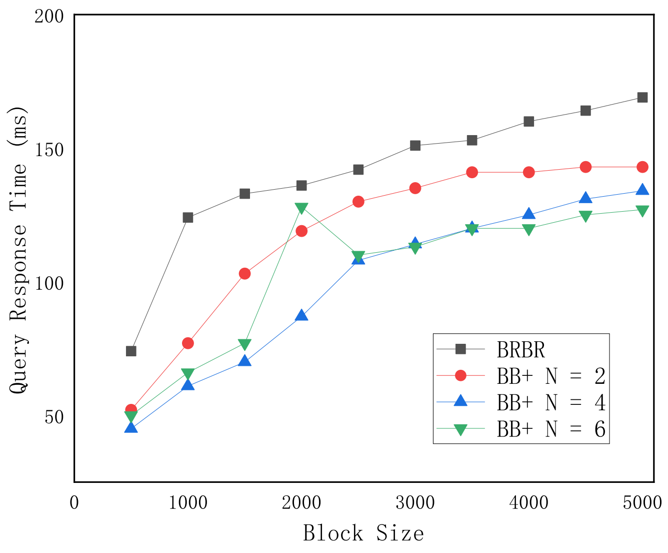

Figure 11. Query response time in different block sizes.

4.2.2. Concurrent Retrieval Performance

This experiment mainly focuses on the performance of the retrieval structure in different concurrent retrieval scales. Firstly, the blockchain retrieval system extracts the account address and block hash in the Bitcoin data provided by xblock and inserts the key–value pairs to BB+ tree and BRBR. Then we conduct the blockchain concurrent retrieval experiments. The retrieval target is shown in Table 5. The retrieval structure contains the target, and the number of target key–value pairs is fixed. We select the block sizes of 500, 1500, 3000, and 5000, and the number of concurrent retrievals is 2, 4, 6, 8, 10, and 20. Finally, we analyze the comparison results of the retrieval time increase.

Table 5. Concurrent retrieval example.

Type	Description	Example
Concurrent retrieval	Account address retrieval	address = "111113cvaEeMfzNNASCkn49qddBTdX1y4"

We introduce a comparison parameter B_i [22], where i is the number of retrieval participants in concurrent retrieval and t_2 is the retrieval time when the number of retrieval participants is 2. This parameter is used to compare the performance changes of different retrieval structures when the size of the retrieval requests increases:

$$B_i = \frac{t_i}{t_2}(\mathrm{i} = 4, 6, 8, 10, 20) \tag{12}$$

As is shown in Figure 12, the abscissa is the retrieval requests number, and the ordinate is the comparison parameter B_i. When the number of blocks is 500, the retrieval time of BRBR is longer than BB+. As the number of concurrent retrievals increases, the increase in B_i is relatively tiny. It shows that BB+ tree has a shorter retrieval time in a small-scale block retrieval scenario, and has weaker stability than that of BRBR in the face of the increase in the number of concurrent retrievals. As the block size increases, especially in blocks of 1500 and 3000, the BB+ tree has good performance. Compared with BRBR, the retrieval time is less, and the increase in the BB+ B_i is also more minor. With the rise in the number of concurrent searches, the retrieval performance of the BB+ tree-based retrieval structure is excellent and stable. When the number of blocks is 5000, we can see that the two are close to each other in terms of stability, but the overall retrieval time based on BB+ tree is better.

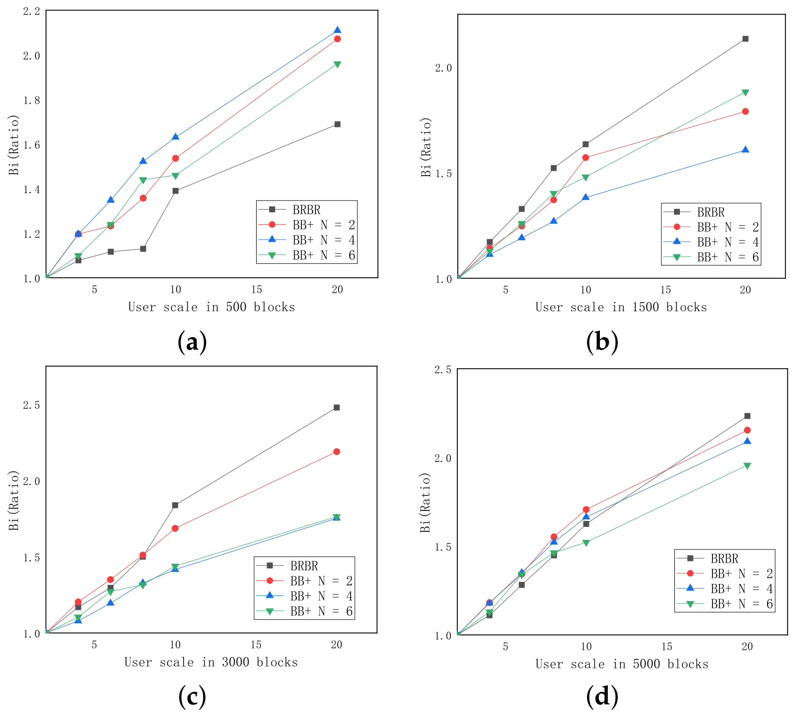

Figure 12. The increasing trend of retrieval time under different concurrent retrieval scales.

Overall, including different concurrent retrieval scales and block sizes, it is similar to the situation in the first experiment. First, in terms of retrieval time, BB+ (N = 4) and BB+ (N = 6) have identical performance and stability, indicating that the increase in Bloom filters does not bring more significant performance improvement to BB+ tree. In some cases with block sizes of 500, 1500, and 3000, BB+ (N = 4) is even better than BB+ (N = 6). The main reason is that the addition of the Bloom filter increases the computational complexity of the retrieval algorithm. There is a balance between the increase in the Bloom filter's bit array and the reduction in the retrieval time.

4.2.3. Multi-Feature Retrieval

This experiment mainly considers that the retrieval system introduces time retrieval parameters and evaluates performance in multiple features retrieval. Firstly, the blockchain retrieval system extracts the account address and time in the Bitcoin data provided by xblock. It inserts the account retrieval item to the BB+ tree and BRBR. It inserts the time retrieval item to the I2B+ tree. Then we conduct the blockchain multi-feature retrieval experiments. The retrieval target is shown in Table 6. The retrieval structure contains the target, and the number of targets is fixed. Random retrieval experiments are conducted 30 times under the scale of 1500 blocks. For multi-feature retrieval, we randomly generate multi-feature attributes, including account address and time, according to the retrieval attributes. BRBR can retrieve the target sender account address without time. Finally, we analyze the comparison results of the retrieval time.

Table 6. Multi-feature retrieval example.

Type	Description	Example
Multi-Feature retrieval	Account address and time retrieval	address = "111113cvaEeMfzNNASCkn49qddBTdX1y4" AND time = "1414771239"

As can be seen from Figure 13a, "BB+" in the figure means that the retrieval only uses the account address. "BB+ with time" means that the retrieval uses the account address and time. N is the bits number in the Bloom filter. We can see that the BB+ tree is better than BRBR in terms of retrieval time, on the whole, indicating that the retrieval structure based on the BB+ tree performs better than BRBR.

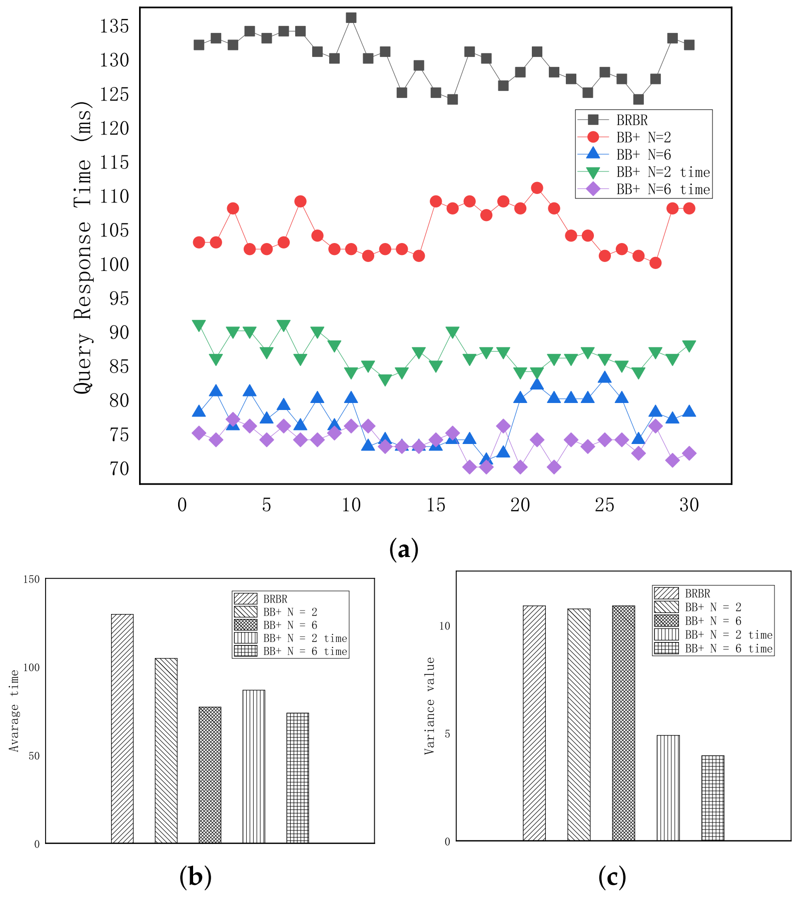

Figure 13. (**a**) Multi-feature query response time; (**b**) average analysis of different structures; (**c**) variance analysis of different structures.

Figure 13b shows the average time of each retrieval structure. According to the analysis of the average retrieval time, the BB+ is improved by 19.25% compared with the BRBR when N is equal to 2. When N is equal to 6, BB+ is improved by 40.54% in retrieval time. BB+ with time is increased by 33.16% compared with BRBR at N = 2. When N is equal to 6, BB+ is improved by 43.16% compared with BRBR. When N is equal to 2, the addition of time features greatly improves the retrieval efficiency. When N is equal to 6, the addition of time features makes the retrieval efficiency improve relatively little. Figure 13c show the stability analysis of each retrieval data structure in these 30 experiments. We can see that the stability of BRBR and BB+ is close, indicating that the Bloom filter does not influence the stability of the B+ tree. However, it can be observed from the curve fluctuation and stability analysis in the figure that the stability of BB+ is significantly improved, mainly due to the addition of the time retrieval feature. After the block set intersection is taken, the retrieval range is narrowed, and the block can be hit more accurately, thus ensuring a fast and stable retrieval algorithm.

5. Conclusions

With the rapid increase in data amount on the blockchain, ledger storage puts tremendous storage pressure on the blockchain nodes. To ease the storage pressure, existing research trends store blockchain ledger data in third-party storage networks to reduce the storage pressure on blockchain nodes. However, after the blockchain ledger is distributed to the network, the blockchain node needs to fetch the ledger frequently from the storage network during transaction retrieval when it is not possible to locate the transaction, resulting in huge network overhead. The blockchain lacks an efficient transaction retrieval system after distributed storage. For this purpose, we extract the universal blockchain transaction features, such as account address and time, and establish a blockchain transaction retrieval system. Based on system, this article first proposes an improved B+ tree—BB+ tree based on the Bloom filter for account address retrieval. Compared with the B+ tree, this kind of data structure has stronger expansibility and flexibility and higher retrieval efficiency. Secondly, the I2B+ tree is introduced to manage the time retrieval effectively. Finally, to verify the blockchain retrieval system proposed in this paper, we implement BRBR to conduct comparative experiments. The experimental results prove that our method has better results in single-feature retrieval and multi-feature hybrid retrieval. The retrieval time is reduced by 40.54% under single feature retrieval, and the retrieval time under multi-feature mixed retrieval is reduced by 43.16%. It has better stability in the face of different block sizes and user scales.

This study focuses on two more common characteristics: time and account address. With the more extensive application of blockchain in the future, there will be more and more data types on the blockchain. This study can be improved according to the specific scene characteristics of blockchain and the content recorded on the blockchain to achieve a better retrieval effect. At present, various application areas have formed initial preparation and accumulation work on blockchain technology, and gradually combine the blockchain system with the original business system, which requires the blockchain system to accelerate the improvement of its own shortcomings and limitations. The blockchain transaction retrieval system proposed in this paper is for the performance improvement and function extension of blockchain system transaction retrieval. It can provide a feasible idea for the future application of blockchain in pan-financial applications, industrial supply chain, information management and other read-intensive scenarios.

Author Contributions: Conceptualization, H.F. and J.W.; methodology, H.F. and J.W.; software and data curation, H.F.; validation and writing—original draft preparation, H.F.; writing—review and editing, H.F., Y.L. and J.W.; supervision, J.W. and Y.L.; project administration, J.W.; funding acquisition, J.W. All authors have read and agreed to the published version of the manuscript.

Funding: The Strategic Leadership Project of the Chinese Academy of Sciences: SEANET Technology Standardization Research System Development (Project No:XDC02070100). Frontier Basic Research Project of Institute of Acoustics of Chinese Academy of Sciences (Contract No: QYJC201902).

Data Availability Statement: Not Applicable, the study does not report any data.

Acknowledgments: We would like to express our gratitude to Jinlin Wang and Yang Li for their meaningful support for this work.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Yuan, Y.; Wang, F.Y. Blockchain: The state of the art and future trends. *Acta Autom. Sin.* **2016**, *42*, 481–494.
2. Treleaven, P.; Brown, R.G.; Yang, D. Blockchain technology in finance. *Computer* **2017**, *50*, 14–17. [CrossRef]
3. Ma, Z.; Jiang, M.; Gao, H.; Wang, Z. Blockchain for digital rights management. *Future Gener. Comput. Syst.* **2018**, *89*, 746–764. [CrossRef]
4. Saberi, S.; Kouhizadeh, M.; Sarkis, J.; Shen, L. Blockchain technology and its relationships to sustainable supply chain management. *Int. J. Prod. Res.* **2019**, *57*, 2117–2135. [CrossRef]
5. Yang, R.; Yu, F.R.; Si, P.; Yang, Z.; Zhang, Y. Integrated blockchain and edge computing systems: A survey, some research issues and challenges. *IEEE Commun. Surv. Tutorials* **2019**, *21*, 1508–1532. [CrossRef]
6. Wang, Q.; Pu, H.; Nie, T.; Shen, D.; Ge, Y. Survey of Data Storage and Query Techniques in Blockchain Systems. *Comput. Sci.* **2018**, *45,* 12–18.
7. Xie, J.; Yu, F.R.; Huang, T.; Xie, R.; Liu, J.; Liu, Y. A survey on the scalability of blockchain systems. *IEEE Netw.* **2019**, *33*, 166–173. [CrossRef]
8. Zheng, Q.; Li, Y.; Chen, P.; Dong, X. An innovative IPFS-based storage model for blockchain. In Proceedings of the 2018 IEEE/WIC/ACM International Conference on Web Intelligence (WI), Santiago, Chile, 3–6 December 2018; pp. 704–708.
9. Chou, I.T.; Su, H.H.; Hsueh, Y.L.; Hsueh, C.W. BC-Store: A Scalable Design for Blockchain Storage. In Proceedings of the 2020 2nd International Electronics Communication Conference, Singapore, 8–10 July 2020; pp. 33–38.
10. Ali, M.S.; Dolui, K.; Antonelli, F. IoT data privacy via blockchains and IPFS. In Proceedings of the Seventh International Conference on the Internet of Things, Linz, Austria, 22–25 October 2017; pp. 1–7.
11. Xu, Q.; Song, Z.; Goh, R.S.M.; Li, Y. Building an ethereum and ipfs-based decentralized social network system. In Proceedings of the 2018 IEEE 24th International Conference on Parallel and Distributed Systems (ICPADS), Singapore, 11–13 December 2018; pp. 1–6.
12. Yli-Huumo, J.; Ko, D.; Choi, S.; Park, S.; Smolander, K. Where is current research on blockchain technology?—A systematic review. *PLoS ONE* **2016**, *11*, e0163477.
13. Tijan, E.; Aksentijević, S.; Ivanić, K.; Jardas, M. Blockchain technology implementation in logistics. *Sustainability* **2019**, *11*, 1185. [CrossRef]
14. Secinaro, S.; Dal Mas, F.; Brescia, V.; Calandra, D. Blockchain in the accounting, auditing and accountability fields: A bibliometric and coding analysis. *Account. Audit. Account. J.* **2021**. Available online: https://www.emerald.com/insight/content/doi/10.1108/AAAJ-10-2020-4987/full/html (accessed on 21 August 2022). [CrossRef]
15. Yu, G.; Nie, T.; Li, X.; Zhang, Y.; Shen, D.; Bao, Y. The challenge and prospect of distributed data management techniques in blockchain systems. *Chin. J. Comput.* **2019**, *42*, 1–27.
16. Jia, D.Y.; Xin, J.C.; Wang, Z.Q.; Lei, H.; Wang, G.R. SE-Chain: A Scalable Storage and Efficient Retrieval Model for Blockchain. *J. Comput. Sci. Technol.* **2021**, *36*, 693–706. [CrossRef]
17. Li, Y.; Zheng, K.; Yan, Y.; Liu, Q.; Zhou, X. EtherQL: A query layer for blockchain system. In *Database Systems for Advanced Applications, Proceedings of the 22nd International Conference, DASFAA 2017, Suzhou, China, 27–30 March 2017*; Springer: Cham, Switzerland, 2017; pp. 556–567.
18. Peng, Z.; Wu, H.; Xiao, B.; Guo, S. VQL: Providing query efficiency and data authenticity in blockchain systems. In Proceedings of the 2019 IEEE 35th International Conference on Data Engineering Workshops (ICDEW), Macao, China, 8–12 April 2019; pp. 1–6.
19. Gupta, H.; Hans, S.; Aggarwal, K.; Mehta, S.; Chatterjee, B.; Jayachandran, P. Efficiently processing temporal queries on hyperledger fabric. In Proceedings of the 2018 IEEE 34th International Conference on Data Engineering (ICDE), Paris, France, 16–19 April 2018; pp. 1489–1494.
20. Chatterjee, K.; Chen, S.C. GeM-tree: Towards a generalized multidimensional index structure supporting image and video retrieval. In Proceedings of the 2008 Tenth IEEE International Symposium on Multimedia, Berkeley, CA, USA, 15–17 December 2008; pp. 631–636.
21. Ren, Y.; Zhu, F.; Sharma, P.K.; Wang, T.; Wang, J.; Alfarraj, O.; Tolba, A. Data query mechanism based on hash computing power of blockchain in internet of things. *Sensors* **2020**, *20*, 207. [CrossRef] [PubMed]
22. Tu, J.; Zhang, J.; Chen, S.; Weise, T.; Zou, L. An Improved Retrieval Method for Multi-Transaction Mode Consortium Blockchain. *Electronics* **2020**, *9*, 296. [CrossRef]
23. Baeza-Yates, R.; Ribeiro-Neto, B. *Modern Information Retrieval*; ACM Press: New York, NY, USA, 1999; Volume 463.
24. Sapon-White, R. Cataloging and Indexing, 1999. Available online: https://www.ala.org/alcts/resources/org/cat/research/subjindclass98 (accessed on 21 August 2022).
25. Yadav, A.K.; Yadav, D.; Prasad, R. Efficient textual web retrieval using wavelet tree. *Int. J. Inf. Retr. Res. (IJIRR)* **2016**, *6*, 16–29. [CrossRef]
26. Rosnan, S.; Abd Rahman, N.; Hatim, S.M.; Ghul, Z.H. Performance evaluation of inverted files, B-Tree and B+ Tree indexing algorithm on Malay text. In Proceedings of the 2019 4th International Conference and Workshops on Recent Advances and Innovations in Engineering (ICRAIE), Kedah, Malaysia, 27–29 November 2019; pp. 1–6.
27. Huang, S.; Wang, H.; Zhang, Y.; Jiang, Y. The Design and Implementation of Indexing System Based on Lucene. *J. Mod. Inf.* **2009**, *7*, 169–171.
28. Aguilera, M.K.; Golab, W.; Shah, M.A. A practical scalable distributed b-tree. *Proc. VLDB Endow.* **2008**, *1*, 598–609. [CrossRef]

29. Rao, J.; Ross, K.A. Making B+-trees cache conscious in main memory. In Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data, Dallas, TX, USA, 15–18 May 2000; pp. 475–486.
30. Schlegel, B.; Gemulla, R.; Lehner, W. K-ary search on modern processors. In Proceedings of the Fifth International Workshop on Data Management on New Hardware, Providence, RI, USA, 28 June 2009; pp. 52–60.
31. Jin, P.; Yang, P.; Yue, L. Optimizing B+-tree for hybrid storage systems. *Distrib. Parallel Databases* **2015**, *33*, 449–475. [CrossRef]
32. Wang, Y.; Zhao, C.; Wang, Z.; Du, J.; Liu, C.; Yan, H.; Wen, J.; Hou, H.; Zhou, K. MLB+-tree: A Multi-level B+-tree Index for Multidimensional Range Query on Seismic Data. In Proceedings of the 2018 5th International Conference on Systems and Informatics (ICSAI), Nanjing, China, 10–12 November 2018; pp. 1176–1181.
33. Fan, B.; Andersen, D.G.; Kaminsky, M.; Mitzenmacher, M.D. Cuckoo filter: Practically better than bloom. In Proceedings of the 10th ACM International on Conference on Emerging Networking Experiments and Technologies, Sydney, Australia, 2–5 December 2014; pp. 75–88.
34. Chang, F.; Dean, J.; Ghemawat, S.; Hsieh, W.C.; Wallach, D.A.; Burrows, M.; Chandra, T.; Fikes, A.; Gruber, R.E. Bigtable: A distributed storage system for structured data. *ACM Trans. Comput. Syst. (TOCS)* **2008**, *26*, 1–26. [CrossRef]
35. Quan, W.; Xu, C.; Vasilakos, A.V.; Guan, J.; Zhang, H.; Grieco, L.A. TB2F: Tree-bitmap and bloom-filter for a scalable and efficient name lookup in content-centric networking. In Proceedings of the 2014 IFIP Networking Conference, Trondheim, Norway, 2–4 June 2014; pp. 1–9.
36. Sasaki, K.; Nakao, A. Packet cache network function for peer-to-peer traffic management with bloom-filter based flow classification. In Proceedings of the 2016 18th Asia-Pacific Network Operations and Management Symposium (APNOMS), Kanazawa, Japan, 5–7 October 2016; pp. 1–6.
37. Anitha, R.; Mukherjee, S. 'MAAS': Fast retrieval of data in cloud using metadata as a service. *Arab. J. Sci. Eng.* **2015**, *40*, 2323–2343. [CrossRef]
38. Rottenstreich, O.; Kanizo, Y.; Keslassy, I. The variable-increment counting Bloom filter. *IEEE/ACM Trans. Netw.* **2013**, *22*, 1092–1105. [CrossRef]
39. Crainiceanu, A.; Lemire, D. Bloofi: Multidimensional bloom filters. *Inf. Syst.* **2015**, *54*, 311–324. [CrossRef]
40. Carneiro, E.; de Carvalho, A.V.; Oliveira, M.A. I2B+ tree: Interval B+ tree variant towards fast indexing of time-dependent data. In Proceedings of the 2020 15th Iberian Conference on Information Systems and Technologies (CISTI), Seville, Spain, 24–27 June 2020; pp. 1–7.
41. Wu, J.; Liu, J.; Chen, W.; Huang, H.; Zheng, Z.; Zhang, Y. Detecting mixing services via mining bitcoin transaction network with hybrid motifs. *IEEE Trans. Syst. Man Cybern. Syst.* **2021**, *52*, 2237–2249. [CrossRef]