electronics

MDPI

Article

An Improved Retrieval Method for Multi-Transaction Mode Consortium Blockchain

Jing Tu [1,*], Jiarui Zhang [1], Shengbing Chen [1], Thomas Weise [1] and Le Zou [1,2,*]

1 School of Artificial Intelligence and Big Data, Hefei University, Hefei 230601, China; zhangjiarui099@163.com (J.Z.); shbchen@hfuu.edu.cn (S.C.); tweise@hfuu.edu.cn (T.W.)
2 Institute of Intelligent Machines, Hefei Institutes of Physical Science, Chinese Academy of Sciences, Hefei 230031, China
* Correspondence: tujing@hfuu.edu.cn (J.T); zoule@mail.ustc.edu.cn (L.Z.)

Received: 19 January 2020; Accepted: 7 February 2020; Published: 8 February 2020

check for updates

Abstract: The traditional method of blockchain retrieval is to search the "Block File" in sequence from the "tail" to the "head" of the blockchain, which always takes a lot of time. How to reduce the retrieval time has been a hot issue in blockchain research. This paper proposes a fast retrieval method for the Multi-Transaction Mode Consortium Blockchain (MTMCB). Firstly, we create a "User Set" and "Block Name Set" cached in Redis. Then, according to the transaction participants and "Block Name Set", we can get the relevant "Block Name List", and quickly obtain the corresponding block files. On this basis, in order to meet the needs of rapid retrieval in large-scale systems, an improved retrieval algorithm based on a B+-tree data structure is proposed. Firstly, the block file information is put into different ordered sets according to the transaction participants, and the B+-tree index is established to quickly get the information of relevant block files by participants. Experimental results show that the improved method of Redis cache retrieval in this paper can greatly increase the efficiency of blockchain retrieval, and can settle some crucial problem in the blockchain application and popularization.

Keywords: consortium blockchain; blockchain retrieval; Redis cache; B+-tree; block storage extension

1. Introduction

The consortium blockchain is one of the three forms of blockchain, which is characterized by a weak centralized network. It has been widely used in many fields, such as asset, credit, time proof of key events, existence proof, and trading market [1–3]. The Multi-Transaction Mode Consortium Blockchain [4] was developed on the basis of retaining some original characteristics of the blockchain, realizing transaction type diversification, virtual nodes, and block cloud storage, allowing attached transaction data, etc., which makes the application of blockchain technology more flexible.

In the blockchain that supports "Bitcoin" transactions, the retrieval algorithm uses sequential searches from the "tail" to the "head" of the blockchain. When the number of blocks reaches a certain size, it takes a long time to retrieve an early block (close to the chain head). Block retrieval is the basis of blockchain-related services, and the existing sequential retrieval methods are inefficient, which seriously affects the performance of its application business system.

In recent years, researchers have proposed some techniques to improve the performance of blockchain retrieval. Google [5] has added Bitcoin, Ethereum blockchain data, and Ethereum classic ETC network plug-ins to BigQuery, and has employed artificial intelligence to make the blockchain searchable. Wang [6] proposed to establish a simple database index directory for medical records in medical blockchain applications, including the hash value information of the relevant blocks of the department and patient medical records. Ren [7] proposed a method, DCOMB (dual combination Bloom filter), combining the data stream of the IoT (Internet of Things) with the timestamp of the blockchain, to improve the versatility of the IoT database system. Shibata [8] proposed a retrieval scheme, where a client provides a computer program called a searcher that implements a randomized search algorithm such as a genetic algorithm. Lv [9] proposed a retrieval model based on the combination of chain and chain: the method builds an inverted index for the log data, and expands the block chain header node to store the inverted index in it. The subsequent log retrieval achieves the purpose of rapid positioning by sequential retrieval of the inverted index. Zhou [10] proposed a ledger data query platform called Ledgerdata Refiner. With ledger data analysis middleware, the platform provides sufficient interfaces for users to retrieve block or transaction efficiently. Do [11] proposed a private keyword search component designed for searching in the encrypted dataset. However, these studies generally either complicate the hierarchical structure of the blockchain itself by introducing third-party support, or the search performance improvement effect is limited.

Many experts have put forward patent applications for invention in the field of blockchain retrieval since 2015. The method for distributing and retrieving data on a blockchain network with peer nodes developed in [12] forms a blockchain network by forming peers, sharing and distributing private files, and sending messages to complete the request and private retrieval. The search method and system for business information of blockchain given in [13] improves the retrieval efficiency of blockchain business information by establishing a business-related index database through the unified use of personnel names, and does not need to rearrange the block content. The personalized privacy information retrieval method based on the blockchain as discussed in [14], encrypts the data through the encryption algorithm of the buyer and seller on the data trading platform, and decrypts the ciphertext with their own public key encryption algorithm to obtain the retrieval results and achieve content retrieval and intention privacy protection. These studies need to use relational databases or encryption algorithms, and database maintenance needs to monitor the block modification repeatedly. Although the efficiency has been improved, the block file still cannot meet the real-time requirements of the business system when it is large, and needs large system consumption.

In this paper, a fast blockchain retrieval method for the Multi-Transaction Mode Consortium Blockchain is proposed, which mainly includes the following:

1. A "Block Name File" is defined, which establishes an "index mechanism" with user files and block files, so as to improve the retrieval efficiency without affecting the security of user information.
2. According to the problem that the retrieval efficiency will decline with the increase of "Block Name File", combined with the excellent read–write performance of the Redis memory database, a fast retrieval method of "block name collection" under Redis cache is proposed.
3. In order to mitigate the problem that the retrieval efficiency of the "Block Name Set" will decline under the large block size and large user scale, a new "User Block Set" is designed to replace "Block Name Set" to participate in the retrieval, and a B+-tree index is introduced to improve the retrieval process.

Our experiments show that the algorithms proposed in this paper can significantly improve the retrieval efficiency, and the improved algorithm improves the retrieval efficiency even further and has better stability.

2. Background

2.1. Multi-Transaction Mode Consortium Blockchain (MTMCB)

Multi-Transaction Mode Consortium Blockchain (MTMCB) inherits several features from the existing Blockchain technologies, such as relative decentralization, distributed storage, point-to-point communication, and secure encryption. On this basis, MTMCB generalizes "electronic currency transaction" in blockchain technology into "process and result of event processing", and redesigns "transaction verification mechanism" and "block distribution storage mechanism". As shown in

Figure 1, MTMCB includes the Regulatory Node System (RNS) and the Transaction Node System (TNS). The RNS is deployed on the server, performing initialization, transaction processing, and audit services. The initialization operation only needs to be performed once. Transaction nodes can be PCs, mobile devices, or even automatic teller machines (ATMs), etc.

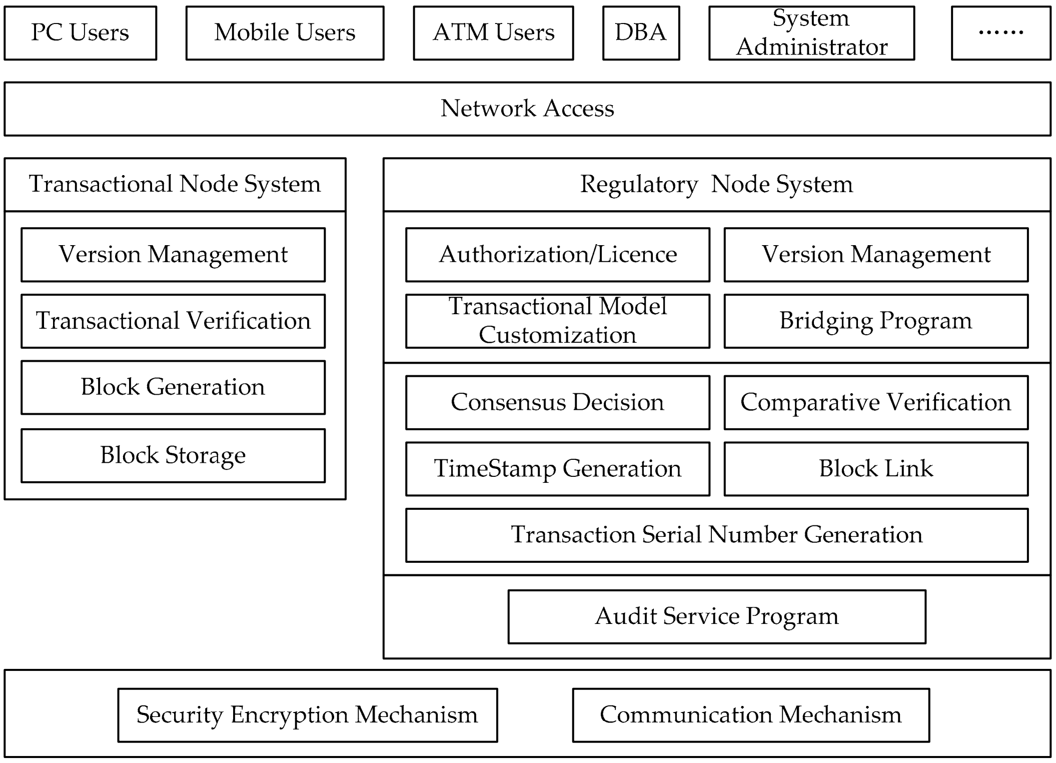


Figure 1. The overall architecture diagram of MTMCB.

After authorization, different user types write their initialization information into the "User File". For example, the storage structure is shown in Figure 2.

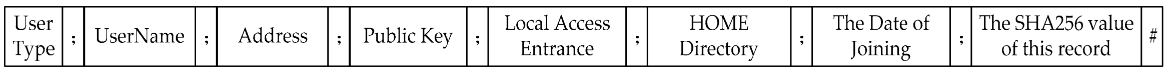


Figure 2. The schematic diagram of the storage structure of User Files.

In the MTMCB, the Regulatory Node is responsible for managing the "User File", which is used to record the correspondence between the user name and its address, where "Address" is 10 characters and records the user's transaction address. In the subsequent transaction records and retrieval process of the user, only the "Address" of the transaction participant is used instead of the "User Name", which effectively protects the privacy of the transaction participant.

The block includes a block header and a block body, as shown in Figure 3. The block body includes the transaction information and its SHA256 [15] value.

The first block in the blockchain is the genesis block. Its block number is distinguished by a specific hash value. Each block (except the genesis block) stores the hash value of the previous block. In this way, the blockchain is formed between the blocks, and the information of the chain tail block is stored and updated by the chain tail file (system file, which is used to store the file name of the chain tail block of the blockchain and the hash value of the bank). Using the hash value of the previous block, we can retrieve from "tail" to "head".

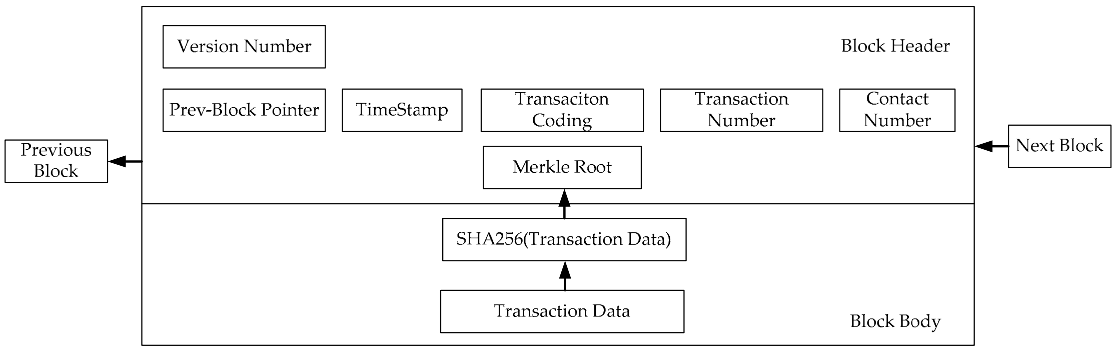


Figure 3. The structure diagram of Block.

2.2. Redis Cache Technology

For the retrieval of block files stored on disk, a corresponding index needs to be established in memory to reduce the number of disk reads and writes and speed up the retrieval speed. Currently, Redis cache technology is usually used to achieve fast indexing of block files. Redis [16,17] is a high-performance memory-based Key-Value database, which mainly solves the problem of timeliness of data processing in the case of high concurrency in relational databases. Because of its pure in-memory operation, Redis has excellent read and write performance. Throughputs of more than 100,000 read and write operations per second have been recorded [18]. Redis provides rich data types, including linked lists (List), strings (String), sets (Set) and ordered sets (Zset). All Redis operations are atomic, and Redis also supports the atomic execution of several operations. It also supports the calculation of unions, intersections and complements of sets in the service front-end, and supports a variety of sorting. At the same time, it provides APIs for Java, python, ruby, PHP, Erlang and other clients, which is suitable for various implementation occasions. Additionally, Redis also provides publish /subscribe, notification, key expiration, and other features.

One of the most important application scenarios of Redis is as business cache, which is used to keep some hot data that are not frequently changed but frequently accessed in memory [19], effectively reducing the number of database reads, reducing database pressure, improving response time, and enhancing throughput. Redis also provides the compressed lists as a data structure to store a series of data and its encoding information in a continuous memory area. The purpose is to reduce unnecessary memory overhead as much as possible under certain controllable time and complex reading conditions.

2.3. B+-Tree Index

B+-tree [20] is a high balance tree, which has the advantages of high self-balance, low update cost and high query efficiency. An m-order B+-tree is either an empty tree or satisfies the following characteristics [21]:

- Each node in the tree has, at most, M subtrees.
- If the root node is not a leaf node, there are at least two subtrees.
- If the Non-terminal nodes is not the root, there are at least one subtree.
- Non-leaf nodes with K subtrees contain exactly $k - 1$ keywords.
- All non-leaf nodes contain the following information: data ($P0$, $K1$, $P1$, $K2$, . . . , $Pn - 1$, $Kn - 1$, Pn), where Ki ($i = 1, 2, \ldots, n$) are the keywords and $Ki < Ki + 1$. Pi ($i = 0, 1, \ldots, n$) are the pointers to the root node of the subtree.
- The keywords of all nodes in the subtree indicated by the pointer $Pi - 1$ are less than Ki.
- The keywords of all nodes in the subtree indicated by Pn are greater than Kn.
- All leaf nodes are in the same layer of the tree structure, and contain the actual information.

When querying the B+-tree [22] with the goal to find all records whose key is K, one first needs find the minimum key Ki greater than K in the root node and then follows the pointer $Pi - 1$ on the left of Ki to the node of layer 2. If K is greater than all of the key words in the root node, then follow the pointer Pn to the node of layer 2. In layer 2, the same method is applied to find the pointer to the node in layer 3, and so on, until, finally, a record pointing to the data file is found in the leaf node.

As a kind of balanced multi-fork tree, the B+-tree only has leaf nodes to store data, and the inner nodes are only used for indexing. It has significant advantages in searching external data (that is, disk data). Due to the long seek time and rotation delay time of traditional disks when reading and writing, it is necessary to reduce the disk IO times as much as possible when searching. The best case is to find the target index quickly, and then read the data from the disk. Using B+-trees can achieve this and has become the main search optimization technology of the mainstream database.

3. Application of Redis in Block Retrieval Algorithm

From the structure of the blockchain, each block stores the hash value (block file name) of its previous block. The blockchain is connected by one block hash value, but this connection is unidirectional, and the previous block can only be found by the latter block, otherwise it is not possible. Therefore, the traditional retrieval method obtains the hash value of the end-of-chain block from the end-of-chain file, and then finds the previous block through this hash value, and so on. Its sequential search time complexity is $O(N)$. As the blockchain system is put into use, new blocks will continuously be generated and added to the chain. As time goes by, the number of blocks will become, larger and larger. The efficiency of this sequential retrieval method will decline rapidly. In serious cases, it may cause long-term operation, affecting the performance of the system.

Based on the characteristics of the weakly centralized network in the MTMCB, we introduce a "Block Name Set" by using the MTMCB Regulatory Node System (RNS). An index between user files and blocks is established and held in a Redis cache, so as to achieve the purpose of fast block name search and fast block content location.

3.1. Definition and Construction of "Block Name Set"

The "Block Name File" is a specific format of the consortium blockchain system file, which is managed by the regulatory node. When the blockchain system is storing blocks, the block name of the block and the corresponding addresses of all transaction participants of the block are simultaneously stored in the "Block Name File". The logical structure of "Block Name File" storage is shown in Figure 4.

Block Name	;	UserAddress	;	UserAddress	;	······	;	The SHA256 value of this record	#

Figure 4. The logical structure of "Block Name File".

Among them,

- "Block name" is a 32 byte hexadecimal number representing the block name of a block;
- ";" is the separator;
- "User Address" is a hexadecimal number of 20 bytes, representing the address of the transaction participants of this block;
- "The SHA256 value of this record" is a 32 byte hexadecimal number, which is used to verify whether the data in this record have been tampered with;
- "#" is the end character, indicating the end of this record.

Considering the high retrieval frequency of the "Block Name File" and "User File", the read performance of disk storage cannot meet the demand of fast retrieval. When the system is started, the two files are read and parsed, and stored in the Redis cache database in the form of "Block Name Set" and "User Set".

Redis is a NoSQL database technology based on the memory key-value structure [23]. By building a Redis database cluster, data are cached in memory and master–slave replication is realized, so that the high-frequency accessed data can be read directly from memory, effectively reducing the corresponding time of data query. The traditional relational database maps the logical model to a series of tables. Redis needs to map the logical relationship to one or multiple key value pairs. Therefore, the choice of the key structure is important. A reasonable design can effectively improve the query efficiency and save memory overhead.

In this retrieval process, the "Block Name Set" and "User Set" information are cached and stored, and their logical structure is shown in Figure 5.

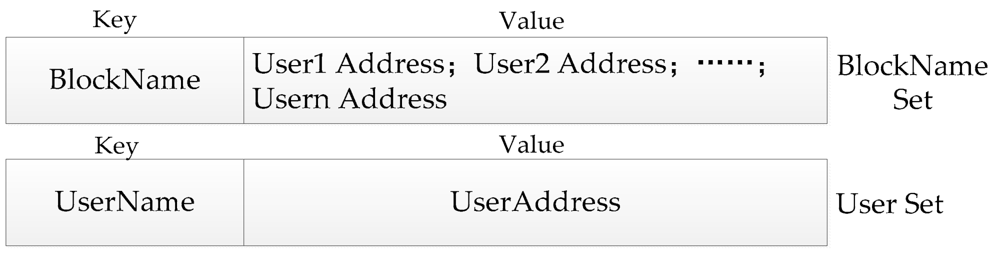


Figure 5. The logical storage structure of "Block Name Set" and "User Set".

The traditional blockchain system only stores the block information after consensus verification, and the blockchain storage extension processing is to store the block name of the block and the corresponding addresses of all transaction participants of the block into the "Block Name File" and cache while realizing the blockchain system's block storage. The specific methods are as follows:

Step 1: When the block is stored, a block file will be generated, which records the contents of the current block. For convenience, the file is named after the hash value of the block.

Step 2: Obtain the block file name of the block file at the same time of the block storage, extract the "User address" of one or more participants in the current block, and write the "Block Name File" in the block name file storage structure.

Step 3: Record the SHA256 value of the new block name file record memory for anti–tamper requirements.

Step 4: Update the "Block Name Set" in the Redis cache.

3.2. Block Retrieval Algorithm through Redis Cache "Block Name Set"

When retrieving data, first use the "User Set" in the Redis cache to determine the address of the user (participant), and then search in the "Block Name Set" according to this address to obtain the name of the block to be retrieved. The specific methods are as follows:

Step 1: Obtain the names of one or more participants to be retrieved. Search the user set to find the user address from the user name. The user name here must be information that can identify the unique user to prevent users with the same name from affecting the search results. The system uses an ID number as the user name.

Step 2: In the "Block Name Set", match the line by line by "User address" to find the block file name containing this user.

Step 3: Find the block from the block folder, parse the data and return it as query result.

The specific process is shown in Figure 6.

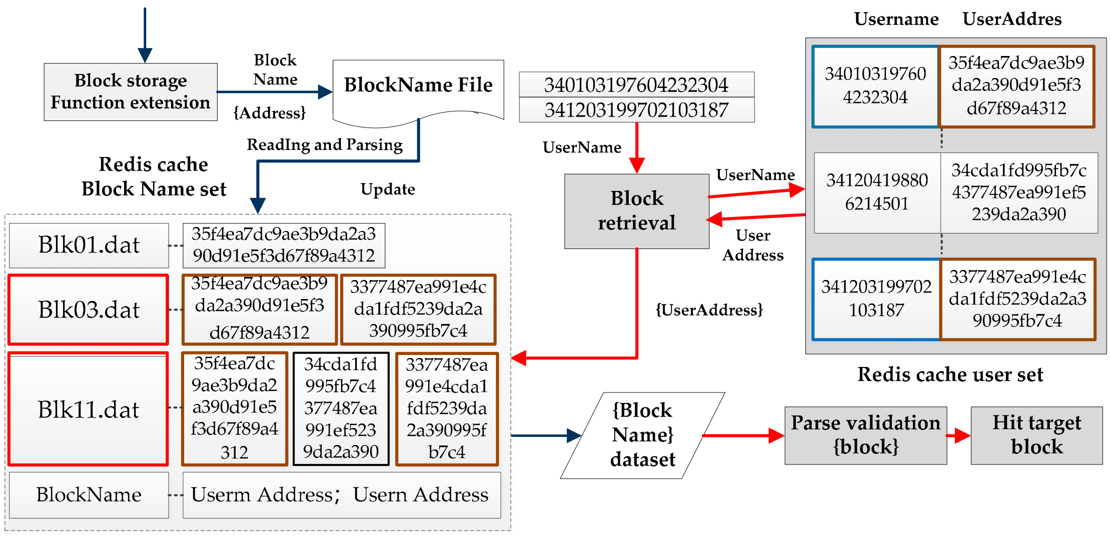


Figure 6. Block file retrieval process.

Through the above process, the block name file is searched in the Redis cache to obtain the file name list of all the blocks in which the queried user participates, and finally, the block files in the file name list are processed one-by-one in the disk file.

The differences between the application of Redis in block retrieval method and the traditional retrieval method are show in Figure 7.

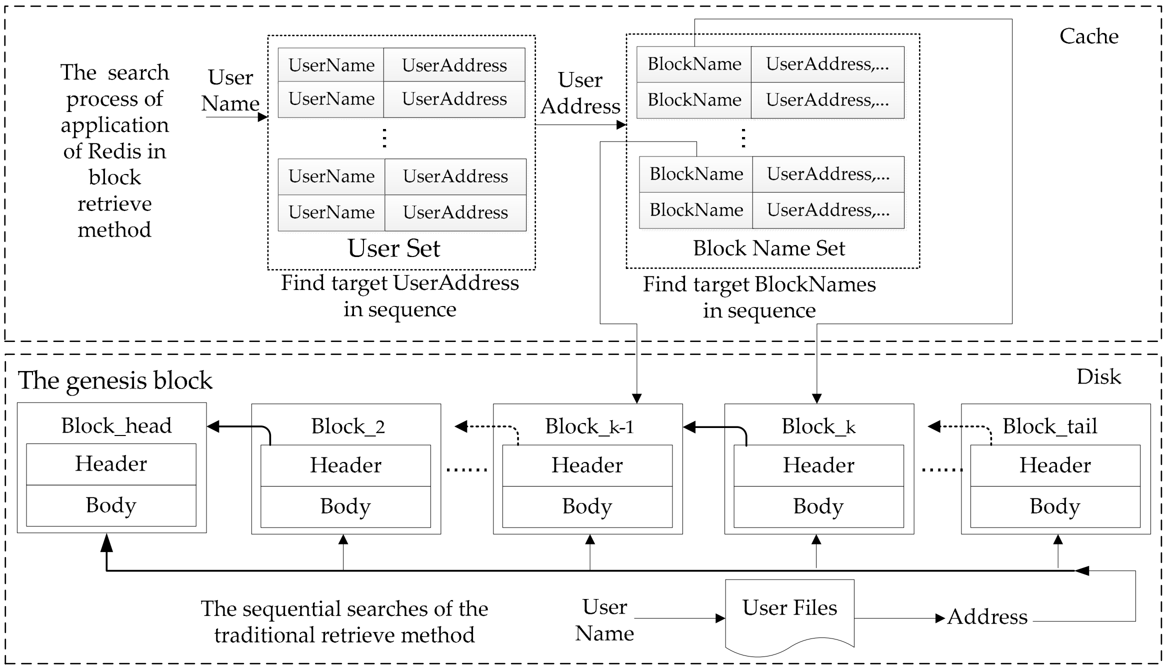

Cache
The search process of application of Redis in block retrieve method
User Name
UserName
UserAddress
User Set
Find target UserAddress in sequence
User Address
BlockName
UserAddress,...
Block Name Set
Find target BlockNames in sequence
Disk
The genesis block
Block_head
Block_2
Block_k-1
Block_k
Block_tail
Header
Body
The sequential searches of the traditional retrieve method
User Name
User Files
Address

Figure 7. The differences between two retrieval methods (the application of Redis in block retrieval method and the traditional retrieval method).

The above process is implemented by Algorithm 1. RetrieveBlockNames(), as shown in "Block Name Set" retrieval algorithm.

Algorithm 1. RetrieveBlockNames(UserNames[1..*n*])

Input: UserNames[1..n]
Output: BlockNameSet

```
BlockNameSet←null
FileName←null
UserAddresses←null
// Search the "User Set" to find the UserAddresses from the UserNames
for i←1,..,n do
  Find the "UserAddress" value corresponding to UserName[i] in "User Set"
  if can be found then add a new Value to UserAddresses
  else throw an exception and return
end for
// Match the line by line by UserAddresses to find the block filenames in the "Block Name Set"
  vRead a record in the "Block Name Set" in Redis cache
do {
    Read the "Key" value and put it into FileName
    Match the "Value" value with UserAddresses
    if UserAddress ⊂ the "Value" value
      BlockNameSet.add(FileName)
    Read next record in the "Block Name Set"
    if there is no new record then exit the loop
}loop
    return BlockNameSet
```

4. Application of B+-Tree and Redis Cache in the Improved Block Retrieval Algorithm

The method of building the "Block Name Set" in the Redis cache can eliminate the inefficiency of traditional block chain search methods, which is from the end of the chain to the beginning of the chain one-by-one. However, as the system is put into operation for a longer and longer time, the number of users becomes larger and larger, resulting in a sharp increase in the number of transactions and blocks. According to Algorithm 1, each retrieval needs to traverse all the "Block Name Set". When the number of users participating in the transaction increases, the computational complexity will increase fast. To solve the above problems, this paper uses a B+-tree to reconstruct the user and block name storage structure in the Redis cache, and proposes an improved retrieval algorithm.

4.1. Improved Redis Cache Storage Model

The starting point of the retrieval method in the original MTMCB is "user name", and the search object is block files related to the "user name" of the transaction party. In order to achieve this function, the "User Name" substitute value "User Address" is used as the Key, and the combination of block file names is used as the Value that data type is "Set". The storage structure model is shown in Figure 8.

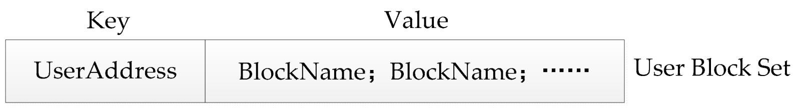


Figure 8. The Data storage structure of "user block collection" in Redis cache.

In order to be different from the "Block Name Set" in the original retrieval algorithm, this data file is named "User Block Set", which is used to store the block file name information of each user transaction and replace the "Block Name Set" in the Redis cache in the original retrieval algorithm. In this improved method, only "User Set" and "User Block Set" are stored in the Redis cache.

Although the reading of the Redis cache is relatively fast, the average time complexity of adding compressed lists is $O(N)$. As N increases, the time consumption will also increase. Due to the large number of users participating in the transactions in the MTMCB, the number of users and files in the Redis cache is large, and the response time to retrieve multiple "User Addresses" still cannot meet the needs of the business system. In order to improve the retrieval efficiency, we use a B+-tree index to improve the retrieval efficiency of the "User Address". The logical structure of this metadata index is shown in Figure 9.

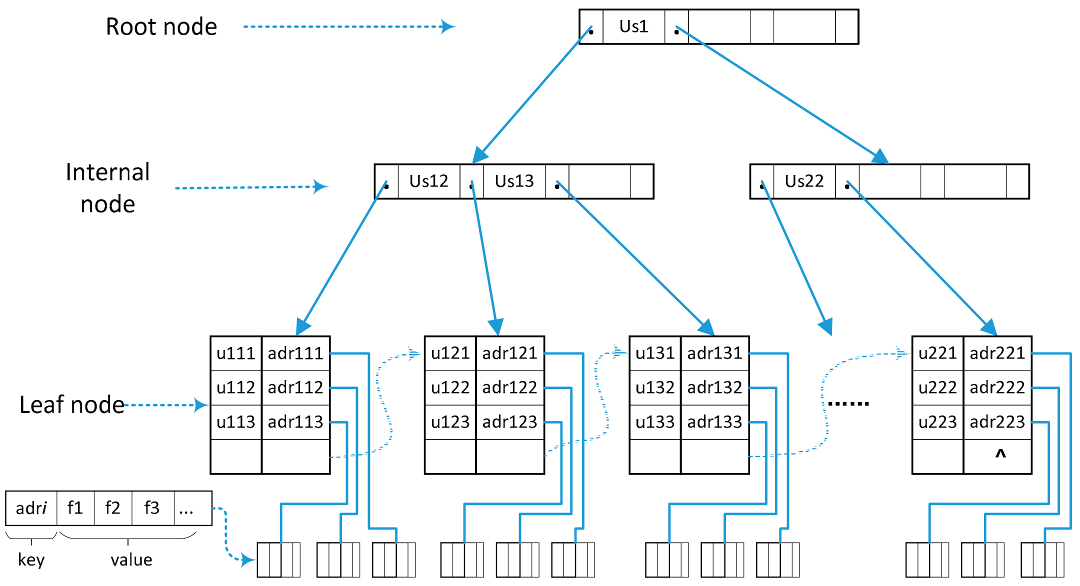


Figure 9. The B+-tree index structure of "User Block Set".

4.2. Initialization of Redis Cache Storage

1. When the system starts, read and analyze the "Block Name File", and initialize the "User Block Set" Redis database according to the "User address" and "Block Name" content in the block name file.
2. According to the Algorithm 2 in B+-tree Algorithm, the B+-tree index of "User Block Set" is generated, and the corresponding relationship between the index and "User Block Set" is established.
3. Considering that the "User Set" also has many transaction users and low efficiency of sequential retrieval, another B+-tree index is established for "User Files" in a similar fashion, with the user name as keyword.

4.3. "User Block Set" Construction Algorithm

According to the storage expansion processing in the original MTMCB system, after the "Block Name File" is expanded, the block name information is stored in the "User Block Set" according to the transaction participant information.

The specific methods are as follows:

Step 1: When the block is stored, a block file will be generated, which records the contents of the current block. For convenience, the file is named after the hash value of the block.

Step 2: There are one or more transaction participants in the event processing of the system Select one transaction participant and find the corresponding "User address" in the B+-tree index with the user name from the "User Set".

Step 3: Find the Key in the "user block file" according to the participant's address. If it exists, extend the value, write the block name to the end of the value, and sort by the time-stamp order. If it does not exist, add the key-value metadata. The key is the participant's address, the value is the new block file name, and update the B+-tree index.

Step 4: Repeat step 2 and step 3 to process the next transaction participant until all participants finish processing and complete this extension operation.

The above process is implemented by Algorithm 3. AppendUserfile(), as shown in "User Block Set" extension algorithm.

Algorithm 2. InsertTree(nodepointer,address)

Input: nodepointer, address
Output: newchildentry

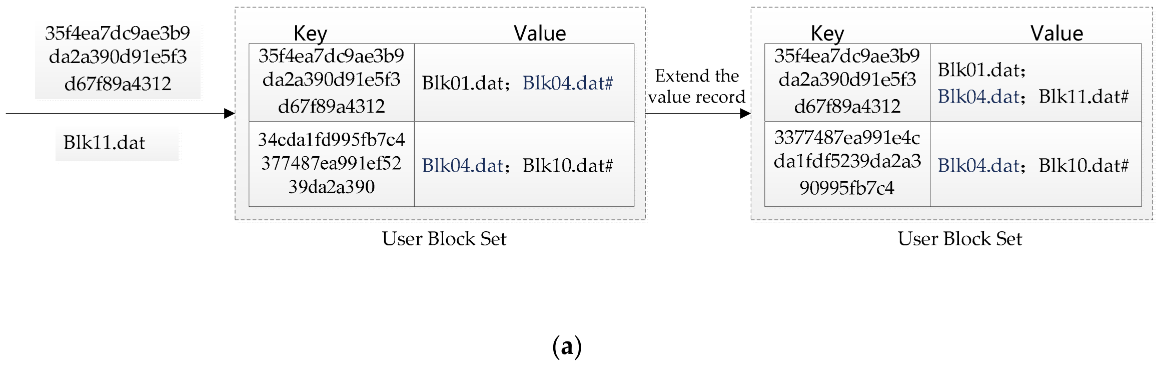
```
newchildentry←*null*
*if* nodepointer is not a leaf node *then*
    N←nodepointer
    Find $i$ satisfying the condition: $Ki \leq$ the code value of address< $Ki + 1$
      newchildentry←InsertTree($Pi$, address);
      *if* newchildentry is *null then return null*
      *else*
            *if* $N$ has space *then*
                  Put newchildentry into $N$
                  newchildentry←*null*
            *else*
        Split $N$; //$M$ code values and $M$ node pointers
                  The first $M$/2 code values and $M$/2 node pointers are left;
        The last $M$/2 code values and $M$/2 node pointers are moved into the new node $N$2;
            *if* $N$ is Root Node *then* Split root node;
        Create a pointer to $N$ so that the pointer of the root node of the tree points to the new node;
      *return* newchildentry
*if* nodepointer is leaf node then
    $L$←nodepointer
      *if* $L$ has space *then*
      Put address into $L$
      newchildentry←*null return null*
      *else*
      Split leaf node $L$ into two left and right leaf nodes;
      The first $L$/2 data items are left in $L$1, and the rest are moved to the new leaf node $L$2;
      Set $L$ and $L$2 link pointers;
      *return* newchildentry
```

Algorithm 3. AppendUserFile(UserAddress[1..*n*],BlockName)

Input: UserAddress[1..n],BlockName
Output: null

```
*for i*←1,..,$n$ do
    Read UserAddress[$i$] and use the B+ -tree index to find out if the value exists in the "User Block set"
    *if* can be found *then*
        Extend the BlockName to the value of the "Key" corresponding to UserAddress[$i$]
    *else* Add a new metadata for the "User Block Set", the "Key" value is UserAddress[$i$], and the "Value"
value is BlockName
*end for*
```

Taking two user addresses "35f4ea7dc9ae3b9da2a390d91e5f3d67f89a4312" which is the existing user address and "3377487ea991e4cda1fdf5239da2a390995fb7c4" which is not existing as examples, the extension process is shown in Figure 10. If the user address exists, extend the "User Block Set", otherwise add a new metadata for the set.


35f4ea7dc9ae3b9 da2a390d91e5f3 d67f89a4312
Blk11.dat
Key
Value
35f4ea7dc9ae3b9 da2a390d91e5f3 d67f89a4312
Blk01.dat；Blk04.dat#
34cda1fd995fb7c4 377487ea991ef52 39da2a390
Blk04.dat；Blk10.dat#
User Block Set
Extend the value record
Key
Value
35f4ea7dc9ae3b9 da2a390d91e5f3 d67f89a4312
Blk01.dat；Blk04.dat；Blk11.dat#
3377487ea991e4c da1fdf5239da2a3 90995fb7c4
Blk04.dat；Blk10.dat#
User Block Set
(a)

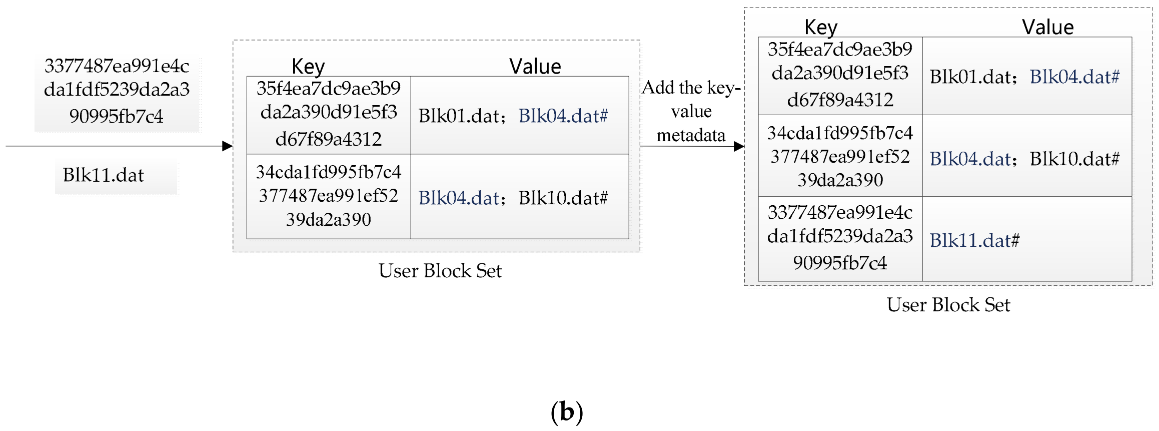

3377487ea991e4c da1fdf5239da2a3 90995fb7c4
Blk11.dat
Key
Value
35f4ea7dc9ae3b9 da2a390d91e5f3 d67f89a4312
Blk01.dat；Blk04.dat#
34cda1fd995fb7c4 377487ea991ef52 39da2a390
Blk04.dat；Blk10.dat#
User Block Set
Add the key-value metadata
Key
Value
35f4ea7dc9ae3b9 da2a390d91e5f3 d67f89a4312
Blk01.dat；Blk04.dat#
34cda1fd995fb7c4 377487ea991ef52 39da2a390
Blk04.dat；Blk10.dat#
3377487ea991e4c da1fdf5239da2a3 90995fb7c4
Blk11.dat#
User Block Set
(b)

Figure 10. Examples of “User Block Set” extension algorithm: (**a**) The extended UserAddress exists in the metadata, extending the value record; (**b**) The extended UserAddress does not exist in the metadata, new metadata is added.

4.4. Application of B+-Tree and Redis in Improved Block Retrieval algorithm

After the introduction of the “User Block Set”, the block retrieval algorithm no longer relies on the “Block Name Set”, but establishes an association relationship between the “User Set” and the “User Block Set”. Assume that when the system needs to query all block files in which two users participate together. As long as the corresponding two addresses are found through the user file, and then the intersection of the set of block file names corresponding to the two addresses is found in the user block file, one can quickly get all the block files in which the users in question participate together. The retrieval process is shown in Figure 11.

The specific methods are as follows:

Step 1: The retrieval method receives the names of one or more participants, and finds the “User address” from the “user name” by retrieving the “User Set”.

Step 2: Find the Key where the “User Address” is located through the B+-tree index in the “User Block Set” and obtain all the block file name records of the user.

Step 3: Repeat step 2 until all relevant block file name records of the transaction participants are found.

Step 4: According to the block name file record set obtained in step 2 and step 3, a block file related to all participants is obtained, that is, the target file for this query.

Use the B+-tree to construct the fast location of participants and users and obtain the block file name based on the association between the “User Set” and the “User Block Set” in the Redis cache. Finally, read the “block file” on the disk for verification and processing, to locate the target block.

The above process is implemented by Algorithm 4. RetrieveBlockFiles (), as shown in Block retrieval algorithm with “User Block Set”.

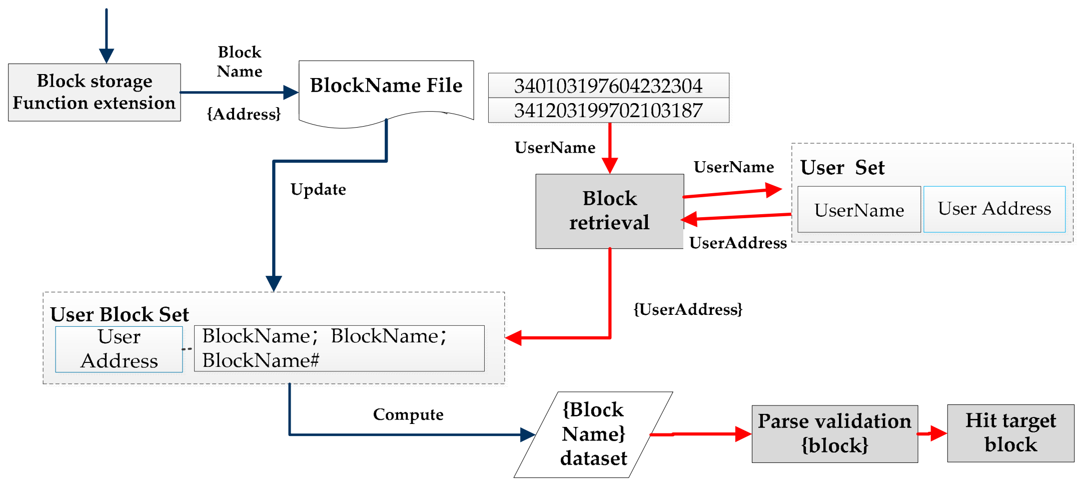


Figure 11. The block retrieval method with "User Block Set".

Algorithm 4. RetrieveBlockFiles(UserName[1..*n*])

Input: UserName[1..n]
Output: BlockFilesSet

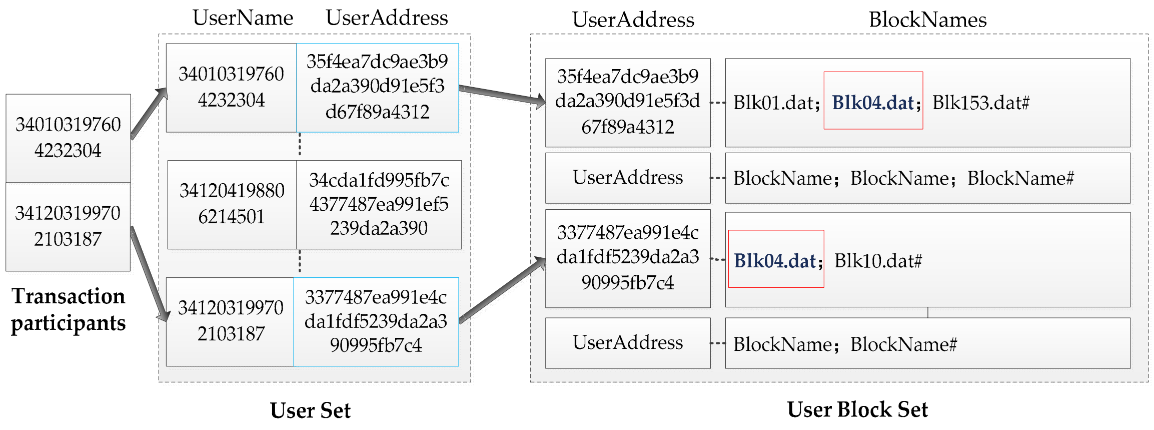
```
BlockFilesSet←null
UserAddress←null
FileSet←null
// Search the "User Set" to find the UserAddresses from the UserName[1..n]
for i←1,..,n do
    Read UserName[i] and use the B+ -tree index to find out if the value exists in the "User set"
    if can be found then
        store the "Value" value of the corresponding record in "User Set" into UserAddress[i]
    else throw an exception and return
end for
// Find the Key where the "User Address" is located through the B+-tree index in the "User Block Set" and
obtain all the block file name records of the user
for i←1,..,n
    Read UserAddress[i] and use the B+ -tree index to find out if the value exists in the "User Block set"
    if can be found then
        store the "Value" value of the corresponding record in "User Block Set" into FileSet
    else throw an exception and return
    if BlockFilesSet is null then
        BlockFilesSet=Fileset
    else
        Store the intersection of BlockFilesSet and FileSet in BlockFilesSet
end for
return BlockFilesSet
```

Taking participants "340103197604232304" and "341203199702103187" as examples, the search process is shown in Figure 12. According to the block name set of the two, the intersection "blk04. dat" is the target block for storing the transactions of the two.

4.5. Time Complexity Analysis of Retrieval Algorithm

In this paper, the search algorithm time consists of two parts: B+-tree search time and "User Block Set" intersection time.

In order to illustrate the search time of B+-tree, suppose that the search probability of any item in the tree is *p*, the total number of users in the consortium blockchain is *M*, the total number of block files is N, the order of B+-tree is d, the number of orders is t, the average space utilization is f, $0.5 \leq f \leq 1$ (f = used storage space/maximum available storage space), and each user participates in m blocks on average, then the relationship shown in Table 1 can be obtained.



UserName
UserAddress
UserAddress
BlockNames
34010319760 4232304
34120319970 2103187
Transaction participants
34010319760 4232304
35f4ea7dc9ae3b9 da2a390d91e5f3 d67f89a4312
34120419880 6214501
34cda1fd995fb7c 4377487ea991ef5 239da2a390
34120319970 2103187
3377487ea991e4c da1fdf5239da2a3 90995fb7c4
35f4ea7dc9ae3b9 da2a390d91e5f3d 67f89a4312
Blk01.dat; Blk04.dat; Blk153.dat#
UserAddress
BlockName; BlockName; BlockName#
3377487ea991e4c da1fdf5239da2a3 90995fb7c4
Blk04.dat; Blk10.dat#
UserAddress
BlockName; BlockName#
User Set
User Block Set

Figure 12. Example block retrieval diagram for "User Block Set".

Table 1. B+-tree series, number of Block files and space requirements.

The Number of Orders	The Number of Nodes	Average Number of Users (Subtree)	The Number of Block Files	Space Requirement of the Tree
1	1	fd	$m * fd$	1
2	$fd+1$	$fd*(fd+1)$	$m*fd*(fd+1)$	$fd+1$
……	……	……	……	……
t	$(fd+1)^{t-1}$	$fd*(fd+1)^{t-1}$	$m*fd*(fd+1)^{t-1}$	$(fd+1)^{t-1}$

In the B+-tree of this paper, all users are in the leaf node, so the number of users M can be expressed as:

$$M = fd*(fd+1)^{t-1} \tag{1}$$

Therefore, the number t of orders of the B+-tree can be expressed as:

$$t = LOG_{(fd+1)}M - LOG_{(fd+1)}(fd) + 1 = LOG_{(fd+1)}M + C \text{ (Cis a constant close to 1)} \tag{2}$$

Similarly, the relationship between t and the total number of block files N can be obtained:

$$t = LOG_{(fd+1)}N - LOG_{(fd+1)}(m*fd) + 1 = LOG_{(fd+1)}N + C' \text{ (}C'\text{is a constant less than 1)} \tag{3}$$

For the intersection time of user block files in the Redis cache, since the user block names have been sorted (Ordered Set), it can be known from the ordered set algorithm [24] that the time complexity is $O(m)$.

Therefore, the time complexity of the improved retrieval algorithm in this paper is $O(LOG_{(fd+1)}M)$ + $O(m)$ = $O(LOG_{(fd+1)}M + m)$. Among them, m is the average number of blocks that each user participates in, and m is far less than N.

In terms of spatial complexity, the traditional retrieval method is $O(N)$. The improved retrieval algorithm in this paper mainly adds the index structure of the B+-tree. The spatial complexity of the internal nodes of B+ is $O(N/fd)$, so the total spatial complexity is $O(N + N/fd)$.

The traditional block retrieval directly reads data from the disk in the order of the blockchain, and the time for each read from the disk is assumed to be H. Based on the analysis above, we present the time complexity of three retrieval methods (traditional block retrieval, application of Redis in block retrieval, and application of B+-tree and Redis in improved block retrieval) in Table 2:

Table 2. Complexity analysis of three block retrieval algorithms.

Retrieval Algorithms	Retrieval Steps	Retrieval Method	Time Complexity	Space Complexity
Traditional block retrieval, BR [1] algorithm	Retrieve block file	Sequential search	$O(H^*M)$	$O(M)$
RBR [2] algorithm	Retrieve user *i* and user *j* in the User Set	Sequential search	$O(N)$	$O(N)$
	Retrieve Block Name Set	Sequential search	$O(M)$	$O(M)$
	Retrieve block file	Read directly	$O(H^*1)$	$O(M)$
BRBR [3] algorithm	Retrieve user *i* and user *j* in the User Set	B+-tree index	$O(LOG_{(fd+1)}N)$	$O(N + N/fd)$
	Retrieve User Block Set	Ordered intersection	$O(m)$	$O(M)$
	Retrieve block file	Read directly	$O(H^*1)$	$O(M)$

[1] BR: the Traditional Block Retrieval algorithm; [2] RBR: Application of Redis in Block Retrieval algorithm; [3] BRBR: Application of B+-tree and Redis in Improved Block Retrieval algorithm.

In terms of space complexity, since the number of users N is less than the number of block files M, the space complexity of all three retrieval methods is $O(M)$. In terms of time complexity, BR, RBR, BRBR three block retrievals are: $O(H^*M)$, $O(N + M)$ and $O(LOG_{(fd+1)}N + m)$. When N and M are large, the efficiency of the BRBR algorithm in this paper is more advantageous.

4.6. Privacy and Security Concerns of Non-Regulatory Nodes

In terms of tamper-proof performance, the MTMCB has adopted two methods to ensure security:

1. Blockchain cloud storage transformation algorithm: The complete blockchain is stored in the local and corresponding cloud of the regulatory node, and the blockchain stored in the cloud is the transformation of the local blocks;
2. Distributed storage: The blocks generated by the same exchange will be distributed and stored in the cloud directory of all relevant participants in its transactions.

Among them, the privacy and security concerns of non regulatory nodes (transaction nodes) includes:

1. All transactions recorded in the block use "UserAddress" instead of "UserName", which well isolates the privacy of transaction participants;
2. The transaction node only stores the blocks related to the participant, not the complete blockchain, and the system adopts the "virtual node" mechanism. All the blocks of the transaction node are stored in the virtual node of the cloud corresponding to the node, which indirectly realizes the block security with the help of cloud security.
3. The structure of the block is transparent to the regulatory node. The regulatory node has a timing patrol mechanism, which can regularly verify the same block stored in several transaction related nodes (actually their cloud virtual nodes). Once any forgery or tampering is found, the block rebuilding mechanism is started to realize block synchronization.

In the next step, we plan to further improve the following technical methods so as to improve the security of transaction nodes, the whole system and blocks: (1) decentralized identity based on zero knowledge proofs; (2) secure multiparty computing for privacy enhanced; (3) Layer 2 solutions based on sidechains.

5. Experimental Evaluation

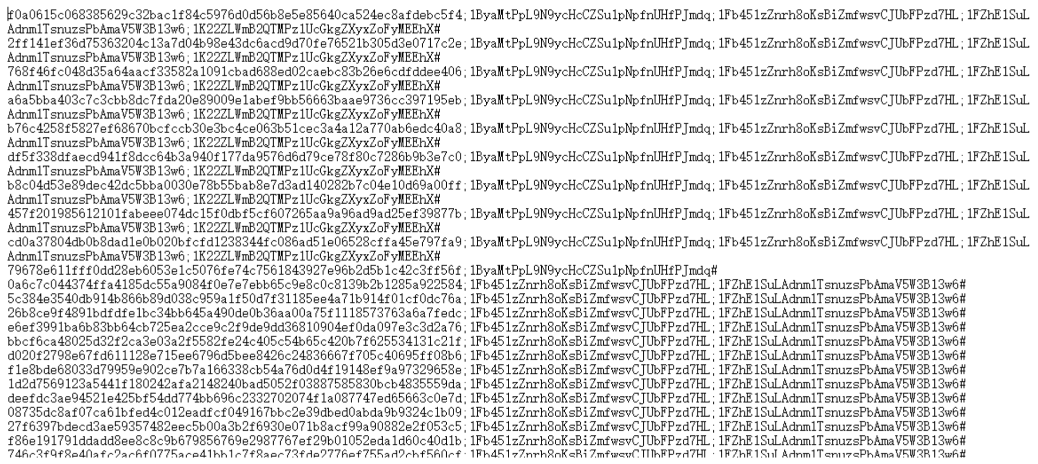
5.1. Experimental Environment and Data Preparation

In our experiment, we use a PC with Intel Core i5 Quad Core CPU, 500 GB HDD, and 8 GB memory. The operating system is Windows 8.1, the Redis version is 3.2.8, and the Java JDK version is

1.8.0_131. We use Jedis, the client implementation of the Java version of Redis, to implement operations such as retrieval queries on Redis.

Since the MTMCB system has not been put into practical use for the time being, this experiment ignores steps such as consensus verification, and successively generates 10, 100, 1000, 5000 pieces of data into the chain to realize the extended storage processing of the block while entering the chain.

The file information related to the method in this paper is shown in Figure 13. In this experiment, for the above simulation data and file content, the effects of the original sequential retrieval, the fast retrieval supported by block name file and the fast retrieval supported by Redis cache are compared.



```
f0a0615c068385629c32bac1f84c5976d0d56b8e5e85640ca524ec8afdebc5f4；1ByaMtPpL9N9ycHcCZSu1pNpfnUHfPJmdq；1Fb451zZnrh8oKsBiZmfwsvCJUbFPzd7HL；1FZhE1SuL
Adnm1TsnuzsPbAmaV5W3B13w6；1K22ZLWmB2QTMPz1UcGkgZXyxZoFyMEEhX#
2ff141ef36d75363204c13a7d04b98e43dc6acd9d70fe76521b305d3e0717c2e；1ByaMtPpL9N9ycHcCZSu1pNpfnUHfPJmdq；1Fb451zZnrh8oKsBiZmfwsvCJUbFPzd7HL；1FZhE1SuL
Adnm1TsnuzsPbAmaV5W3B13w6；1K22ZLWmB2QTMPz1UcGkgZXyxZoFyMEEhX#
768f46fc048d35a64aacf33582a1091cbad688ed02caebc83b26e6cdfddee406；1ByaMtPpL9N9ycHcCZSu1pNpfnUHfPJmdq；1Fb451zZnrh8oKsBiZmfwsvCJUbFPzd7HL；1FZhE1SuL
Adnm1TsnuzsPbAmaV5W3B13w6；1K22ZLWmB2QTMPz1UcGkgZXyxZoFyMEEhX#
a6a5bba403c7c3cbb8dc7fda20e89009e1abef9bb56663baae9736cc397195eb；1ByaMtPpL9N9ycHcCZSu1pNpfnUHfPJmdq；1Fb451zZnrh8oKsBiZmfwsvCJUbFPzd7HL；1FZhE1SuL
Adnm1TsnuzsPbAmaV5W3B13w6；1K22ZLWmB2QTMPz1UcGkgZXyxZoFyMEEhX#
b76c4258f5827ef68670bcfccb30e3bc4ce063b51cec3a4a12a770ab6edc40a8；1ByaMtPpL9N9ycHcCZSu1pNpfnUHfPJmdq；1Fb451zZnrh8oKsBiZmfwsvCJUbFPzd7HL；1FZhE1SuL
Adnm1TsnuzsPbAmaV5W3B13w6；1K22ZLWmB2QTMPz1UcGkgZXyxZoFyMEEhX#
df5f338dfaecd941f8dcc64b3a940f177da9576d6d79ce78f80c7286b9b3e7c0；1ByaMtPpL9N9ycHcCZSu1pNpfnUHfPJmdq；1Fb451zZnrh8oKsBiZmfwsvCJUbFPzd7HL；1FZhE1SuL
Adnm1TsnuzsPbAmaV5W3B13w6；1K22ZLWmB2QTMPz1UcGkgZXyxZoFyMEEhX#
b8c04d53e89dec42dc5bba0030e78b55bab8e7d3ad140282b7c04e10d69a00ff；1ByaMtPpL9N9ycHcCZSu1pNpfnUHfPJmdq；1Fb451zZnrh8oKsBiZmfwsvCJUbFPzd7HL；1FZhE1SuL
Adnm1TsnuzsPbAmaV5W3B13w6；1K22ZLWmB2QTMPz1UcGkgZXyxZoFyMEEhX#
457f201985612101fabeee074dc15f0dbf5cf607265aa9a96ad9ad25ef39877b；1ByaMtPpL9N9ycHcCZSu1pNpfnUHfPJmdq；1Fb451zZnrh8oKsBiZmfwsvCJUbFPzd7HL；1FZhE1SuL
Adnm1TsnuzsPbAmaV5W3B13w6；1K22ZLWmB2QTMPz1UcGkgZXyxZoFyMEEhX#
cd0a37804db0b8dad1e0b020bfcfd1238344fc086ad51e06528cffa45e797fa9；1ByaMtPpL9N9ycHcCZSu1pNpfnUHfPJmdq；1Fb451zZnrh8oKsBiZmfwsvCJUbFPzd7HL；1FZhE1SuL
Adnm1TsnuzsPbAmaV5W3B13w6；1K22ZLWmB2QTMPz1UcGkgZXyxZoFyMEEhX#
79678e611fff0dd28eb6053e1c5076fe74c7561843927e96b2d5b1c42c3ff56f；1ByaMtPpL9N9ycHcCZSu1pNpfnUHfPJmdq#
0a6c7c044374ffa4185dc55a9084f0e7e7ebb65c9e8c0c8139b2b1285a922584；1Fb451zZnrh8oKsBiZmfwsvCJUbFPzd7HL；1FZhE1SuLAdnm1TsnuzsPbAmaV5W3B13w6#
5c384e3540db914b866b89d038c959a1f50d7f31185ee4a71b914f01cf0dc76a；1Fb451zZnrh8oKsBiZmfwsvCJUbFPzd7HL；1FZhE1SuLAdnm1TsnuzsPbAmaV5W3B13w6#
26b8ce9f4891bdfdfe1bc34bb645a490de0b36aa00a75f1118573763a6a7fedc；1Fb451zZnrh8oKsBiZmfwsvCJUbFPzd7HL；1FZhE1SuLAdnm1TsnuzsPbAmaV5W3B13w6#
e6ef3991ba6b83bb64cb725ea2cce9c2f9de9dd36810904ef0da097e3c3d2a76；1Fb451zZnrh8oKsBiZmfwsvCJUbFPzd7HL；1FZhE1SuLAdnm1TsnuzsPbAmaV5W3B13w6#
bbcf6ca48025d32f2ca3e03a2f5582fe24c405c54b65c420b7f625534131c21f；1Fb451zZnrh8oKsBiZmfwsvCJUbFPzd7HL；1FZhE1SuLAdnm1TsnuzsPbAmaV5W3B13w6#
d020f2798e67fd611128e715ee6796d5bee8426c24836667f705c40695ff08b6；1Fb451zZnrh8oKsBiZmfwsvCJUbFPzd7HL；1FZhE1SuLAdnm1TsnuzsPbAmaV5W3B13w6#
f1e8bde68033d79959e902ce7b7a166338cb54a76d0d4f19148ef9a97329658e；1Fb451zZnrh8oKsBiZmfwsvCJUbFPzd7HL；1FZhE1SuLAdnm1TsnuzsPbAmaV5W3B13w6#
1d2d7569123a5441f180242afa2148240bad5052f03887585830bcb4835559da；1Fb451zZnrh8oKsBiZmfwsvCJUbFPzd7HL；1FZhE1SuLAdnm1TsnuzsPbAmaV5W3B13w6#
deefdc3ae94521e425bf54dd774bb696c2332702074f1a087747ed65663c0e7d；1Fb451zZnrh8oKsBiZmfwsvCJUbFPzd7HL；1FZhE1SuLAdnm1TsnuzsPbAmaV5W3B13w6#
08735dc8af07ca61bfed4c012eadfcf049167bbc2e39dbed0abda9b9324c1b09；1Fb451zZnrh8oKsBiZmfwsvCJUbFPzd7HL；1FZhE1SuLAdnm1TsnuzsPbAmaV5W3B13w6#
27f6397bdecd3ae59357482eec5b00a3b2f6930e071b8acf99a90882e2f053c5；1Fb451zZnrh8oKsBiZmfwsvCJUbFPzd7HL；1FZhE1SuLAdnm1TsnuzsPbAmaV5W3B13w6#
f86e191791ddadd8ee8c8c9b679856769e2987767ef29b01052eda1d60c40d1b；1Fb451zZnrh8oKsBiZmfwsvCJUbFPzd7HL；1FZhE1SuLAdnm1TsnuzsPbAmaV5W3B13w6#
746c3f9f8e40afc2ac6f0775ace41bb1c7f8aec73fde2776ef755ad2cbf560cf；1Fb451zZnrh8oKsBiZmfwsvCJUbFPzd7HL；1FZhE1SuLAdnm1TsnuzsPbAmaV5W3B13w6#
```

Figure 13. The schematic diagram of "Block Name File" extended by RNS.

5.2. Experiment and Result Analysis

5.2.1. Block Retrieval Performance Test and Analysis

In order to test the retrieval efficiency of the two proposed retrieval methods named RBR (Application of Redis in Block Retrieval) and BRBR (Application of B+-tree and Redis in Improved Block Retrieval), we choose the traditional block sequential retrieval method (BR) as a reference comparison, including the amount of data is equal in all scenarios, so we can directly compare the retrieval efficiency of the algorithms. In the experiment, the B+-tree is 8-order, and the transaction user is randomly selected from the user file.

As can be seen in Table 3, the time consumption of the three block retrieval algorithms depends on the number of users and the number of block files. We compare the retrieval time of various block retrieval files under different user scales and block file sizes.

For two transaction users and of 500–50,000 block files, the retrieval performance is evaluated. The experimental range of block number is initially 500. In view of the adaptability of the algorithm under the larger block scale, the range is enlarged to 1000 (block size > 4000) and 5000 (block size >10,000), as shown in Table 3.

It can be seen that the RBR and BRBR algorithms are significantly better than the traditional BR algorithm in terms of retrieval time. Under larger data scales (block file size > 10,000), the time performance of the BR algorithm decreases rapidly compared with that of the RBR and BRBR methods which are not significantly affected.

In order to reflect the difference between the RBR and the BRBR algorithm more clearly, Figure 14 focuses the experimental results of the two algorithms. The experiment shows that when the block size is less than 10,000, the retrieval time growth trend of the two is about the same, but greater than 10,000. Afterwards, the time increase trend of the RBR algorithm is faster than the BRBR algorithm.

Table 3. Time comparison table of three retrieval algorithms under different block file sizes (unit: ms).

Number of Blocks	BR Algorithm	RBR Algorithm	BRBR Algorithm
500	78	78	61
1000	81	81	69
1500	567	86	72
2000	672	91	74
2500	793	98	77
3000	900	110	82
3500	976	124	85
4000	1056	130	89
5000	1152	145	94
6000	1189	149	106
7000	1376	156	110
8000	1422	167	116
9000	1646	171	120
10,000	1988	178	129
15,000	3033	213	143
20,000	4312	245	164
25,000	4878	267	173
30,000	5792	292	181
35,000	7316	342	198
40,000	8444	389	209
45,000	10,451	422	217
50,000	12,976	493	234

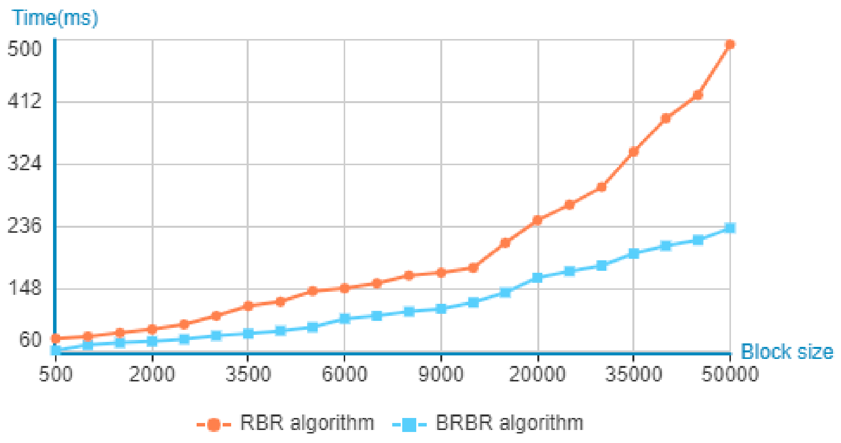


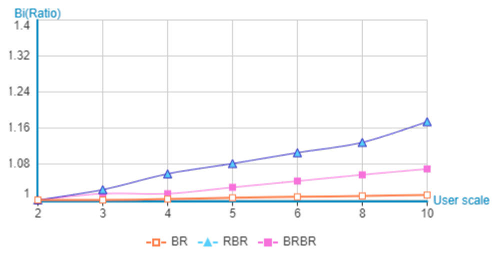
Figure 14. Comparison of retrieval efficiency of RBR and BRBR algorithms under different block sizes (number of transaction users = 2).

Considering that different numbers of user sizes will also have an impact on the retrieval efficiency, we now select block sizes 1500, 3000, 5000, and 10,000 and retrieval user sizes are from 2, 3, 4, 5, 6, 8, and 10 (randomly selected combinations from user files).

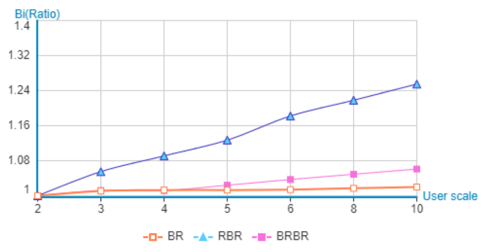
Since the RBR and BRBR algorithm already are very fast, in order to better compare the growth trend of the algorithm in retrieval time, the small absolute increases here turn into relative proportion. We compute and compare the metric as $B_i = t_i/t_2$ $(i = 2, .., 10)$, where i is the number of users participating in the transaction, t_2 is the retrieval time when the number of transaction users is 2, and Bi is the retrieval time when the number of transaction users = i. The results of data processing and comparison are shown in Table 4 and Figure 15.

Table 4. Comparison of retrieval time under different user scale (unit: ms).

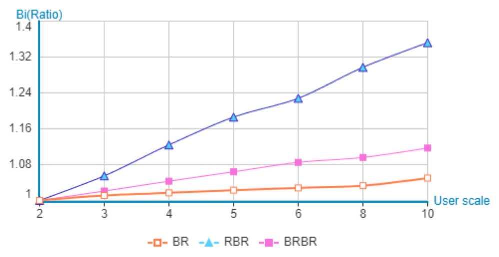
	User Scale *i*	**BR**		**RBR**		**BRBR**	
		Time	***Bi***	**Time**	***Bi***	**Time**	***Bi***
Block Size = 1500	2	567	1.000	86	1.000	72	1.000
	3	567	1.000	88	1.023	73	1.014
	4	568	1.002	91	1.058	73	1.014
	5	570	1.005	93	1.081	74	1.028
	6	571	1.007	95	1.105	75	1.042
	8	572	1.009	97	1.128	76	1.056
	10	573	1.011	101	1.174	77	1.069
	User scale *i*	**BR Time**	***Bi***	**RBR Time**	***Bi***	**BRBR Time**	***Bi***
Block size = 3000	2	900	1.000	110	1.000	82	1.000
	3	910	1.011	116	1.055	83	1.012
	4	912	1.013	120	1.091	83	1.012
	5	912	1.013	124	1.127	84	1.024
	6	913	1.014	130	1.182	85	1.037
	8	915	1.017	134	1.218	86	1.049
	10	918	1.020	138	1.255	87	1.061
	User scale *i*	**BR Time**	***Bi***	**RBR Time**	***Bi***	**BRBR Time**	***Bi***
Block size =5000	2	1152	1.000	145	1.000	94	1.000
	3	1165	1.011	153	1.055	96	1.021
	4	1172	1.017	163	1.124	98	1.043
	5	1179	1.023	172	1.186	100	1.064
	6	1184	1.028	178	1.228	102	1.085
	8	1190	1.033	188	1.297	103	1.096
	10	1210	1.050	196	1.352	105	1.117
	User scale *i*	**BR Time**	***Bi***	**RBR Time**	***Bi***	**BRBR Time**	***Bi***
Block size =10,000	2	1988	1.000	178	1.000	129	1.000
	3	2012	1.012	189	1.062	132	1.023
	4	2045	1.029	201	1.129	135	1.047
	5	2078	1.045	212	1.191	139	1.078
	6	2100	1.056	224	1.258	142	1.101
	8	2134	1.073	233	1.309	146	1.132
	10	2153	1.083	250	1.404	148	1.147



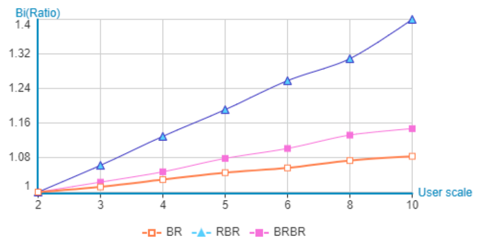
(**a**) $M = 1500$



(**b**) $M = 3000$



(**c**) $M = 5000$



(**d**) $M = 10000$

Figure 15. The increasing trend of retrieval time of three retrieval algorithms in different user scale and block number (M is the module of block gauge).

As can be seen from Figure 15, the traditional retrieval BR algorithm is the least affected by the number of transaction users, and the RBR algorithm is the most affected. In different block numbers, when the number of users is less than 3, the three retrieval algorithms are not affected by the number of blocks. However, when the number of users is more than 3, the RBR method is affected significantly, which shows that the complexity of calculation will increase with the number of users in the RBR algorithm.

5.2.2. Stability Analysis of Block Retrieval

The stability of retrieval refers to the relatively consistent retrieval efficiency for the same type of retrieval requirements. We investigate the retrieval stability of three retrieval algorithms. We set the number of transaction users to 2 and the size of block file to 5000. We randomly generate 20 groups of transaction user combinations and respectively find the first block satisfying the conditions in the three retrieval algorithms, as shown in Table 5 and Figure 16.

Table 5. First response time comparison of three retrieval algorithms (unit: ms).

Serial Number	BR	RBR	BRBR	Serial Number	BR	RBR	BRBR
1	972	45	85	2	577	167	87
3	888	114	86	4	1045	123	85
5	987	55	87	6	1142	54	84
7	1065	153	86	8	544	156	88
9	898	132	85	10	1033	98	84
11	873	123	83	12	1029	87	85
13	1078	107	86	14	922	138	87
15	1033	56	87	16	682	135	84
17	962	97	89	18	712	142	83
19	1057	138	85	20	572	75	85

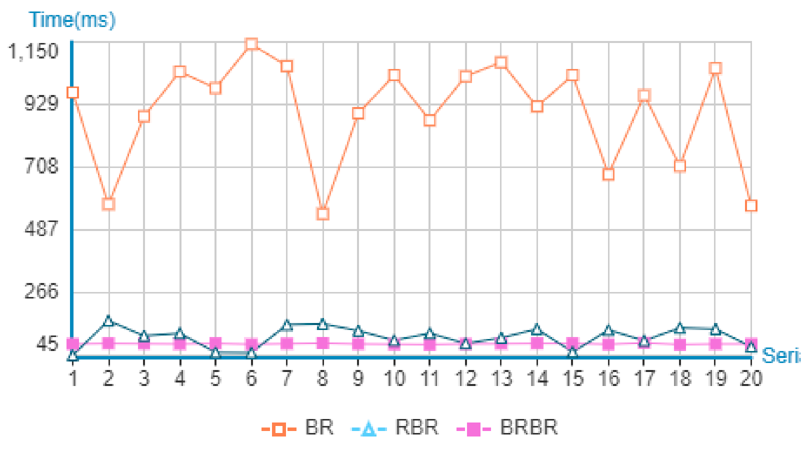


Figure 16. First response time of three retrieval algorithms in block retrieval.

The experimental results show that both BR and RBR algorithms fluctuate in different degrees under the demand of the first search. The retrieval performance is not only affected by the algorithm, but also affected by the location of block information storage. As the BRBR method uses a B+-tree index, the number of searches is not affected by the location of block storage and instead only determined by the number of blocks. It thus has a stable retrieval performance. It seems to be equivalent to the time required to find all blocks.

According to the above data analysis, both RBR and its improved variant BRBR achieve greater performance compared with the traditional retrieval method. BRBR exhibits better retrieval efficiency and stability than RBR, and with increasing number of blocks, this efficiency gap is also growing.

6. Conclusions

The retrieval time is one of the factors limiting applications of blockchains. Traditional blockchain retrieval needs to traverse the whole chain, no matter whether the block is near the end or the head of the chain. The retrieval time is always too long to meet application requirements. For this reason, this paper proposes two retrieval methods named RBR (Application of Redis in Block Retrieval) and BRBR (Application of B+-tree and Redis in Improved Block Retrieval) for the MTMCB. Without changing the original structure of the blockchain, RBR merely adds a "User Set" and a "Block Name Set" on the RNS. These two sets establish an index of block names, users, and block files, which can realize the rapid retrieval of blocks. RBR can meet the requirements of general consortium blockchain application system normally, but for the application system with large-scale users and block files, RBR cannot have a satisfactory retrieval speed. Therefore, this paper presents a complex structure of B+-tree and block name inverted, and proposes a corresponding block retrieval algorithm named BRBR. Firstly, the block information is put into the orderly set according to the transaction participants, and the B+-tree index is established for the transaction participants. The block information is composed into ordered sets according to the transaction participants. Then, the B+-tree index is established for the transaction participants. When searching, the B+-tree index allows us to quickly find the ordered set, including the participants and block name. At last, we can get the information of the relevant block file quickly efficiently by finding the intersection of several related ordered sets. We analyze the time requirements of the retrieval algorithms in a comprehensive experiment. The efficiency and stability of the improved algorithm are verified by experiments.

This solution is currently only applied to MTMCB and may not be applicable to other blockchain systems. In the future, we will study how to apply the ideas in this article to private blockchain systems.

Author Contributions: Conceptualization, J.T. and J.Z.; methodology, J.T. and S.C.; software, J.T.; validation, J.T. and L.Z.; formal analysis, J.Z.; investigation, J.T.; resources, T.W.; data curation, J.T. and L.Z.; writing—original draft preparation, J.Z.; writing—review and editing, J.T.; visualization, S.C.; supervision, J.T.; project administration, J.T. and L.Z. All authors have read and agreed to the published version of the manuscript.

Funding: The authors would like to express their thanks to the referees for their valuable suggestions. This work was supported by the grant from the National Natural Science Foundation of China, Nos. 61672204, 61673359; in part by the Key Scientific Research Foundation of Education Department of Anhui Province, Nos.KJ2018A0555, KJ2019A0833; the Natural Science Foundation of Anhui Provincial, Nos.1908085MF184; in part by the Key Technologies R&D Program of Anhui Province, No.1804a09020058; in part by the Major Science and Technology Project of Anhui Province, No. 17030901026; and in part by the Key Constructive Discipline Project of Hefei University, No. 2016xk05.

Acknowledgments: Weise acknowledges support from the National Natural Science Foundation of China under Grant 61673359 and the Hefei Specially Recruited Foreign Expert support.

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

References

1. Raju, S.; Rajesh, V.; Deogun, J.S. The Case for a Data Bank: An Institution to Govern Healthcare and Education. In Proceedings of the 10th International Conference on Theory and Practice of Electronic Governance, New Delhi, India, 7–9 March 2017.
2. Vo, H.T.; Mehedy, L.; Mohania, M.; Abebe, E. Blockchain-based Data Management and Analytics for Micro-insurance Applications. In Proceedings of the 2017 ACM on Conference on Information and Knowledge Management (CIKM '17), New York, NY, USA, 6–10 November 2017.
3. Zhao, W.; Lv, J.; Yao, X.; Zhao, J.; Jin, Z.; Qiang, Y.; Che, Z.; Wei, C. Consortium Blockchain-Based Microgrid Market Transaction Research. *Energies* **2019**, *12*, 3812. [CrossRef]
4. Zhang, J. A Multi-Transaction Mode Consortium Blockchain. *Int. J. Perform. Eng.* **2018**, *14*, 765–784. [CrossRef]
5. Castillo, M. Navigating Bitcoin, Ethereum, XRP: How Google Is Quietly Making Blockchains Searchable. Available online: https://www.forbes.com/sites/michaeldelcastillo/2019/02/04/navigating-bitcoin-ethereum-xrp-how-google-is-quietly-making-blockchains-searchable/#6bb7d774248f (accessed on 12 January 2020).

6. Wang, H. Medical Information Security Storage and Network Sharing Based on Blockchain Technology. Available online: https://kns-cnki-net/kcms/detail/50.1075.TP.20190816.1701.059.html (accessed on 12 January 2020).
7. 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]
8. Shibata, N. Proof-of-Search: Combining Blockchain Consensus Formation with Solving Optimization Problems. *IEEE Access* **2019**, *7*, 172994–173006. [CrossRef]
9. Lv, J. Research on Log Security Storage and Retrieval Based on Combination of Chains and Chains. Available online: http://kns.cnki.net/kcms/detail/50.1075.TP.20191122.1551.016.html (accessed on 12 January 2020).
10. Zhou, E.; Sun, H.; Pi, B. Ledgerdata Refiner: A Powerful Ledger Data Query Platform for Hyperledger Fabric. In Proceedings of the 2019 6th IEEE International Conference on Internet of Things: Systems, Management and Security (IOTSMS), Granada, Spain, 22–25 October 2019; pp. 433–440.
11. Do, H.G.; Ng, W.K. Blockchain-based System for Secure Data Storage with Private Keyword Search. In Proceedings of the 2017 IEEE World Congress on Services (SERVICES), Honolulu, HI, USA, 25–30 June 2017; pp. 90–93.
12. Liu, W. Method for Distributing and Retrieving Data on a Blockchain Network with Peer Nodes. China Patent No. CN201810774852.4, 7 July 2018.
13. Hou, D. A Search Method and System for Business Information of Blockchain. China Patent No. CN201711192111.7, 24 November 2017.
14. Wang, X. A Method of Personalized Privacy Information Retrieval Based on Blockchain. China Patent No. CN201710606170.8, 24 July 2017.
15. Devika, K.N.; Bhakthavatchalu, R. Parameterizable FPGA Implementation of SHA-256 using Blockchain Concept. In Proceedings of the 2019 IEEE International Conference on Communication and Signal Processing (ICCSP), Chennai, India, 4–6 April 2019; pp. 0370–0374.
16. Redis. Available online: https://redis.io/community (accessed on 16 January 2020).
17. El, A.A.; Bahaj, M.; Khourdifi, Y. Supply of a Key Value Database Redis In-Memory by Data from a Relational Database. In Proceedings of the 19th IEEE Mediterranean Electrotechnical Conference (MELECON), Marrakech, Morocco, 2–7 May 2018; pp. 46–51.
18. Chen, S.; Tang, X.; Wang, H. Towards Scalable and Reliable In-memory Storage System: A case Study with Redis. In Proceedings of the 2016 IEEE Trustcom/BigDataSE/ISPA, Tianjin, China, 23–26 August 2016; pp. 1660–1667.
19. Seo, K.; Ahn, J.; Im, D.-H. Optimization of Shortest-Path Search on RDBMS-Based Graphs. *ISPRS Int. J. Geo-Inf.* **2019**, *8*, 550. [CrossRef]
20. Graefe, G. B-tree Indexes, Interpolation Search, and Skew. In Proceedings of the 2nd International Workshop on Data Management on New Hardware, Chicago, IL, USA, 27–29 June 2006.
21. Park, D.J.; Choi, H.G. Effect of Node Size on the Performance of the B+-tree on Flash Memory. *KIPS Trans. Part A* **2008**, *15*, 325–334. [CrossRef]
22. Chen, S.; Gibbons, P.B.; Mowry, T.C. Fractal Prefetching B+ -Trees: Optimizing both Cache and Disk Performance. In Proceedings of the ACM Sigmod International Conference on Management of Data, Madison, WI, USA, 3–6 June 2002.
23. Diogo, M.; Cabral, B.; Bernardino, J. Consistency Models of NoSQL Databases. *Future Internet* **2019**, *11*, 43. [CrossRef]
24. Min-Allah, N. Effect of Ordered Set on Feasibility Analysis of Static-priority System. *J. Supercomput.* **2019**, *75*, 475–487. [CrossRef]