An efficient data structure for distributed ledger in blockchain systems

Tzu-Lun Huang
Department of Computer Science and
Information Engineering
St. John University
New Taipei City, Taiwan, R.O.C.
tlhuang@mail.sju.edu.tw

Jason Huang
Advanced Financial Engineering LLC
Palo Alto, CA 94301, USA
jhuang1223@yahoo.com

***Abstract*—The storage structure on the data block of the current blockchain systems is still a linked list. This traditional structure is one-way and is not efficient for query operation. In view of this, we proposed a novel searching structure based on a height balanced Binary Search Tree (BST). The data structure we proposed not only retains the characteristics of the traditional ledgers but also adds the function of quick query. Through the new data structure, we can quickly find the starting position of the search and start searching for all transaction records within a specified range of time from this position. We also made an analysis and comparisons in the final.**

Keywords—blockchain, AVL tree, threaded, binary search tree

I. Introduction and Related Work

Storage and Consensus are the two most important components that make up the blockchain system. The blockchain can be regarded as a distributed database system. One of the main differences between blockchain systems and the general distributed database systems is on the way of storage. The way of storage in the general distributed database systems is to break up the complete information and distribute it evenly in different storage devices. In this case, a mapping function must be created for finding out the corresponding storage device where each piece of information is stored. However, the blockchain is completely different. The blockchain is a replicated data storage, that is, the same data will be copied to all storage devices. The reason why the blockchain is designed in this way is the consideration of security. If someone want to tamper with the data in the blockchain, he or she must modify a lot of copies of data instead of just only one data. Such an approach can greatly increase the difficulty of the tampering and relatively increase the integrity and correctness of the data.

Another different with the traditional database systems (centralized and distributed) is the location where the data is stored. Data in the traditional databases is stored in the external storage devices, e.g., disk or magnetic tape. The data structure designed for these databases is generally a multiway searching tree, e.g., B tree or B+ tree. This type of search is also called external search. External search mainly reduces the number of searches by reducing the height of the searching tree. After all, external searches involve the operation of the externally mechanical devices.

However, the blockchain data is stored in the memory, and this way of data storage is also called in-memory database [2]. The data structure of the data block employed in the current blockchain is still the structure of the linearly linked list. The linked list can be regarded as a skew tree and this structure is not efficient for the operation of search or query. In this case, the data structure of the internal search is very suitable to be used in the in-memory database. So we use the data structure of the internal search as the mainly searching structure in this paper. There are four basic operations in the general database systems, they are insertion, deletion, modification, and query. But in the in-memory database like the blockchain, only insertion and query these two operations are allowed. Therefore, the blockchain is also called append-only database.

The related research is not too many, but roughly can be divided into two categories, ledger and transaction. On the structure of transactions, Merkle-Patricia Tree [3, 4] is currently and widely used as a data structure for storing transaction data. The data structure of the Merkle-Patricia Tree is a complete BST and all hashes of transaction data is placed on level of the leaves. Two transaction hashes are paired and hashed, each non-leaf node is labeled with a hash code of its two children. This procedure is repeated until a single hash remains, that is, the root of the Merkle-Patricia tree. On the respect of the ledger, the paper [2] described three characteristics of the distributed ledger, they are data model, number of ledgers, and the ownership of the ledger. The paper [5] presented a data structure of the ledger based on Directed Acyclic Graph (DAG). The protocol SPECTRE collects the relevant data blocks and establishes a directed and acyclic order relationship among blocks. The purpose of block DAG is for the consideration of the security instead of the efficiency for searching or querying data. The paper [6] proposed a framework of the network coded storage for distributed ledgers in the blockchain. Analysis shows a significant improvement on the respect of saving the storage room. The paper [7] presented an architecture of virtualization, i.e., vDLT, for distributed ledgers. By the approach of vDLT, the management and configuration of the system can be simplified and hence also increase the scalability of the blockchain.

All current public blockchain systems, such as Bitcoin and Ethereum, use batch processing to deal with the transaction data. Each transaction request will not be processed immediately but have to wait for a period of time or collect a sufficient amount of transactions. The waiting time on average for Bitcoin is about 10 minutes [8] but one hour is required if the security of the payment is considered [9]. The same is true for Ethercoin, 15 ~ 30 seconds waiting for processing a new transaction block and a few minutes are required for the security of the payment [10]. Unlike the batch processing, the real-time processing is to process the request immediately once the request arrives at the blockchain system. For some applications that required the real-time processing, e.g., credit card or instant payment, to fetch data from ledgers must be more efficient, especially for those operations about searching or querying the data. These real-time applications have a common characteristic on the distributed ledger, that is, the number of data blocks in the ledger will be much more than the Bitcoin and the number of transaction operations in each data block is much less than the number in the Bitcoin relatively.

Based on the characteristic of the data storage described above, the direction of our design will be toward to the real-time processing. That is, there are much more data blocks in the blockchain list but less or much less number of transaction operations in each data block. In this case, we proposed a data structure for efficiently searching and querying the data and this is also the major contribution of our paper. To best our knows, there is currently no paper discussing how to improve the efficiency of the data access on the blockchain storage and our paper is the first one.

II. Concept, Structure, Algorithm, and Analysis

In this section, we present the concept, assumption, and the data structure of our proposed mechanism. Analysis and comparisons are described in the final subsection.

A. *Concept and Assumption*

As shown in Fig. 1, the data structure of our distributed ledger is a doubly linked list. The transaction records stored in the ledger can be searched from the both side, i.e., the head or the tail. We implement the idea by combining the concept of Threaded Binary Search Tree (TBST) and AVL tree [1]. Hence, the final TBST is a height balanced tree and the searching key is the time-stamp.

We assume the value of the next one time-stamp must be greater than the previous one. Under this assumption, we can quickly search for the desired block and the corresponding transactions within the same block from the TBST. In addition, because the data structure used has bidirectional links, after finding out the specified block, we can use the two-way links to find out the relevant transaction records within a specified rang of time.

B. *Structure, Algorithm, and an Example*

The structure can be divided into two parts and are as follows.

1) *Block format*

Table 3 shows the data structure of each block. The structure are mainly composed of the nine important data fields. The identification of the block is recorded in the field of Block's ID. Two fields, BlockHash and PreBlockHash, are used for the purpose of tampering resistance. The field of Time-Stamp is the primary key and used for searching a specified data block in the binary search tree. Two links are used for the threaded links in TBST. The left link is used for indicating the predecessor of the in-order traversal in BST and the successor of the traversal is indicated in the field of the right threaded link. The left and right tree links indicate the left and right children in the BST. All transactions waiting for execution are stored in the field of "Transactions". Each transaction has its ID, time-stamp, and content. The format of each transaction is also shown in Table 3.

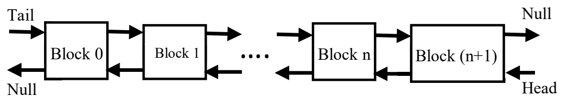

Fig. 1 Concept of the ledger made of doubly linked list

Block ID							
Block Hash							
PreBlock Hash							
Block's Time-Stamp							
Left Threaded Link	Left Tree Link	T_1	T_2	…..	T_C	Right Tree Link	Right Threaded Link

Transaction ID
Transaction Time-Stamp
Transaction Content

Table 3. Format of Blocks and Transactions

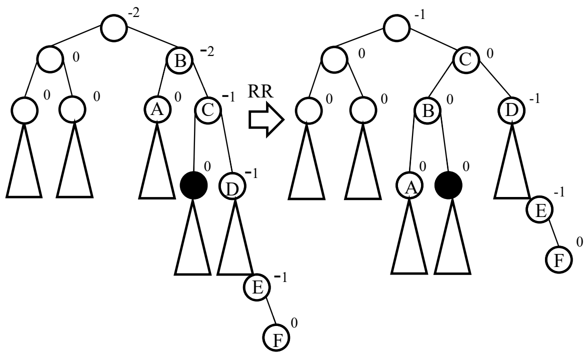

Fig. 2 The schematic of RR operation

2) *Threaded AVL tree*

The characteristics of the distributed ledger is append-only and this feature is the same as the AVL tree. The storage way of the distributed ledger and the AVL tree are also in the way of in-memory. Because the searching key of the BST is the time-stamp and we assume the value the next one must be larger than the previous one, the newly created block is always to insert to the subtree at the right hand side. In order to maintain the height balance of the BST, we use the mechanism of the adjustment used in AVL tree. Fig. 2 shows the operation of the adjustment. As the newly created block is appended to the rightmost location and cause the BST unbalanced, the RR[1] algorithm is executed for adjusting the BST such that the absolute value of the difference on the height between the right and the left subtree is equal or less than one. The triangles in the figure represents the left and the

[1] The definition of RR [1] is that the newly joining node is inserted in the rightmost position of the right subtree.

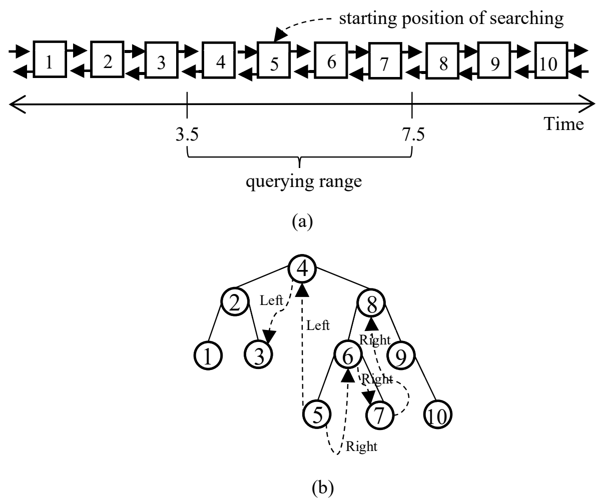

Fig. 4 A searching example starting from the block node 5

right subtrees with the same height. The number next to each node in the tree is the height difference between the left and the right subtree on the height. The negative sign represents the left subtree is shorter than the right subtree. There are four operations of adjustment in AVL tree, but only RR operation can be carried out in the blockchain list. This is because the newly added block will always be placed at the end of the linked list and this location relatively corresponds to the position of the rightmost subtree in BST.

After adjusting the BST from unbalanced to balanced state, the next step is to create links and point to the ..predecessor and the successor of the current block in the linked list. In the practical implementation, we are using the concept of TBST. The left and the right threaded links in TBST are set and point to the predecessor and the successor of the in-order traversal of the tree. Upon the adjustment and the setting the links, the final TBST is a height balanced BST.

3) An Example

We illustrate a scenario as an example in this part. In Fig. 4(a), there are currently ten data blocks in the ledger and the number in the box represents the time-stamp when the block was created. Things to do in the blockchain now is to query all transaction records in a certain range of time between the time-stamp value 3.5 and 7.5. The system will first calculate the starting position of the search and the calculated position is at the value of 5.5, i.e., (3.5+7.5)/2.

Fig. 4(b) is a completely the same representative way of our TBST relative to Fig. 4(a). After searching the TBST in Fig. 4(b) by the value 5.5 of the time-stamp, the starting position of this search falls on the tree node 5. The search starts from the tree node 5, follows the two threaded links at the left and the right hand side (i.e., the dash lines shown in Fig. 4(b)), and finds out all transaction records between 3.5 and 7.5. The output result is all the transaction records in the block 4, 5, 6, and 7.

C. Analysis and Comparisons

In this subsection, we make an analysis and a comparisons between the traditional blockchain and our distributed ledger based on the TBST.

1) Complexities

	Traditional Blockchain	Ours
Time Complexity		
Search for a Block	O(n)	$O(\log_2 n)$
Search for a Transaction	O($n + c$)	$O(\log_2 n + c)$
Search for kth adjacent Block	$\theta(k)$	$\theta(k)$
Search for a Transaction within kth adjacent Block	$\theta(k + c)$; c is a constant	$\theta(k + c)$
Space complexity	$\theta(n)$	$\theta(n)$
Maintaining Complexity		
Insert a Block	$\theta(1)$	$O(\log_2 n)$

Table 5. Comparisons between Traditional Blockchain and Ours

***Theorem* 1:** The time complexity of searching a specified block is $O(\log_2 n)$.

Proof. Because the data structure of our ledger TBST is also an AVL tree and the AVL tree is with the characteristic of height balanced, the time complexity of searching a specified block is $O(\log_2 n)$.

***Theorem* 2:** The time complexity for inserting a newly created block is $O(\log_2 n)$.

Proof. This is because the time complexity for adjusting an AVL tree from unbalanced to balanced state is $O(\log_2 n)$.

2) Comparisions

The comparisons of time, space, and maintaining complexities are shown in Table. 5. The data structure of the traditional blockchain is the structure of the linked list. Because the linked list can be regarded as a skew tree, the searching efficiency is linear complexity, that is, $O(n)$. However, the skew tree can be converted and become a height balanced BST. Hence, the searching efficiency can be improved further to $O(\log_2 n)$. The number of transactions in each block is small and fixed. We define the variable c as the maximum number of transactions stored in a block. Hence, after finding out the specified block, it still take $O(c)$ time to find the specified transaction record, the variable c is a constant value.

We can also query transaction records within a certain range of time through the left and right threaded links in each block. After spending $O(\log_2 n)$ to find out the starting block, and then search for the previous and the next k block data through these two links. This way of the search can also be executed in parallel from both the left and the right hand sides. The search for a range of k blocks takes $\theta(k)$ time and $\theta(k + c)$ for a specified transaction record in a block.
The space complexity of our ledger is the same as the traditional blockchain ledger, that is, $\theta(n)$. To insert a block takes $\theta(\log_2 n)$ in our ledger but $\theta(1)$ in the traditional ledger. This is because our ledger requites $O(\log_2 n)$ for adjusting the height of tree to be balanced.

III. Conclusion

In this paper, we proposed a data structure which is different from the one-way linked list employed in the traditional blockchain systems. The concept of our proposed data structure is a doubly linked list. But for the sake of the consideration of query efficiency, we implement the concept by using the threaded AVL tree. The resulting tree is height balanced and the desired block (i.e., the starting position of the search) can be efficiently found out in $O(\log_2 n)$ time by the key value of the time-stamp. In addition, through the two threaded links in each block, we can also quickly query the all transaction records within a certain range of time. Analysis and comparisons also show the superiority on the searching efficiency.

REFERENCES

[1] E. Horowitz, S. Sahni, and S. Anderson-Freed, “Fundamentals of Data Structures in C 2nd Edition,” Silicon Press, 2008..

[2] T. T. A. Dinh, R. Liu, M. Zhang, G. Chen, B. C. Coi, and J. Wang, “Untangling Blockchain: A data processing view of blockchain systems,” IEEE Transactions on Knowledge and Data Engineering, 2018, 30(7): 1366-1385.

[3] R. C. Merkle, “A digital signature based on conventional encryption function,” Proc. 7th Conf. Adv. Cryptol. (CRYPTO’87), 1987, 369-378,.

[4] D. R. Morrison, “Patrica: Practical algorithm to retrieve information coded in alphanumeric,” Journal of ACM, 1968, 15(4): 514-534.

[5] Y. Sompolinsky, Y. Lewenberg, and A. Zohar, “SPECTRE: A fast and scalable cryptocurrency protocol,” Int. Assoc. Cryptol. Res. Tech. Rep., 2016. Available: https//eprint.iacr.org/2016/1159..

[6] M. Dai, S. Zhang, H. Wang, and S. Jin, “A low storage room requirement framework for distributed ledger in blockchain,” IEEE Access, 2018, 6: 22970-22975,.

[7] F. R. Yu, J. Liu, Y. He, P. Si, and Y. Zhang, “Virtualization for distributed ledger technology (vDLT),” IEEE Access, 2018, 6: 25019-25028,.

[8] K. Croman, C. Decker, I. Eyal, A. E. Gencer, A. Juels, A.Miller, P. Saxena, E. G. Sirer, D. Song, and R. Wattenhofer, “On scaling decentralized blockchains,” International Conference on Financial Cryptography and Data Security, 2016, 106-125.

[9] “The Bitcoin lighting networks,” https://raiden.network/.

[10] “The Raiden Network,” https://raiden.network/.