Mathematical Biosciences and Engineering

http://www.aimspress.com/journal/MBE

MBE, 17(5): 5633–5650.
DOI: 10.3934/mbe.2020303
Received: 07 July 2020
Accepted: 10 August 2020
Published: 24 August 2020

Research article

Intelligent manufacturing security model based on improved blockchain

Jiahe Xu[1,*], Yuan Tian[2], Tinghuai Ma[3] and Najla Al-Nabhan[4]

[1] School of Automation, Southeast University, Nanjing 211189, China
[2] Nanjing Institute of Technology, Nanjing 211167, China
[3] Nanjing University of Information Science & Technology, Nanjing 210044, China
[4] Department of Computer Science, King Saud University, Saudi Arabia

* **Correspondence:** Email: 213170687@seu.edu.cn.

Abstract: The Industrial Internet of Things (IIoT) plays an important role in the development of smart factories. However, the existing IIoT systems are prone to suffering from single points of failure and unable to provide stable service. Meanwhile, with the increase of node scale and network quantity, the maintenance cost presents to be higher. Such a disadvantage can be effectively compensated by the features such as security, privacy, non-tamperability and distributed deployment supported by the blockchain. In this paper, first, an intelligent manufacturing security model based on blockchain was proposed. Due to the high power consumption and low throughput of the traditional blockchain, IoT devices with limited power consumption can not work independently. Therefore, in this paper, a new Merkle Patricia tree (MPT) was adopted to extend the blockchain structure and provide fast query of node status. Second, since the MPT does not support concurrent operation and the data operation performance deteriorates with high data volume, a lock-free concurrent and cache-based Merkle Patricia tree was proposed (CMPT) to support lock-free concurrent data operation, which can improve the data operation efficiency in multi-core system. The experimental results indicate that, compared with the original MPT, the CMPT proposed in this paper effectively reduced the time complexity of data insertion and data query and improved the speed of block construction and data query.

Keywords: smart manufacturing; Industrial Internet of Things; blockchain; Merkle Patricia tree; concurrent data structure

1. Introduction

Under the background of industry 4.0, through the paradigm shift from the existing centralized manufacturing industry to the distributed manufacturing industry, the smart factory can handle complex

environmental changes with more efficient production [1]. The core of such a shift is to integrate the standards and technologies of the Internet of things (IoT) into the industrial process [2], that is, the Industrial Internet of Things (IIoT). However, for smart factories, with the increase in network connecting devices, the fancy price as well as the maintenance fee of big servers has brought about high cost. At the same time, a large number of connecting devices have not only increased the risk of privacy leak (caused by network attack) but also incurred the problem on economy and security. Consequently, the security, privacy and fault tolerance of current IIoT architecture are the problems which should be resolved immediately [3,4].

With the emergence of blockchain technology and the expansion of its application field, the above problems caused by the traditional central entity mode of smart factory can be effectively solved by blockchian [4,5]. In other words, the blockchain has incorporated the security of cryptography algorithm, the non-tamperability based on hash chain and the consistency guaranteed by the consensus algorithm. However, due to the large number of IIoT devices and the limited computing ability as well as power limitation, the traditional blockchain has faced the challenges such as low security, limited throughput, slow transaction, and limited storage resources. Consequently, at the current stage, blockchain can not meet the requirements of the Industrial Internet of Things, or IIoT.

The traditional blockchain based on bitcoin system has following drawbacks:

1) The transaction is open and completely transparent, and due to the limitations of consensus algorithm, the transaction delay is severe. For example, the waiting time of bitcoin transaction follows "Poisson" distribution, with an average of 10 minutes [6], which is under the real-time requirement of the industrial Internet.

2) Decentralized design. In fact, all nodes can join or leave the blockchain network at any time. IIoT system usually needs to manage a large number of transactions per second. Due to the system redundancy caused by decentralization, bitcoin and Ethereum can only process three to four or dozens of transactions per second, which is far to the high transaction throughput requirements of IIoT [7,8]. Performance and scalability are the main drawbacks to be enhanced, in the integration of IIoT system and blockchain [9].

The essence of transactions is a set of UTXO (unused transaction output), which is output by previous transactions and used for future transactions. The balance of users is the sum of the face value of UTXO that they owned. UTXO is stateless, therefore, except the virtual currency, it is not suitable for more complex applications. In order to improve the overall efficiency and availability of blockchain, Ethereum has proposed an account based data model, which can manage transactions better than UTXO. The Ethereum has also designed three index trees (the Merkle-Patricia tree) to speed up the query [10]. More specifically, Patricia tree can quickly locate query results; Merkle tree is used to ensure that the data received from others is not damaged or replaced; finally, the account model, using the MPT tree as the index, can support efficient query of transactions to facilitate the construction of complex applications [11].

In this paper, we have used the MPT tree to expand traditional blockchain so that, via limited hash computing, it is able to rapidly locate the manufacturing equipment or products from intelligent manufacturing system. Such an improvement can be applied to intelligent manufacturing management or product sourcing. The requester can get the latest data of the system through the MPT tree in the latest block, which reduces the time complexity and storage requirements of the query.

However, the insertion performance of the Merkle-Patricia tree is limited. Each insertion needs to traverse and calculate the hash tree from its root. And, due to the insufficient scalability, the insertion can not be conducted concurrently, which affects the speed of block construction and query. Focusing on above drawbacks, in this paper, we additionally improve the traditional blockchain by CMPT tree. More specifically, based on the atomic and lock-free operation "CAS" (compare-and-swap), the CMPT tree can support concurrent caching. Compared to traditional MPT-based blockchian, the CMPT-based one can have higher efficiency of insertion and query, as well as the better performance of the whole intelligent manufacturing system. In addition, via the CMPT-based blockchian, we proposed an improved intelligent manufacturing security model which has higher throughput and is able to meet the real-time requirement of the Industrial Internet of Things (IIoT).

In all, the innovation of this paper can be concluded as: In order to overcome the single-point-failure drawback of IIoT system, we turn to enhance the security of the whole IIoT system. Therefore, an intelligent manufacturing security model based on blockchain was proposed. However, the blockchain has higher power consumption which could further deteriorate the single-point-failure. Consequently, the Concurrent Merkle-Patricia tree (CMPT) is adopted to extend the ordinary blockchain. In this way, the security model could be implemented in a power-conserved way.

2. Related works

Intelligent manufacturing can use blockchain to solve many problems in terms of the deployment process. For example, it can help the intelligent factory to conduct horizontal or vertical integration by providing public trust data. And, multiple intelligent factories can also conduct transactions and secure data exchange through the nodes in blockchain nodes [4].

In [12], an IIoT platform based on blockchain and smart contract technology, named BPIIoT, has been proposed. For BPIIoT, transactions on the blockchain is realized by smart contract. The storage and processing of data are carried out by the off chain network. In this way, the distributed blockchain can reduce the storage and computing costs, and the operating costs of the IoT. However, the BPIIoT is still based on traditional public blocktrain, in other words, fully distributed network is not suitable for the scenario of industrial Internet. In [13], an IIoT architecture with high privacy and security based on private chain has been proposed. Such an architecture divides the system into internal and external layers and the private chain is only suitable for single smart factory environment, not for the multi-industry and multi-intelligent factory interaction scenarios. In [14], the author has proposed an optimized IIoT framework which dynamically adjusts the production nodes, consensus algorithm, block size and block interval by deep learning. However, the optimization only focuses on configuration, without any improvement on the internal algorithm of blockchian. In [15], the author has proposed a particular IoT system for blockchain based on secure multiparty computing protocol. In this way, the server can calculate the encrypted data without decrypting the data, reducing the risk of data leakage.

Patricia tree is a dictionary tree and when given an ordinary amount of data, it only has low height. Therefore, it is suitable for concurrent implementation since the complex balancing measures are not involved in maintaining the index structure. Merkel tree is a summary algorithm and when given a particular combination method, it aims to hash each part of the message iteratively in order to guarantee the data accuracy. Because of the simplicity of hash operation, parallel operation is easy to implement

in terms of the Merkel tree. However, there are few works on the concurrency of the Merkle Patricia tree. In [16], a non-blocking Patricia tree has been proposed. It has realized the parallel insertion, deletion and replacement. The work [17] has devoted effort to the study how to optimize the efficiency of Merkle tree in a parallel condition, for example, via the most optimal tree topology and the amount of processors [18]. The work [19] has applied the Merkle Patricia tree to the task of diploma management. The proposed method not only supports quick query of transactions, but also supports the historical query of account information.

In this paper, we proposed an intelligent manufacturing security model based on blockchain, especially for smart factory. The proposed model has a distributed architecture in which the nodes supervise each other, which provides better security than traditional architectures. At the same time, the proposed model has adopted the MPT tree (seldom used in prior works) to expand blockchain and compared with the high computing cost in common blockchain query, queries can be conducted more quickly on MPT tree. However, when MPT tree becomes bigger, the efficiency on query and insertion will deteriorate. And due to the bottleneck on the performance that, every time when conducting insertion, the hash operation will be carried out by multiple times, therefore the MPT tree dose not support parallel operation. To solve this problem, we proposed a lock-free and concurrent-caching-supported Merkle Patricia tree, which supports lock-free and concurrent insertion, at the same time, it can improve the efficiency in multi-core systems.

3. The intelligent manufacturing security model based on blockchain

3.1. Overview

Currently, most intelligent manufacturing architectures are constructed in cloud-center-mode [20]. However, the Industrial Internet of Things (IIoT) systems in cloud-center-mode suffer from the low reliability due to the potential failure like the delayed communication. In other words, once the central node fails, the reliability of the whole IIoT system will not be ensured and the whole system will be in danger. Therefore, based on the blockchain, this paper proposed a distributed and weak centralized security model, which can effectively avoid the above problem of the traditional cloud-center-mode.

As shown in Figure 1, the proposed blockchain-based intelligent manufacturing security model, or architecture, includes five layers. The first layer is the physical resource layer, which includes all manufacturing resources involved in the whole manufacturing life cycle. The second layer is the network structure layer, which is the combination of bus and sensor network. It can obtain information of various devices, manage equipment conditions and place manufacturing orders. The third layer is the management hub layer, which takes the high-performance server node as the core. The data and order information collected by the management hub layer are encrypted and packed into blocks, and the consensus algorithm is used to achieve the consistency. In addition, the management hub layer also needs to schedule the production equipment of each factory, distribute manufacturing orders, integrate and operate intelligent manufacturing equipment. The fourth layer is the storage layer, which is responsible for storing encrypted block data. With the weak-centralized multi-center structure, each node stores the full amount of data, and achieves synchronization through consensus algorithm. When one node fails, the availability of the whole system will not be affected. The last layer is the application layer, which provides different services for users.

For the blockchain-based architecture proposed in this paper, each user, denoted as transaction

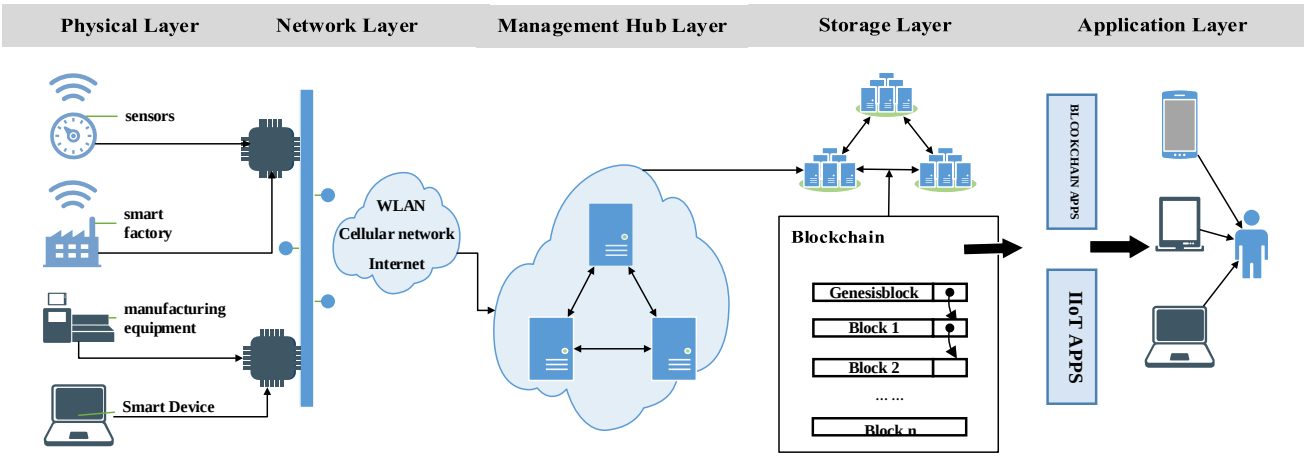

Figure 1. Improved blockchain-based IIoT architecture for a intelligent factory.

node or general node, needs identity authentication to join the system. Users can create blocks in the network, encrypt block information with public key and sign block information with private key in order to ensure the reliability and security of information. In addition to the transaction nodes, there are also nodes responsible for recording blocks, denoted as the central management node in the proposed blockchain-based architecture. Different from the suspicious system of bitcoin, the central management node has incorporated the trusted computing nodes. In this paper, we focus on the availability of resources and adopt the lightweight consistency algorithm such as PBFT or raft, which improves the scalability and real-time performance of the whole system to reduces the waste of computing resources and the high complexity of the existing consistency algorithm. According to the actual situation of industrial environment, each workshop or factory in the smart factory is equipped with one or more data centers. Moreover, multiple distributed nodes can reduce the pressure of a single central node and avoid the system failure caused by the collapse of a single central node. Finally, in Figure 1, the physical resource node receives the issued orders through the link management center. The physical resource node can send a verification request to the central management node in order to verify the correctness of the issued order from the blockchain.

3.2. The storage structure of the improved blockchain

Data and transactions are packaged into blocks through management nodes. In order to adapt to the application environment and scenarios of industrial Internet, this paper extends the transaction structure, as shown in Figure 2.

The block is composed of block head and block body. The block head includes the hash value of the previous block (prehash), the hash value of the current block, the creator's public key (ScriptPubkey), the block height, the timestamp, the Merkle transaction root and the Merkle root of the global state Merkle tree. All transaction information is stored in the k-v database. The block stores the hash summary generated by the transaction and stores it in the Merkle tree in chronological order. Only the hash summary is stored in the block chain, so the memory space is saved, and the block can be compressed by unloading the leaf node.

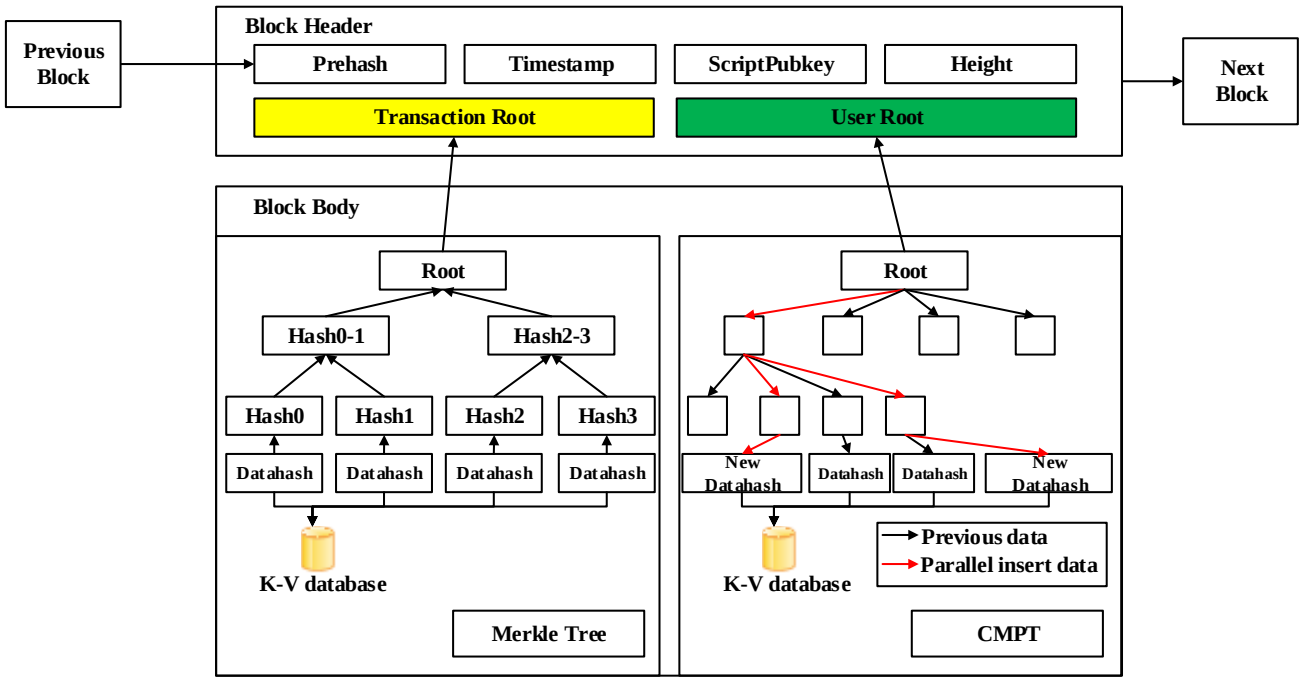

Figure 2. The improved blockchain architecture.

The format of the transaction is shown in Table 1. Each operation of the account or device will be sent to the management node through the network. After the management node packages the transaction and updates the global state tree, the latest running state of each device can be recorded. The management node builds the Merkle tree by hashing the two transactions packed into the block and updating the Merkle root to the block head, so as to realize the non-tampering of the transaction data.

Table 1. Transactions in the block body.

From Address	To address	Message	Signature
01f1	01ffb	Message1	Sign1
01ffb	01ffc	Message2	Sign2
01f1	01ffd	Message3	Sign3

For the proposed storage structure, each device or account is assigned with a unique public key, denoted as node ID, generated by a hash digest function, or a hash value of the node status. The management central node maintains the whole CMPT state tree and once the block has been constructed, it will insert transactions of the verified block into CMPT tree, updating the latest status of the entire system. After insertion, the root of CMPT tree will be obtained and updated in a concurrent way.

The update process of CMPT tree is shown in Figure 3. Transactions related to entities "01f1", "01ffb", "01ffc", "01ffd" are recorded in the current transaction tree. In the process of building the new block, CMPT tree processes the status of the entities involved in the transaction and updates the latest status to CMPT tree.

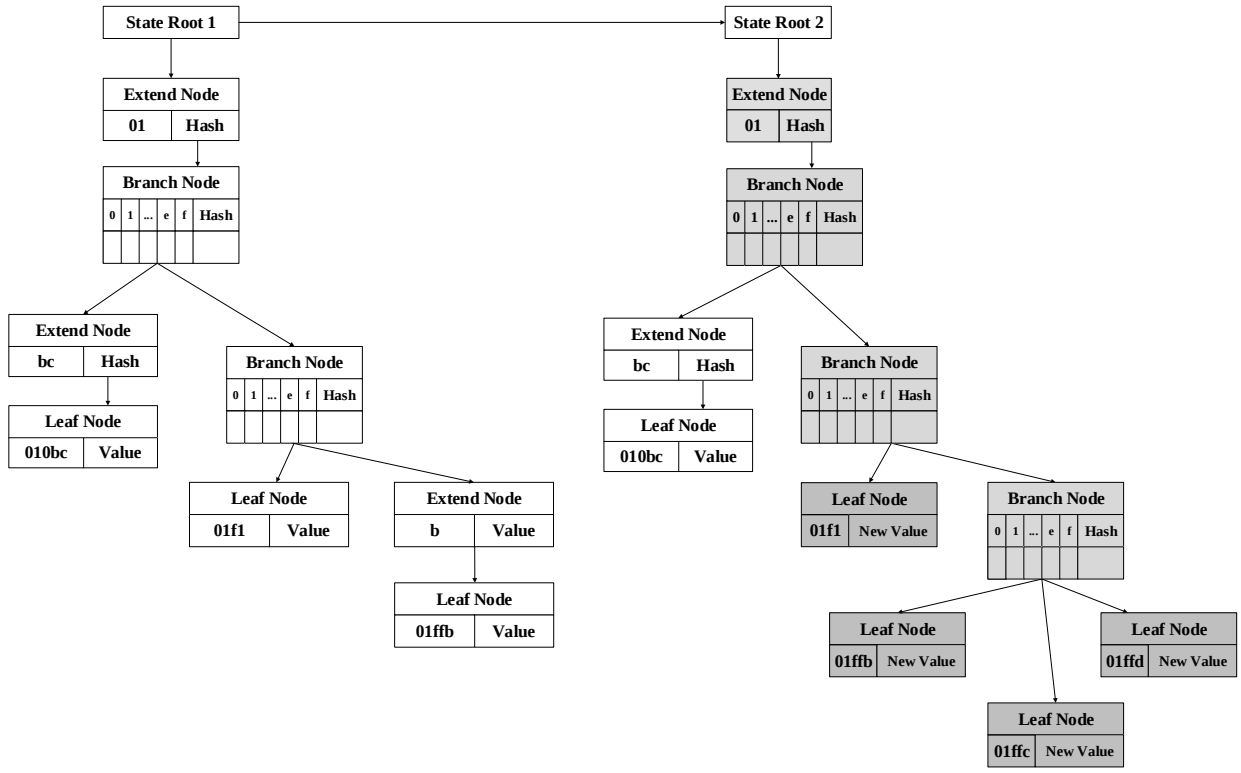

Figure 3. Insert operation example of CMPT.

The gray frame in the figure represents the modified block during packaging. By only calculating the hash of the modified block, the block building speed can be improved. When reading the latest state of the entity node from the CMPT tree, first try to find the hash value of the node state quickly through the CMPT tree cache. Otherwise, start to traverse from the Merkle root of the block header, and find the hash value of the node status in *O(logn)* sthe time. Then according to the hash value, we can find the corresponding information in the K-V database. In other words, it achieves a fast query of transaction and node status. In addition, the index is stored in the CMPT tree through hash operation, which ensures that the data will not be tampered. The optimization of CMPT tree will be explained specifically in Section 4.

4. Lock-free concurrent cache Merkle Patricia tree

The Intelligent Manufacturing Security Model based on blockchain proposed in this paper not only solves the problems of data security and system trust, but also tries to alleviate the performance bottleneck of traditional blockchain. In this section, we propose a tree structure called CMPT, which supports concurrent insertion and lock-free search. We also introduce how to use CMPT to extend conventional blockchain with simple pseudo code. Firstly, the CMPT tree we defined stores a set of key value pairs (k, v), which includes following operations:

- **insert (k,v)**: Given the key value pair (k, v), if there is no key value pair with key of k in CMPT, add a key value pair to the tree, otherwise replace the existing key value pair.

- **find (k)**: Given the key value k, if the CMPT contains the key of k, the value of the corresponding key value pair (k, v) is returned as (v), otherwise null is returned.
- **commit (Trie)**: Given a tree Trie, the information of all nodes is written to the memory database, and the hash value of root node is returned.

4.1. Data structure of CMPT

The data structure of the basic nodes of the CMPT tree is shown in Figure 4. The CMPT tree consists of four types of nodes. The LeafNode is the end of the tree, which also stores the data of the CMPT tree. In Figure 4, serialized hash value has been presented and the corresponding data is stored in the K-V database. HashNode is a simple node, which contains an array of hash values of corresponding nodes. BranchNode is a node with multiple child nodes. CMPT tree is a 16-fork Trie, which means that BranchNode contains a pointer at most of 16 child nodes. The 17th node stores a HashNode, which is used to store the hash values calculated by the child nodes. ExtendNode is an expansion node, which compresses the depth of the tree by merging BranchNode with only one child node to. Each node in the CMPT tree has 4 cache levels. The cache level of the root node is 0, and the cache level of its child node is 4. The CMPT tree made up of the above nodes ensures that if the CMPT tree contains a LeafNode with a hash value of h, then this LeafNode can be obtained by traversing from the root node, via a series of merging of BranchNodes and ExtendNodes.

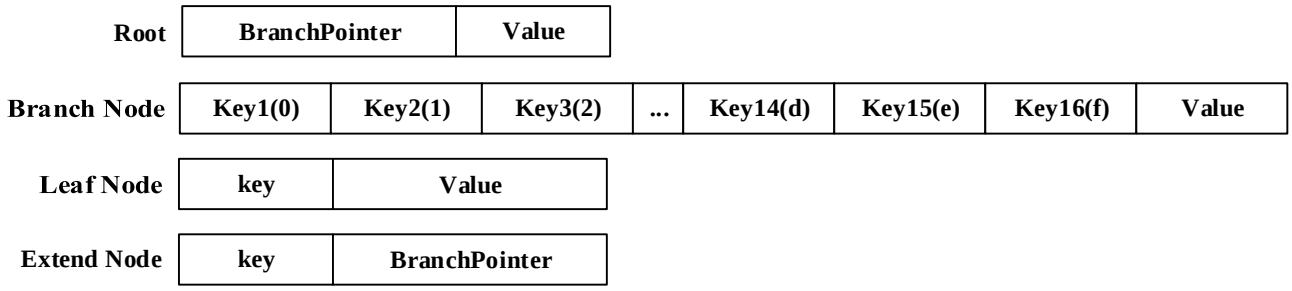

Figure 4. Basic CMPT data types.

4.2. Data insertion algorithm

Given a (k, v) key value pair, we start from the root node to find a BranchNode having a longest prefix, common with the given key value k. if the corresponding node is found, insert a new value into the node according to the following conditions:

- Case 1: If the BranchNode slot corresponding to the longest common prefix is empty, then, create a new LeafNode or ExtendNode and put it into the corresponding BranchNode, at the same time, put the key value pair (k,v) into the LeafNode, or the LeafNode pointed by the ExtendNode.
- Case 2: If there is a conflict, create a new BranchNode at the level of the ExtendNod with the longest common prefix, and insert two different ExtendNodes or LeafNodes.
- Case 3: When the conflict happened in the LeafNode, if the key value k is the same, replace the original key value pair (k,v).

Algorithm 1: Data insertion algorithm.

```
Require: (k, v)
Ensure: null
1:  insert (k, v)
2:    if(!insert(k, v, 0, root, null))
3:      insert(k, v)   #end of Procedure "insert(k, v)"
4:  insert(k, v, level, cur, prev)
5:    pos = k[level/8] »>(level%8)&15
6:    if(cur ∈ BranchNode)
7:      old = READ(cur.BranchKey[pos])
8:      else if(cur∈ExtendNode)
9:        old = READ(cur.Branchpointer)
10:     if(old == null)
11:       if(keylevel(key)==level))
12:         en = new LeafNode(k, v)
13:         else en=new ExtendNode(Subkey(k,level), LeafNode(k,v))
14:         if(CAS(cur.BranchKey[pos], old, en) || CAS(cur.Branchpointer, old, en)) return true
15:         else return insert(k, v, level, cur,prev)
16:       else if(old∈BranchNode)
17:         return insert(k, v, h, level+4, old, cur)
18:       else if(old∈ExtendNode)
19:         matchlen= prelen(subkey(k,level), old.hashnode.value)
20:         if(matchlen == old.hashnode.value)
21:           insert(k, v, keylevel(matchlen)+level,old,cur)
22:         else
23:           bn = extendbreanch(key, old)
24:           if(CAS(cur.BranchKey[pos], old, bn) || CAS(cur.Branchpointer, old, bn))
25:             return insert(k, v, keylevel(matchlen)+level, bn, cur)
26:           else return insert(k, v, level, cur, prev)
27:   else   #to "if" in line 6
28:         en = new LeafNode(k, v)
29:         if(CAS(cur.BranchKey[pos], old,en) || CAS(cur. Branchpointer,old,en))
30:           return true
31:         else return insert(k, v, level, cur, prev)
32:   return false   #end of Procedure "insert(k, v, level, cur, prev)"
```

Lines 5–9 reads the child node according to the hash value of the parent node in the way of atomic reading. For lines 10–15, when the child node is empty, then insert the data into the corresponding child node in the way of CAS as Case 1. Lines 16, 17, 20 and 21 mean that if there is a common prefix between the corresponding node and the current key value, then the insert operation should be conducted recursively in the next layer. Lines 23–26 mean that when the child node is ExtendNode and the current key have some prefixes common with it, then, expand the node as Case 2 and insert the

child node into the node at the next level. Lines 28–31 show that if the current node is a LeafNode, and replace the (k, v) value of the child node according to Case 3. Finally, the Algorithm 1 checks whether the insertion is successful. If it failed, the insert should be restarted from the root.

4.3. Data search algorithm

Given a key value k, we provide a query method to query the value v associated with k. if the key value k is not a part of the CMPT tree, the search returns a null value. The search algorithm is based on Eq (1), input the key value k, and start from the root node. For ExtendNode with the cache level l, the algorithm judges whether the maximum common prefix of the hash of ExtendNode and the sub array starting from the l-bit of hash is equal to itself. For BranchNode whose hash value and the key value are at level 1 of the caching, then, the algorithm takes the [l, l + 4) bit of hash as the start position of the next node in the array. The above processes are repeated until reach the LeafNode or empty node.

$$a_1 \xrightarrow{b1[p1]|e1} a_2 \xrightarrow{b2[p2]|e2} \ldots \xrightarrow{bn[pn]|en} a_n \tag{4.1}$$

Algorithm 2: Data search algorithm.

Require: k
Ensure: v

```
1:  find (k)
2:     find(k,0,root)    #end of Procedure "find(k)"
3:  find(k,level,cur)
4:     pos = k[level/8] »>(level%8)&15
5:     if(cur ∈ BranchNode)
6:        old = READ(cur.BranchKey[pos])
7:     else if(cur ∈ ExtendNode)
8:        old = READ(cur.Branchpointer)
9:     if (old == null)
10:       return null
11:    else if(old ∈ BranchNode)
12:       return find(k, level+4, old)
13:    else if(old ∈ ExtendNode)
14:       matchlen=prelen(subkey(k,level), old.hashnode.value)
15:       find(k, keylevel(matchlen)+level, old)
16:    else if(old ∈ LeafNode)
17:       return old.value()
18:    return null    #end of Procedure "find(k,level,cur)"
```

Lines 5–9 of the algorithm reads the hash value of the child node from the corresponding parent node in the way of atomic reading. Lines 9–18 recursively search different nodes. If the corresponding LeafNode is found, its hash value is returned. Otherwise, a null value is returned.

4.4. Cache based data operation optimization

In sections 4.3 and 4.4, we introduced the basic data query and insertion. However, the time complexity is *O(logn)*, which means with the increase of data volume, the search and insertion of data will slow down. In order to improve the performance of query and insert, we extend CMPT tree with additional cache array structure. The cache array stores the references of nodes where most of the keys are located. In this way, it can optimize the time complexity of the query and insert operations to *O(1)*.

The caching structure is shown in Figure 5. The cache stores references to index nodes at the current cache level in an array. The first slot of the array holds a special object CacheNode, which points to the cache array of the previous level, as shown in Figure 5. Each cache level has a cache array. The current level always points to the cache array with cache level 0. The misses array of each layer counts the number of missing cache. The improved search algorithm first calls the *fastfind* method to read the data from the current cache array at the corresponding location of the common prefix. If the cache array is not built, the regular *find* method is called, otherwise, the value corresponding to the common prefix position is read from the cache array at the highest cache level. Then *find* method is used to query data according to the cache result. A successful quick look-up does not update the cache. To maintain the cache array, there are two operations for each *find*. First, if the cache level of the found node is equal to the cache level of the current cache array, the node is added to the cache array. Secondly, if the cache level of the current node is greater than that of the current cache array, the *cachemiss* method is called in the CacheNode of the current cache array to record the cache failure. If the number of failures on the current cache level reaches a threshold, an attempt is made to adjust or raise the cache level.

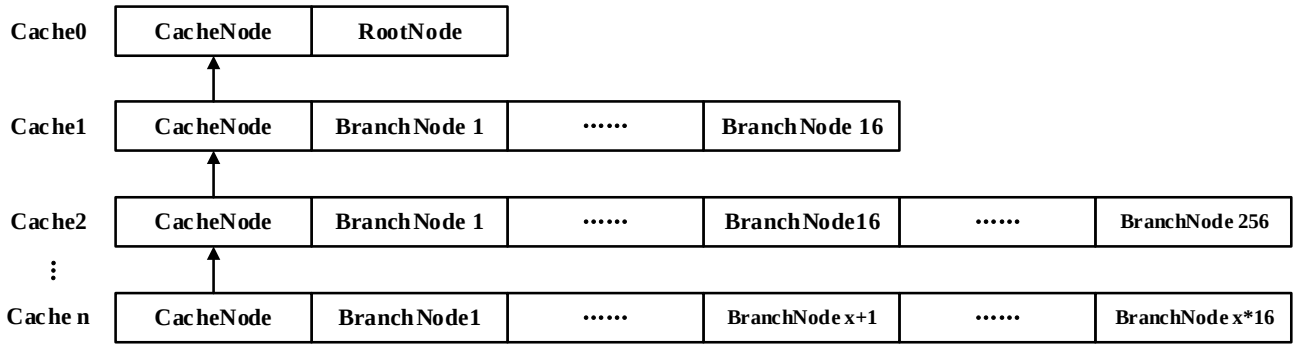

Figure 5. Cache data types.

In order to maintain the cache, we improved the data operation algorithm in Sections 4.3 and 4.4. Lines 5–9 of algorithm 3 show that when the improved *find* method encounters nodes with the same cache level as the current cache, it will call the *Recordcache* method to update the cache. When it encounters nodes that deviate from the current cache level, it will call the *cachemiss* record to fail once. Algorithm 4 shows the update of cache array. The CMPT tree does not need cache array when it is at low cache level. When the cache level of the current node reaches 12, the cache array will be created from cache level 8. If the level of the current node and the cache array is the same, the algorithm will store the node into the cache array to maintain the cache.

Algorithm 3: Cache-based improved data search algorithm.

```
Require: k
Ensure: v
find(k, level, cur, cache, cachelevel):
  if(level == cacheLevel)
    recordcache (cache, cur, level)
    ......
  else if (old ∈ ExtendNode)
    if(level <cachelevel || level >cachelevel+4)
      cachemiss()
    if(level+4 == cachelevel)
      recordcache(cache, old, level+4)
    ......
    return find(k, level+4, old, cache)
#end of Procedure "find(k, level, cur, cache, cachelevel)"
fastfind(k):
  cacheold = READ(cache)
  if(cacheold == null)
      return find(k,0,root,null,-1)
  toplevel = countbits(cache.length - 1)
  while(cache != null)
    pos = 1 + k&(cache.length-2)
    cachenode = READ(cache[pos])
    level = countbits (cache.length - 1)
    if(cachenode ∈ ExtendNode)
      return find(k, keylevel(cachenode.value) + level, old)
    else if (cachenode ∈ BranchNode)
      return find(k, level, cachenode)
    cache = cache[0].parent
    return find(k, 0, root, null, toplevel)
#end of Procedure "fastfind(k)"
```

4.5. The parallel hash computation for the root of Merkle Tree

The traditional Merkle tree is a balanced binary tree, so it allows multiple sub-trees to be processed in parallel at the same level. The hash computation for the root of the Merkle tree can be completed by splitting the tree into multiple parts with the same size, at the same time in a parallel way. However, the CMPT tree is a prefix tree, not a balanced tree. The CMPT tree cannot be evenly split into several subtrees when the overall structure and data volume is unknown. Therefore, this paper improves the concurrent hash computation.

If the CMPT tree does not have a cache array when given low data volume, and the parallel computing has limited improvement on efficiency, then the hash operation of the Merkle root will be computed serially. As shown in Figure 6, if the cache exists, the hash is divided into two phases. First,

Algorithm 4: Cache maintenance algorithm.

Require: k, node, cachelevel

Ensure: null

```
1:  Recordcache(cache: Array[], node: Any, cachelevel: Int):
2:    if (cache = = null)
3:      if( cachelevel >= 12)
4:        cache = createcache(8, null)
5:        CAS(cacheHead, null, cache)
6:        Recordcache(cache, node, cachelevel)
7:      else
8:        length = cache.length
9:        cachelevel = countbits( length Â¨C 1 )
10:       if(cachelevel == cachelevel)
11:         val pos = 1 + k & (cache.length-2)
12:         WRITE(cache[pos], node)
13: #end of Procedure "Recordcache(cache, node, cachelevel)"
```

fine-grained uniformed allocation is performed on the cache array at the current cache level, and nodes are processed in parallel at the current cache level tree. After the entire cache array is computed, the remaining part is divided into *n* sub-trees and handed over to n processors for parallel computing. Finally, the hash root of the CMPT tree is computed in aggregate.

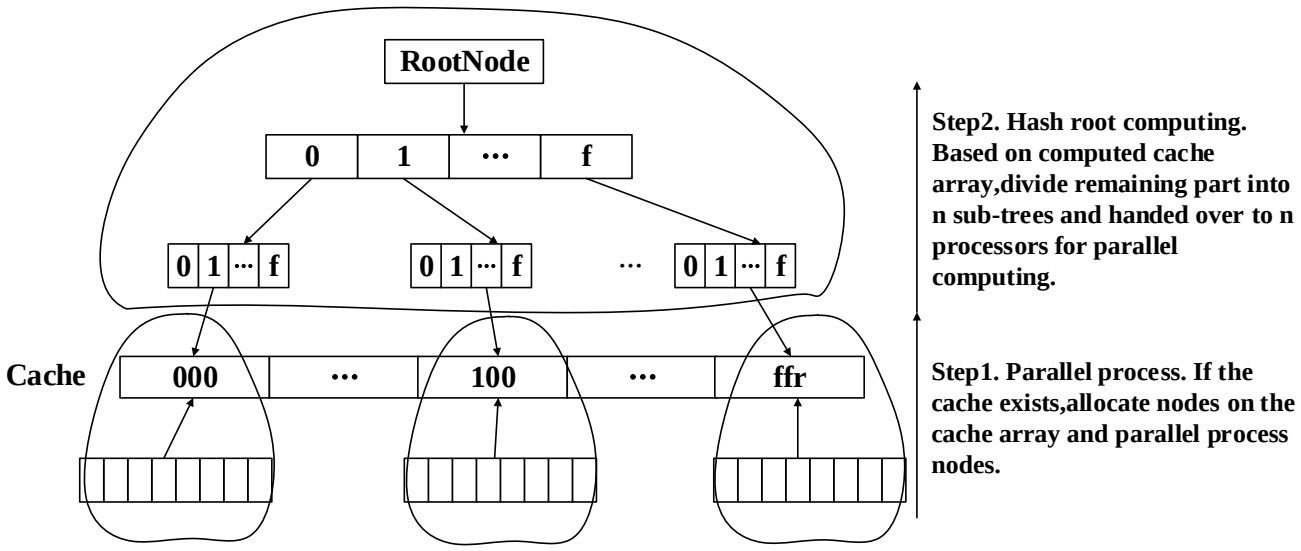

Figure 6. Example of the computation of the Merkle root.

4.6. Correctness analysis of lock-free algorithms

For the MPT tree shared by multiple threads, if there is no concurrency control, when two threads operate the same node at the same time, it will result in multi-thread conflicts, data loss or dirty read

problems. We will analyze the correctness of concurrent operations from the aspects of security and activation.

4.6.1. Security analysis

Security means that the concurrent data-processing is error-free. In order to prove the correctness of the algorithm, we have determined the linearization point. For example, the three CAS instructions on line 14, 24, and 19 of algorithm 1 are linearization points. Excluding such three instructions, the other three parts of the algorithm will not modify the tree. The CAS instruction is an atomic operation. When multiple threads are competing for modification, CAS can ensure that only one of the threads can operate successfully, avoiding the problems such as the loss of data. Therefore, the operation on CMPT tree is secure.

4.6.2. Activation analysis

The operation algorithm of the CMPT tree is lock-free. We can prove that any state change of the tree will be completed in a limited number of steps. Take the data insertion as an example. First, the recursive call will never decrease the cache level, and only if the CAS operation fails, the insert operation will be recalled at the current cache level. From Eq (1), it can be known that each leaf node can be obtained from root, via the limited access of several nodes, so there are only a limited number of steps between the two CAS operations. However, due to the characteristics of CAS operation, it can be observed that, if a CAS operation C0 failed, then between t0 (when C0 reads the expected value) and t1 (when CAS operation failed), a successful CAS operation C1 would must occur in this period of time (t0-t1). From the above, the steps between the two states of the tree are limited. Therefore, the operation on CMPT tree is lock-free.

5. Experiments

5.1. Experimental setup

The experiment was performed on Ubuntu 16.04. The main hardware configuration is as follows: Cpu: Intel (R)_Core (TM)_i7-4790K; 4.0GHz*4core*2; RAM: 32GB; and Oracle JDK is installed 8 environment, set the heap memory to 6G. In addition, the task of this paper is to increase the efficiency of the blockchain (i.e., the time consumed for query when processing different amounts of transactions) so that more power can be conserved in blockchain. In this way, the security model based on the extended blockchain can prevent the single-point-failure due to the power consumption. Therefore, for evaluation metric, we use the number of transactions per second (TPS) and the number of successful queries in a fixed time to measure the efficiency of the extended blockchain.

For the implementation on blockchain of the Ethereum, MerkleRoot needs to be generated by packaging the transaction process. And, the state can be updated by updating the MerkleRoot of the MPT tree. For the model proposed in this paper, the blockchain is implemented by collecting data and constructing the Merkle root with the help of the central management node. In addition, the latest index of the state of corresponding industrial equipment in blockchain can be obtained by updating the node value in CMPT tree and the root of Merkle. Finally, the hash digest of the latest state will also be packaged into the blockchain.

This paper focuses on the index method of data and node status in the blockchain, not involved with the network and distributed consensus. Consequently, we have not conducted the experiments on clusters. All the codes are built from scratch.

5.2. Evaluation metric

The evaluation metrics are listed as follows:

1) The main measurement lies in the number of transactions per second (TPS) in terms of the blockchain architecture. TPS is the number of increased blocks in blockchain times the size of each block and then divided by the average amount of transactions. The higher the TPS, the more transactions processed per second, and the better the overall system performance.

2) The number of successful queries in a fixed time, with the increase in the amount of data. The higher the number, the better the performance of the index, and the more efficiently the device can communicate with the central management node.

In this paper, we implemented the CMPT tree under JDK8 using the Java language, and compared it with the MPT tree implemented by the Elixir Ethereum standard library [21], and designed several sets of experiments to test the performance of the CMPT tree. We conducted five times per group and defined the average as the running time of the algorithm. The compared methods include:

- Merkle Patricia Trie
- Concurrent Merkle Patricia Trie
- Concurrent Merkle Patricia Trie Â¨C 4 thread

5.3. Experimental analysis

5.3.1. The construction performance of CMPT tree

We compared the 4-thread CMPT with the single-thread CMPT and the single thread MPT. We sets the number of transactions in each block to 64, 128, 256, 512, 1024, 2048, 4096, 8192. As shown in Figure 7, the performance of CMPT is similar to that of MPT when given low data volume. The reason is that because caches are not built at low tree height, and both the insert and hash operations have low complexity. But when higher data volume is given, the single and multi-thread CMPT have obviously outperformed the single-thread MPT, especially when given higher data volume, every operation on MPT needs to traverse from the leaf node in order to obtain the root of the hash-tree and the computing complexity of insertion is *log(n)*, a pretty high cost in terms of the efficiency.

5.3.2. Query performance of CMPT tree

For the model proposed in this paper, the architecture maintains the CMPT tree. Following the principle like querying the hash value of each device to obtain the latest status of each device, we designed the experiment to test the time cost of every 10 thousand queries via a tree composed of different data volumes. As shown in Figure 8, due to the assistance of the caching, when given different data volumes, the time cost of every 10 thousand queries remains stable and the multi-thread implementation have little repercussion on the efficiency of the query. However, when the amount of data has increased, the MPT's time cost for query will increase dramatically. Therefore, it can be

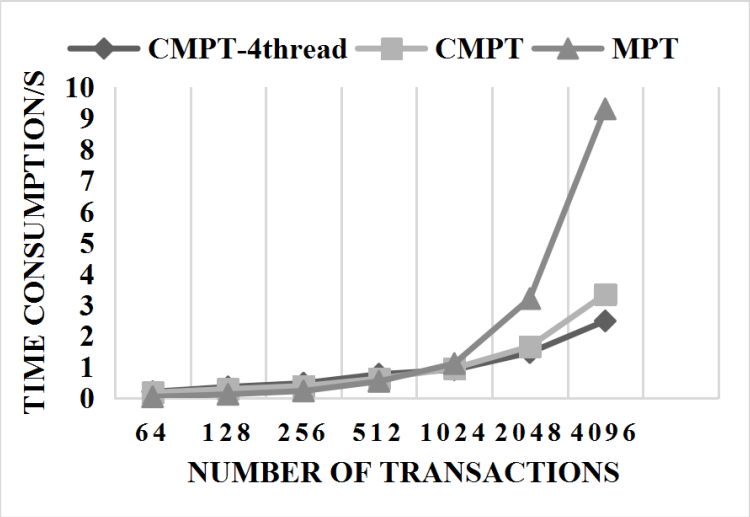

Figure 7. The comparison of CMPT and MPT build cost.

observed that, in the scenario of huge data volume, CMPT tree can maintain the excellent query performance.

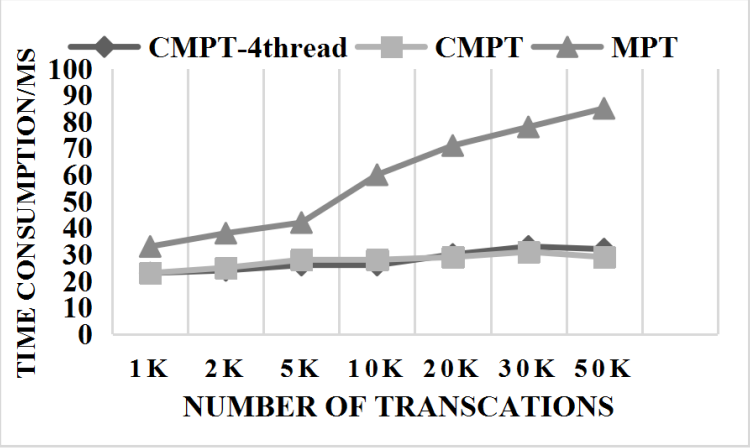

Figure 8. The comparison of CMPT and MPT query cost.

5.4. Experimental summary

Based on the above experiments, the performance of the CMPT tree proposed in this paper is similar to that of regular MPT tree when given low data volume, but in the intelligent manufacturing industry scenario, the CMPT tree maintains scalability as well as high performance. It also supports thread-safe and parallel data operations, speeds up block verification or construction, and further improves the transaction throughput. In addition, when adopting the account-based model, current transactions depend on historical records. And, the CMPT tree can provide efficient query on historical records which improves the availability of the entire blockchain when given a huge amount of data or application nodes.

6. Conclusions

Blockchain has the characteristics of security, privacy, non-tampering and can be deployed in a distributed way. This paper proposes a distributed intelligent manufacturing security model for smart factories based on the blockchain. The hierarchical structure of the proposed model is introduced and analyzed. Such a model can solve the single point of failure and the limitation on scalability in terms of the traditional IIoT systems. An improved blockchain architecture based on the MPT tree is proposed, the storage mode of the blockchain is optimized, and the speed of data query is accelerated. Aiming at the performance bottleneck of the MPT tree when given a large amount of data, a lock-free concurrent cache Merkle Patricia tree (CMPT tree) is proposed to provide lock-free concurrent data operations. The experiments have shown that the CMPT tree proposed in this paper can effectively improve the performance of the blockchain system in the high data volume scenario when it comes to the intelligent manufacturing industry.

Acknowledgments

The authors extend their appreciation to the Deanship of Scientific Research at King Saud University for funding this work through Research Group no. RG-1441-331.

Conflict of interests

The author declares no conflict of interests.

References

1. Y. Liao, E. Loures, F. Deschamps, Industrial Internet of Things: A Systematic Literature Review and Insights, *IEEE Int. Things J.*, **5** (2018), 4515–4525.
2. R. Drath, A. Horch, Industrie 4.0: Hit or Hype? [Industry Forum], *IEEE Ind. Electron. Mag.*, **8** (2014), 56–58.
3. A. Sadeghi, C. Wachsmann, M. Waidner, *Security and privacy challenges in industrial Internet of Things*, 2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC), San Francisco, CA, 2015, 1-6.
4. T. M. Fernandez-Carames, P. Fraga-Lamas, A Review on the Application of Blockchain to the Next Generation of Cybersecure Industry 4.0 Smart Factories, *IEEE Access*, **2019** (2019), 45201–45218.
5. M. Samaniego, R. Deters, *Internet of Smart Things-IoST: Using Blockchain and Clips to Make Things Autonomous*, 2017 IEEE international conference on cognitive computing (ICCC), IEEE, 2017.
6. S. Nakamoto, Bitcoin: A Peer-to-Peer Electronic Cash System [EB/OL], *Consulted*, **1** (2008), 28.
7. Bitcion charts and graphs-blockchain[EB/OL], Available from: https://www.blockchain.com/zh-cn/charts.
8. Ethereum project[EB/OL], Available from: https://www.ethereum.org/.

9. S. K. Lo, Y. Liu, S. Y. Chia, X. W. Xu, Q. Lu, L. Zhu, et al., Analysis of Blockchain Solutions for IoT: A Systematic Literature Review, *IEEE Access*, **7** (2019), 58822–58835.
10. J. Bonneau, *EthIKS: Using Ethereum to Audit a CONIKS Key Transparency Log*, International Conference on Financial Cryptography and Data Security, Springer, Berlin, Heidelberg, 2016.
11. R. Zhang, R. Xue, L. Liu, Security and Privacy on Blockchain, *ACM Comput. Surv.*, **52** (2019), 1–34.
12. L. Bai, M. Hu, M. Liu, J. Wang, BPIIoT: A Light-Weighted Blockchain-Based Platform for Industrial IoT, *IEEE Access*, **7** (2019), 58381–58393.
13. J. Wan, J. Li, M. Imran, D. Li, F. Amin, A Blockchain-Based Solution for Enhancing Security and Privacy in Smart Factory, *IEEE Trans. Ind. Inf.*, **15** (2019), 3652–3660.
14. M. Liu, F. Yu, Y. Teng, V. Leung, M. Song, Performance Optimization for Blockchain-Enabled Industrial Internet of Things (iiot) Systems: A Deep Reinforcement Learning Approach, *IEEE Trans. Ind. Inf.*, **15** (2019), 3559–3570.
15. L. Zhou, L. Wang, Y. Sun, P. Lv, Beekeeper: A Blockchain-Based IOT System With Secure Storage and Homomorphic Computation, *IEEE Access*, **6** (2018), 43472–43488.
16. N. Shafiei, Non-blocking Patricia tries with replace operations, *Distrib. Comput.*, **32** (2019), 423-442.
17. K. Atighehchi, R. Rolland, Optimization of Tree Modes for Parallel Hash Functions: A Case Study, *IEEE Trans. Comput.*, **66** (2017), 1585–1598.
18. D. Yue, R. Li, Y. Zhang, W. Tian, C. Peng, *Blockchain Based Data Integrity Verification in P2P Cloud Storage*, 2018 IEEE 24th International Conference on Parallel and Distributed Systems (ICPADS), IEEE, 2018.
19. Y. Xu, S. Zhao, L. Kong, Y. Zheng, S. Zhang, Q. Li, ECBC: A High Performance Educational Certificate Blockchain with Efficient Query, International Colloquium on Theoretical Aspects of Computing. Springer, Cham, 2017.
20. T. Hegazy, M. Hefeeda, Industrial Automation as a Cloud Service, *IEEE Trans. Parallel Distrib. Syst.*, **26** (2015), 2750–2763.
21. Q. Qu, I. Nurgaliev, M. Muzammal, C. S. Jensen, J. Fan, On spatio-temporal blockchain query processing, *Future Gener. Comput. Syst.*, **98** (2019), 208–218.
22. Q. Zhu, S. W. Loke, R. Trujillo-Rasua, F. Jiang, Y. Xiong, Applications of Distributed Ledger Technologies to the Internet of Things: A Survey, *ACM Comput. Surv.*, **52** (2019), 1–34.

AIMS Press