Peer-to-Peer Networking and Applications (2020) 13:1486–1504
https://doi.org/10.1007/s12083-020-00895-5

A blockchain-based data storage framework: A rotating multiple random masters and error-correcting approach

Yuqi Fan[1] · JingLin Zou[2] · Siyu Liu[3] · Qiran Yin[1] · Xin Guan[1] · Xiaohui Yuan[4] · Weili Wu[5] · Dingzhu Du[5]

Received: 10 April 2019 / Accepted: 18 February 2020 / Published online: 5 April 2020

Abstract
A blockchain is resistant to modification based on the consensus of the network majority, which requires a large amount of communication among distributed nodes. Existing data dissemination protocol solves the wrong block problem also at a high communication cost. This paper investigates the error-correcting tamper-proofing data storage problem and proposes a three-layer framework to store the data in the blockchain to achieve data integrity. The data are stored as blocks, and we design a two-dimension chain data structure consisting of horizontal and vertical chains. This paper proposes a Rotating multiple random Masters and Error-Correcting data storage framework based on blockchain (RMEC) to strike a trade-off between system decentralization and the amount of communication. The proposed Rotating Multiple Random Sampling consensus mechanism (RMRS) randomly selects multiple temporary master nodes to handle each data access request so as to reduce the communication cost. We also propose two error-correcting mechanisms to validate and correct the wrong data blocks. Finally, we implement a prototype and conduct analyses on the system performance. The experiments demonstrate that the framework can achieve data tamper-proof and effectively reduce the communication cost.

Keywords Blockchain · Data integrity · Data storage · Error-correcting

1 Introduction

The traceability and integrity of data are critical for many data storage systems. Traditional centralized and distributed storage systems face the problem that the data are prone to be tampered by outside or internal attackers. Two techniques, digital signature and digital watermarking are currently widely used to protect data integrity. Using the digital signature, the sender signs the protected data with the tag of the data, and the tag is the hash value of the data with the private key of the sender using the public key cryptographic mechanism. The sender transmits both the data and the tag to the receiver who verifies the integrity of the received data. That is, the receiver calculates the hash value of the received data using the public key of the sender. If the hash value calculated is the same as that received, the verification is successful and the data are not modified; otherwise, the verification failed and the data are modified. Digital watermarking embeds digital markup information in a carrier signal, such as digital multimedia data, by means of signal processing. Such information is usually invisible and can be extracted only through a dedicated detector or a reader algorithm. Digital watermarks may be used to verify the authenticity or integrity of the carrier signal.

However, there are some known attacks to these two techniques of digital signature and digital watermarking, such as the attack from the internal attackers inside the system [1–5]. Blockchain has the characteristics of immutability, decentralization, distributed ledgers, consensus, etc. Compared to the traditional data storage systems, blockchain

✉ Xiaohui Yuan
xiaohui.yuan@unt.edu

Yuqi Fan
yuqi.fan@hfut.edu.cn

1 School of Computer and Information Engineering, Anhui Province Key Laboratory of Industry Safety and Emergency Technology, Hefei University of Technology, Hefei, Anhui, 230601, China

2 School of Cyber Science and Engineering, Wuhan University, Wuhan 430072, China

3 School of Computer Science and Technology, Harbin Institute of Technology, Harbin 15001, China

4 Department of Computer Science and Engineering, University of North Texas, Denton, TX 76203, USA

5 Department of Computer Science, University of Texas at Dallas, Richardson, TX 75080, USA

greatly increases the difficulty of tampering the stored data, and hence blockchain has been applied to data storage to improve the data security [6–11].

Blockchain platforms use decentralized consensus to maintain consistency in a distrubted environment, and hence consensus is the *sine qua non* of blockchain. The state-of-art consensus algorithms, e.g. Proof-of-Work (PoW) [12], Proof-of-Stake (PoS) [13], Delegated PoS (DPoS) [14], Practical Byzantine Fault Tolerance (PBFT) [15], Algorand [16], YAC [17], etc., either require coins (stake) and a large amount of computation or use a centralized point which is in contradiction with the decentralization rationale of blockchain.

A wrong block in the blockchain causes data inconsistency [18]. In addition, a wrong block may potentially cause all the successive blocks to be wrong, which makes the node useless. Studies have been conducted to detect such inconsistency [19–21]. The majority rule is often adopted to obtain the correct data. For example, the Gossip data dissemination protocol maintains data consistency among the nodes by transmitting information about consistency to other nodes [22]. Nevertheless, the transmission cost is high due to the large and complex structure of Hyperledger Fabric [23].

In this paper, we achieve data integrity in data storage applications based on blockchain. The data, the hash value of the data and other information are stored as blocks in the blockchain, and the clients can search the data without storing the data. The contributions of this paper are as follows:

(1) We investigate the error-correcting and tamper-proofing data storage problem and propose a Rotating multiple random Masters and Error-Correcting data storage framework based on blockchain (RMEC), where the framework consists of Application Layer, Processing Layer, and Infrastructure Layer. We also propose a two-dimension chain data structure to store the data. The two-dimension chain data structure divides the blockchain into two parts: horizontal chain and vertical chain. The horizontal chain is generated to store the basic information of data, and a block is created in the horizontal chain for the data. We maintain a vertical chain for the data. When data are updated, a new block is generated to store the updated data in the vertical chain. All the blocks containing the updated information of the data form a vertical blockchain.

(2) We propose a Rotating Multiple Random Sampling (RMRS) consensus mechanism to balance decentralization and communication costs. A portal in the infrastructure layer receives the client's requests and forwards the requests to the servers for data storage and query. When storing data, the portal randomly selects K servers as temporary master nodes that own the right to generate and broadcast the block copy. The other nodes receive the block copy sent by each temporary master node, compare the K block copies, and keep the copy with the number of duplicated times no less than $\lceil \frac{K}{2} \rceil$. In the data query process, K servers are also randomly selected by the portal as the temporary master nodes. After verifying the queried block, the K temporary master nodes send the block data copies to the client. The client accepts the data copy with the number of duplicated times no less than $\lceil \frac{K}{2} \rceil$. Each node transmits the hash value of the data instead of the data itself during the consensus.

(3) We propose two data tamper-proofing mechanisms (single-block validation mechanism and periodic blockchain verification mechanism) to correct the wrong blocks and ensure data integrity. The single-block validation mechanism has two main functions: verifying the integrity of a block in a local server and correcting the wrong block. With the periodic verification mechanism, each server periodically verifies the correctness and integrity of all blocks in the blockchain and invokes the single-block validation to correct the wrong block.

(4) We implement a prototype and conduct analyses on the system performance. The experiments demonstrate that the framework can achieve data tamper-proof and effectively reduce the communication cost.

The rest of the paper is organized as follows. The related work is introduced in Section 2. The proposed framework RMEC and the modules are presented in Sections 3 and 4, respectively. The system performance is analyzed and evaluated in Sections 5. The paper concludes in Section 6.

2 Related work

Two techniques, digital signature and digital watermarking are currently widely used to achieve data integrity. Using the digital signature, the sender signs the protected data with the tag of the data, and the tag is the data hash value calculated with the private key of the sender using the public key cryptographic mechanism. The sender transmits both the data and the tag to the receiver who verifies the received data integrity. That is, the receiver calculates the hash value of the received data using the public key of the sender. If the hash value calculated is the same as that received, the verification is successful and the data are not modified; otherwise, the verification fails and the data are modified. However, the following risks exist using the digital signature: (1) there exist attack methods to some of the public key cryptography; for example, it is possible to use a brute-force attack to find the private key in practice; (2) it is difficult to prevent the attack from the internal attackers inside the system [1].

Digital watermarking embeds digital markup information in a carrier signal, such as digital multimedia data, by means of signal processing. Such information is usually invisible and can be extracted only through a dedicated detector or a reader algorithm. Digital watermarks may be used to verify the authenticity or integrity of the carrier signal. However, there are some concerns with digital watermarking: (1) the location of the watermark in digital data must be imperceptible; (2) digital watermarking is easily lost due to signal distortion incurred by various unintentional or intentional signal processing; (3) editing information is easy to affect or even destroy the watermark; (4) there are some known attacks on digital watermarking, such as stego only attack, known cover attack, known message attack, and so on [2].

The blockchain is a new technology that combines distributed data storage, peer-to-peer (P2P) transmission, consensus mechanism, encryption algorithms, etc. It is a new distributed infrastructure and computing paradigm, which uses the block-chain data structure to store and verify data, exploit distributed node consensus algorithm to generate and update data, and apply cryptography to ensure data transmission and access security. In a narrow sense, the blockchain is a type of chain data structure that combines data blocks in chronological order and distributed ledger using cryptography to guarantee the data integrity [24]. Compared to traditional data storage systems, the nodes in the blockchain system do not need to trust each other or any third party [25–28], and hence the blockchain greatly increases the difficulty of tampering the data stored in the system. The blockchain achieves data tamper-proof with the following methods.

(1) The blockchain uses SECP256K1 signature algorithm to generate public keys for identity authentication [29]. Unlike a signature algorithm based on the mathematical problem that a large prime number product is difficult to decompose, SECP256K1 is founded on the mathematical problem similar to the elliptic curve grounded on a discrete logarithm solution. The mathematical problem cannot be solved theoretically and it can hardly be solved by brute-force either, which guarantees an indispensable modification in the process of information transmission in the blockchain transaction process.
(2) The blockchain is also a P2P network without a central node. All the nodes involved in the calculation are connected in a P2P mode, and each node saves a copy of the data block. The consistency of the block replicas can be guaranteed by consensus protocol so as to achieve the fault-tolerance and security of the blockchain system.
(3) A block is linked to the previous block via a hash pointer so that one has to recalculate the hash value of all the successive blocks if he wants to modify a block, which increases the computational complexity [24].

The consensus protocol, the core of the blockchain, is a vital approach to preventing the data tamper. The state-of-the-art algorithms are PoW, PoS, DPoS, PBFT, Algorand, YAC, etc. With PoW [12], each block has a corresponding hash value calculated by the SHA256 algorithm, and the nodes achieve the consensus by comparing the speed of the hash value calculation. A node with high computing power has a good chance of working out the hash value and thereby has a high probability to obtain the right to generate the block [30]. Therefore, PoW faces the security risk of excessive concentration of power. Different from PoW, PoS determines the next node which can create the next block via various combinations of random selection and wealth or age [13]. With DPoS [14], a block is created by a trusted user node (trustee) elected by the community; the users vote to elect the trustee, and the users ranked the top 101 in the final vote become the trustee. Algorand has stake-weighted voting by validators and combines the stake values with a cryptographic sortition function to choose the voting committee [16]. In general, PoS, DPoS, and Algorand are used for financial purposes because of the stake.

With PBFT [15], the nodes are sequentially ordered with one node being the master and others referred to as backup nodes. PBFT consensus rounds are called views. The nodes that have the right to generate the blocks are determined by PBFT's view rotation mechanism and Moore's calculation formula. All nodes in the system communicate with one another with the goal being that all honest nodes will come to an agreement of the system state using a majority rule. Therefore, PBFT requires a huge communication cost.

The basic process of YAC [17] is as follows: (1) after generating and signing a transaction, the client sends the transaction to a peer; (2) the peer performs stateless validation of the transaction, and relays the transaction to the ordering service; (3) the ordering service generates a proposal containing an ordered list of transactions and sends the proposal to the peers; (4) a supermajority of peers determine whether a block can be generated, where the supermajority is defined to be a number greater than two-thirds of all peers in the network; (5) the peer commits the block to its local block store. YAC reduces the communication cost by using the centralized ordering service which is responsible for creating block proposals. However, YAC has the following concerns: (1) the proposal is generated and broadcast by the ordering service which is a centralized point with a strong assumption that the ordering service is safe and honest; (2) when a user sends a signed transaction to a peer, it is not randomly selected, which increases the possibility of system being compromised.

Some studies apply the blockchain to data storage. A blockchain-based data storage framework was built using Ethereum [31]. To cope with the problem of the slow speed of generating blocks in Ethereum, the system adds a centralized point to convert the data into transactions for packaged storage, while extra processor overhead is demanded and the new centralized point is prone to be attacked. A decentralized data storage model suitable for storing metadata was proposed to improve the security of cloud storage [6]. A decentralized privacy data management method was proposed, which uses the blockchain to control access rights and effectively protects the private data [7]. A secure P2P-type storage scheme was proposed to use the secret sharing and the blockchain; user data are divided into different parts, assigned to different nodes, and the node attacking the other nodes can be identified if the nodes supervise each other [8]. A blockchain-based access control system for cloud storage was proposed; the intelligent contract implemented with Ethereum stores all the information related to security operations in the blockchain, and the access control scheme is based on attribute encryption [9]. A blockchain-based data sharing verification scheme uses the blockchain to record the hash value and other necessary information of data sharing by other people and guarantee that the data that the user receives from a third-party source is the originally uploaded data by comparing the hash value of the corresponding data [10]. A consortium blockchain made of different autonomous vehicle manufacturers was proposed to ensure the authenticity and integrity of firmware updates, without using the centralized third parties to distribute the new firmware updates [32]. A decentralized ride-sharing service system based on the public blockchain was proposed to preserve the anonymity and privacy of the users and the data [33]. A decentralized charging coordination mechanism based on the blockchain was built to avoid a centralized charging coordinator and provide privacy protection [34]. A distributed privacy-preserving smart parking system was proposed to use a consortium blockchain created by different parking lot owners, and the system stores the parking offers on the blockchain to ensure security, transparency, and availability [35].

3 Rotating multiple random masters and error-correcting data storage framework

Our proposed Rotating multiple random Masters and Error-Correcting (RMEC) data storage framework based on blockchain adopts a three-layer hierarchical structure consisting of Application Layer, Processing Layer, and Infrastructure Layer, as shown in Fig. 1. The infrastructure layer includes a portal and N servers interconnected via P2P networks. The portal receives the data access requests and forwards the requests to the servers for processing. Each server establishes and maintains a blockchain for storing data. Each block in the blockchain includes data, the hash of the data, timestamp, user information, other data-related information, and the hash of the previous block. The processing layer consists of four modules, e.g. data upload, single-block validation, periodic blockchain verification, and data query, which realize the functions of data storage, data search, and data verification.

The data upload module is responsible for appending the data submitted by the application layer to the blockchain in each server at the infrastructure layer. The single-block validation module is responsible for verifying the

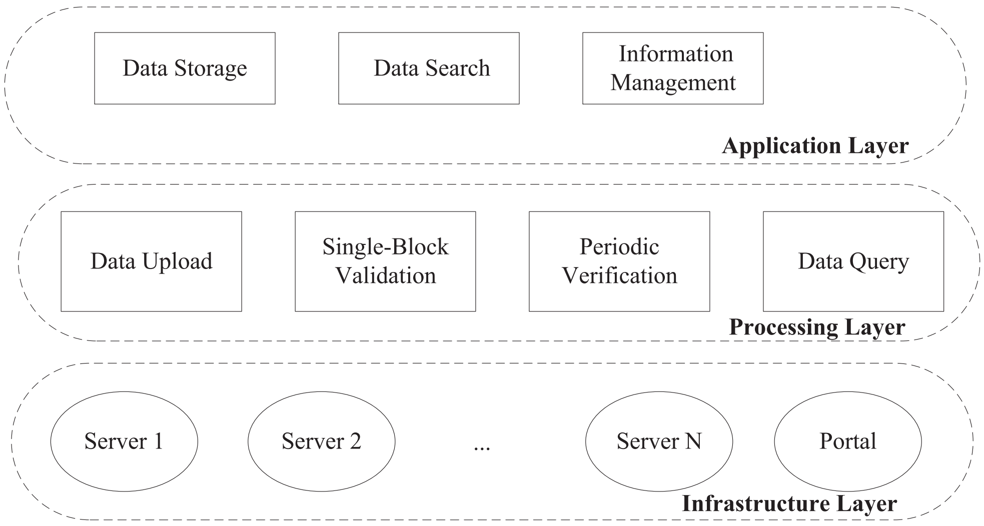


Fig. 1 System architecture

information of a specific block maintained in the blockchain in each server and correcting the data in the block if it is wrong. The periodic blockchain verification module verifies the correctness of the blockchain maintained by each server on a periodic basis. The data query module receives the data search request from the application layer, searches the corresponding block and the data in the blockchain, and returns the query result. Example 1 illustrates how the system is used to achieve data integrity for food production, distribution, and sale.

Example 1 Assume a canning factory uses the proposed framework to store the information of the produced cans. After producing a can, the cannery assigns the can an ID, which is uploaded with the other related information, such as cannery, production date, expiration date, etc., through the application layer of the system. The application layer passes the data storage request to the processing layer, which uses the data upload module to send the data to the infrastructure layer. Every sever in the infrastructure layer generates a block to store the data and appends the block to the blockchain maintained by the server. The can goes through some steps during the logistics and distribution process, and the information about each step is uploaded to the system where a new block is created for the uploaded information. Any modification to the information of the can will create a new block, and the blocks before the modification are kept in the system without being changed. The consumers use the can ID to search the relevant information of the can, such as the manufacturer, production date, expiration date, etc., through the data search function provided by the application layer, which gets the query results from the blockchain maintained by the servers and returns the results to the application layer.

The two-dimension chain data structure shown in Fig. 2 is designed to accelerate the data storage and search. The blockchain is divided into two parts: the horizontal chain and the vertical chain. The horizontal chain is generated from the basic information of the data. When a data is added in the blockchain, we create a new block and append the block to the horizontal chain. Each data has its own vertical chain. When the data is updated, we append a new block containing the updated data to the vertical chain. All the blocks containing the updated information of a data form a vertical blockchain. During the data search, instead of traversing the blocks one after another along with the blockchain, we first find the block for the data on the horizontal chain, and then search the specific data update on the vertical chain. Therefore, the number of blocks traversed can be reduced during the data query.

We propose a Rotating Multiple Random Sampling (RMRS) consensus to randomly select multiple temporary master nodes to handle each data access request so as to reduce the communication cost. K temporary master nodes are randomly selected upon the arrival of each client request. When storing data, the portal randomly selects K servers as temporary master nodes that own the right to generate and broadcast the block copy. The other nodes receive the block copy sent by each temporary master node, compare the K block copies, and keep the copy with the number of duplicated times no less than $\lceil \frac{K}{2} \rceil$. In the data query process, K servers are also randomly selected by the portal as the temporary master nodes. After verifying

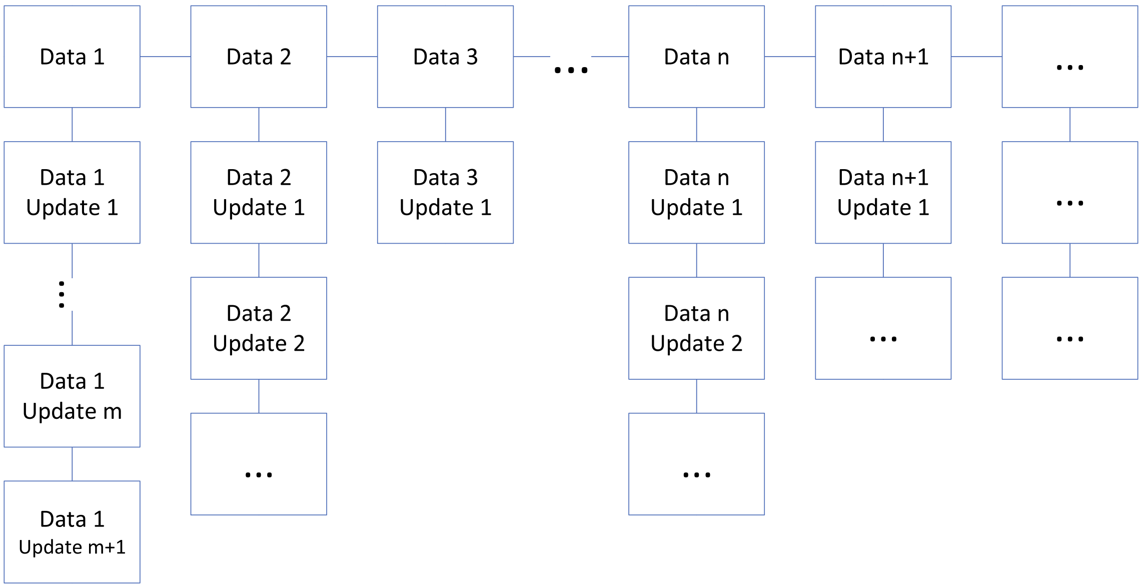


Fig. 2 Two-dimension chain data structure

the queried block, the K temporary master nodes send the block data copies to the client. The client accepts the data copy with the number of duplicated times no less than $\lceil \frac{K}{2} \rceil$. The communication during the consensus process is encrypted using the combination of AES and ECC encryption schemes. The RMRS consensus process is shown in Fig. 3.

Step 1. The user sends a data access request to the portal.
Step 2. The portal randomly selects K temporary master nodes from all the N nodes in the network and sends the addresses of the K temporary master nodes to the user.
Step 3. The user sends specific data access requests to the K temporary master nodes.
Step 4. Each temporary master node broadcasts the data or the requests sent by the client to the other $N-1$ nodes in the network.
Step 5. After receiving K data copies, each node keeps the copy with the most repetitions and sends a response to the K temporary master nodes.
Step 6. Each of the K primary nodes receives $N-1$ responses, takes the data with the most repetitions as the result, and sends a response to the user.

4 Framework modules

The processing layer, the core of framework RMEC, includes the data upload module, the data query module, the single-block validation module, and the periodic blockchain verification module.

4.1 Data upload module

The data upload module receives the data storage request from the application layer, creates a block for the data, and stores the block in the blockchain in each server. After receiving the data storage request, the portal randomly selects $K(K \geq 3)$ servers as the temporary master nodes for the request and returns the addresses of the selected K servers to the client. The client uploads the data to the

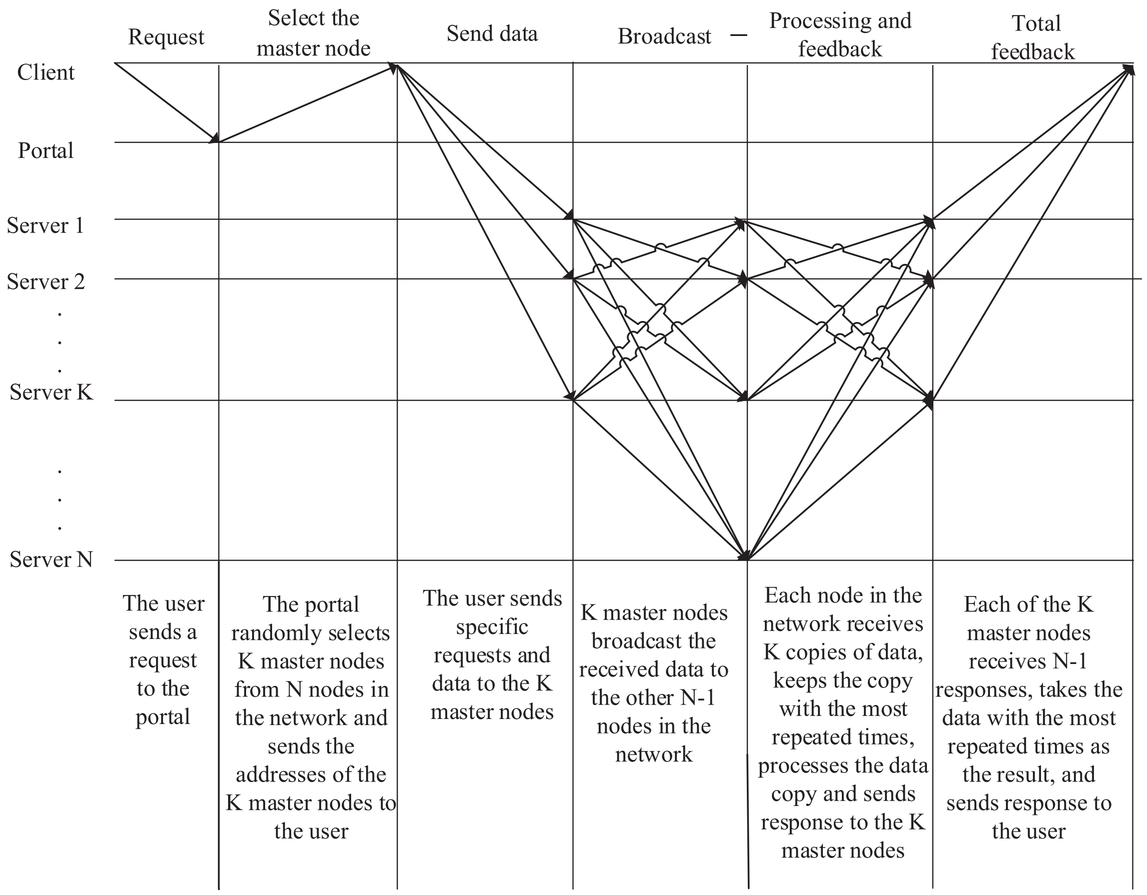

Request
Select the master node
Send data
Broadcast
Processing and feedback
Total feedback
Client
Portal
Server 1
Server 2
Server K
Server N
The user sends a request to the portal
The portal randomly selects K master nodes from N nodes in the network and sends the addresses of the K master nodes to the user
The user sends specific requests and data to the K master nodes
K master nodes broadcast the received data to the other N-1 nodes in the network
Each node in the network receives K copies of data, keeps the copy with the most repeated times, processes the data copy and sends response to the K master nodes
Each of the K master nodes receives N-1 responses, takes the data with the most repeated times as the result, and sends response to the user

Fig. 3 The process of RMRS

K servers which receive the data, generate blocks, and broadcast the hash of the generated block copies to the remaining $N-1$ servers in the network. Each server in the network accepts the hash value with the number of duplicated times no less than $\lceil \frac{K}{2} \rceil$. The server then requests the generated block from one of the servers whose generated block has the same hash value as the accepted one. After verifying the received block and the last block in the local blockchain, the server appends the block to the blockchain the server maintains. The process of data uploading is depicted in Fig. 4.

Step 1. The data upload module sends the client's data storage request to the portal in the infrastructure layer.

Step 2. The portal receives the data storage request, randomly selects K servers in the network as the temporary master nodes, and sends the addresses of the K servers to the client.

Step 3. The client uploads the data to the K selected servers after necessary processing. According to different application requirements, the client's processing of the data may include encryption, generating digital watermarks, etc.

Step 4. A new block is generated for the uploaded data by each of the K selected servers, and the hash of the block is broadcast to the remaining $N-1$ servers.

Step 5. Each server accepts the hash value with the number of duplicated times no less than $\lceil \frac{K}{2} \rceil$.

Step 6. Each server keeps requesting the generated block from one of the servers whose generated block has the same hash value as the accepted one, and calculating the hash value of the received block until it is the same as the previously received hash value.

Step 7. Each server calculates the hash value of the last block in the local blockchain and compares the hash value with the previous hash value in the received block. If they are the same, Step 8 is performed; otherwise, the single-block validation module of the processing layer is executed to correct the last block on the local blockchain.

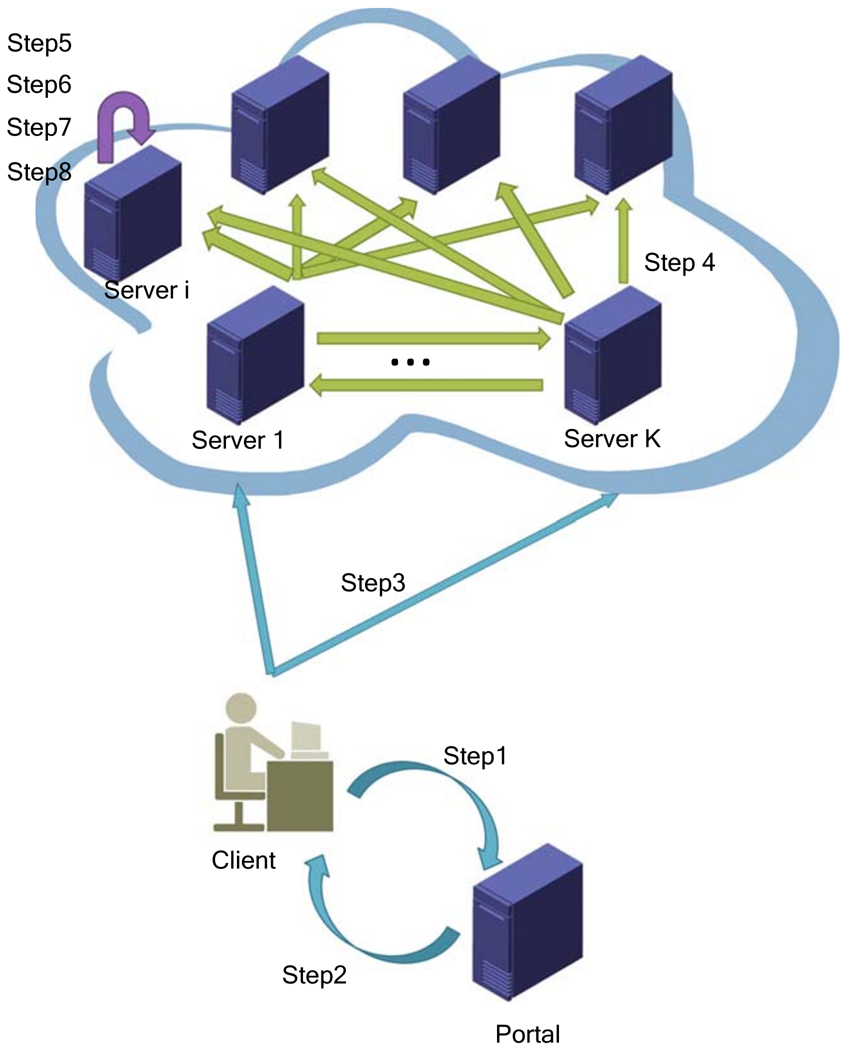


Fig. 4 Data upload module

Step 8. Each server appends the received block to the local blockchain.

4.2 Data query module

After receiving the data search request at the application layer, RMEC calls the data query module to complete the data search. Upon receiving the data search request, the portal randomly selects $K (K \geq 3)$ servers from N servers as the temporary master nodes. Each of the K servers finds the queried block in the server's local blockchain. After verifying the block, each server sends the hash of the block to the client. The client accepts the hash value with the number of duplicated times no less than $\lceil \frac{K}{2} \rceil$ and requests the block from one of the servers whose queried block has the same hash value as the accepted one. The data in the received block is accepted as the query result. The operation of the data query module is illustrated in Fig. 5.

Step 1. The data query module sends the data search request to the portal.

Step 2. The portal randomly selects K servers from N servers as the temporary master nodes and returns the addresses of the K servers to the client.

Step 3. The client sends the search requests to each of the K servers.

Step 4. Each of the K servers finds the block corresponding to the search request and calls the single-block validation module to verify the block.

Step 5. Each of the K servers sends the hash of the corresponding block after verifying the block. According to different application requirements, the processing of data may include encryption, generating digital watermarks, etc.

Step 6. The client accepts the hash value with the number of duplicated times no less than $\lceil \frac{K}{2} \rceil$ as the query result.

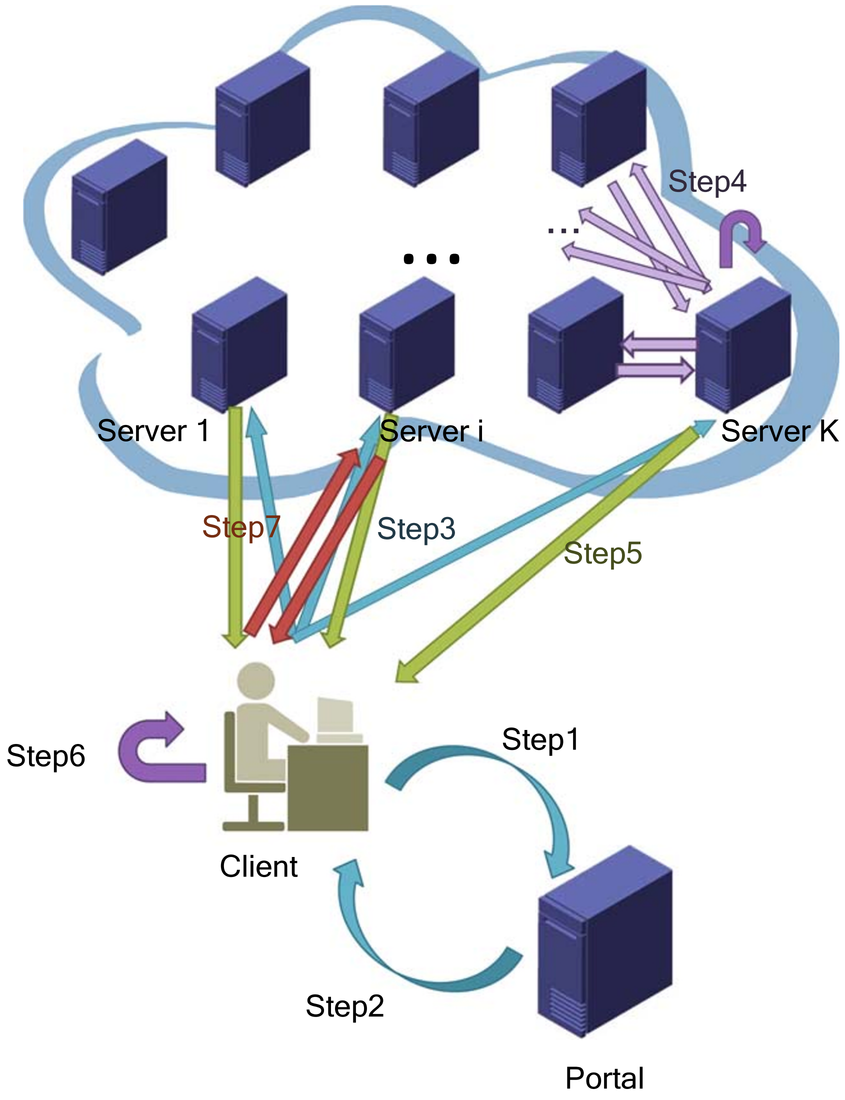


Fig. 5 Data query module

Step 7. The client keeps requesting the block from one of the servers whose block has the same hash value as the accepted one, and calculating the hash value of the received block until the calculated hash value is the same as the previously received one.

4.3 Single-block validation module

The single-block validation module performs the integrity verification of the block in the server and corrects the wrong block. After receiving the call for the single-block validation module, the server sends a request to obtain the hash value of the corresponding block copy from each of the remaining $N-1$ servers and accepts the hash value with the number of repetition times no less than $\lceil \frac{N}{2} \rceil$. If the accepted hash is different from the hash of the corresponding block in the local server, the server requests the block from one of the servers whose block has the same hash value as the accepted one. The received block is used to update the corresponding local block. The operation of the single-block validation module is shown in Fig. 6.

Step 1. Upon receiving a block validation request for a specific block, the server finds the corresponding block in the local blockchain the server maintains.

Step 2. The server broadcasts the block validation request for the specific block to the remaining $N-1$ servers.

Step 3. The server receiving the block validation request returns the hash of the corresponding block in the blockchain maintained by the server.

Step 4. The server sending the validation request accepts the hash value with the number of repetition times no less than $\lceil \frac{N}{2} \rceil$.

Step 5. The server checks whether the accepted hash is the same as the hash of the corresponding local block. If yes, the validation is successful; otherwise, the validation fails, and Step 6 is performed.

Step 6. The server keeps requesting the block from one of the servers whose block has the same hash value as the accepted one, and calculating the hash value of the received block until the calculated hash value is the same as the previously accepted one. The server updates the block in the local blockchain as the received block.

4.4 Periodic blockchain verification module

Each server in the infrastructure layer periodically verifies the integrity of all the blocks in the local blockchain,

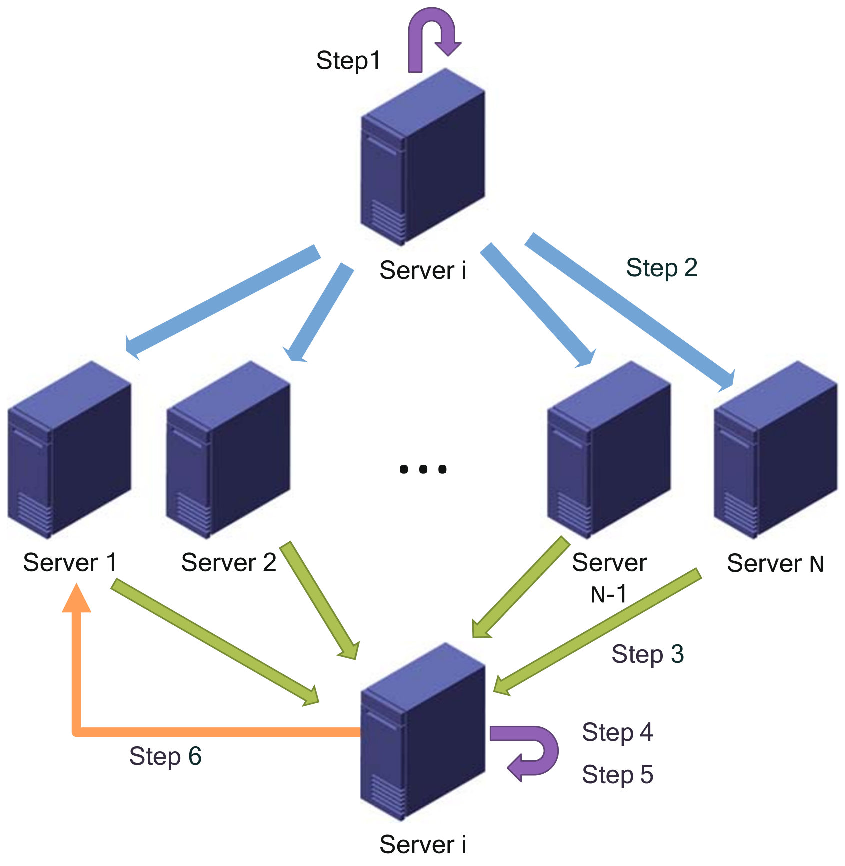


Fig. 6 Single-block validation module

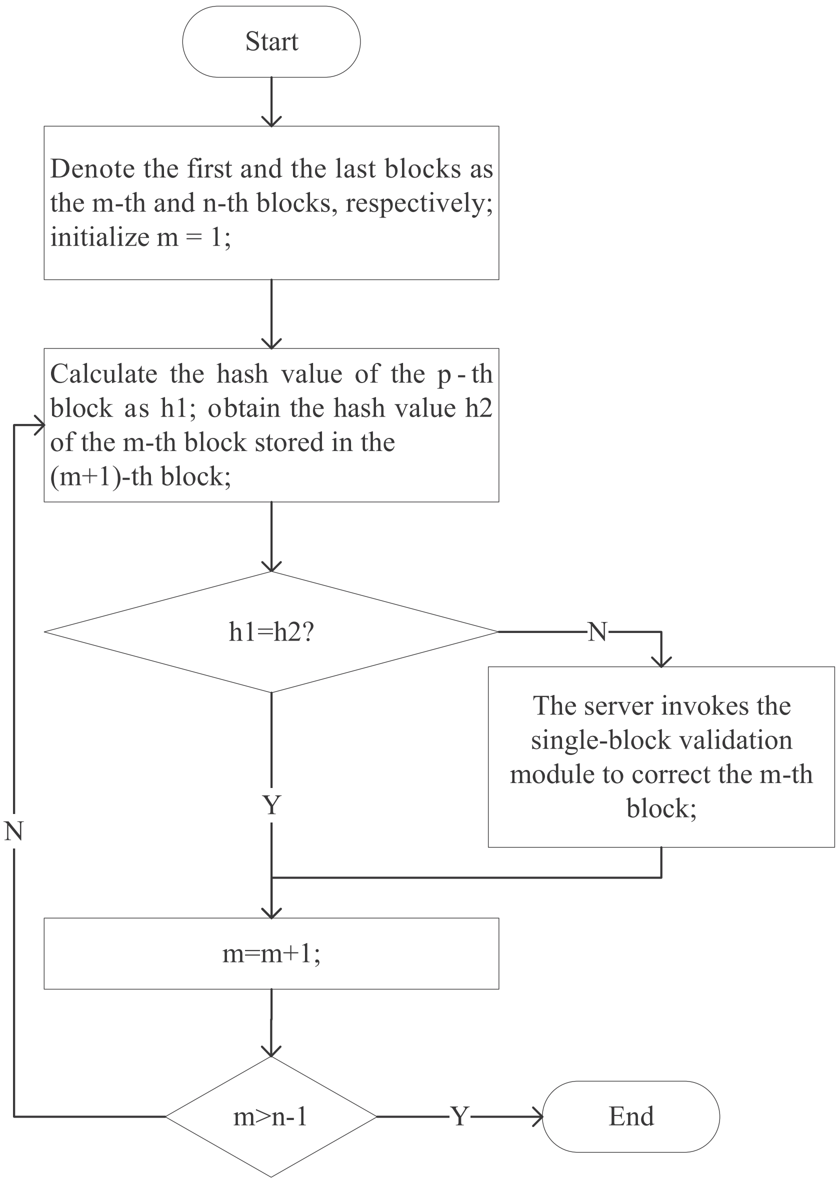

Start
Denote the first and the last blocks as the m-th and n-th blocks, respectively; initialize m = 1;
Calculate the hash value of the p - th block as h1; obtain the hash value h2 of the m-th block stored in the (m+1)-th block;
h1=h2?
N
The server invokes the single-block validation module to correct the m-th block;
Y
N
m=m+1;
m>n-1
Y
End

Fig. 7 Periodic blockchain verification module

starting from the first block. The operation of the periodic blockchain verification module is shown in Fig. 7.

Step 1. The blocks the server maintains are numbered in sequence, starting from 1; denote the last block as the n-th block; initialize $m = 1$.

Step 2. Calculate the hash value of the m-th block as h_1, and compare h_1 with the previous block hash value, h_2, stored in the $(m+1)$-th block; if they are the same, the m-th block is correct; otherwise, the m-th block is wrong and Step 4 is performed.

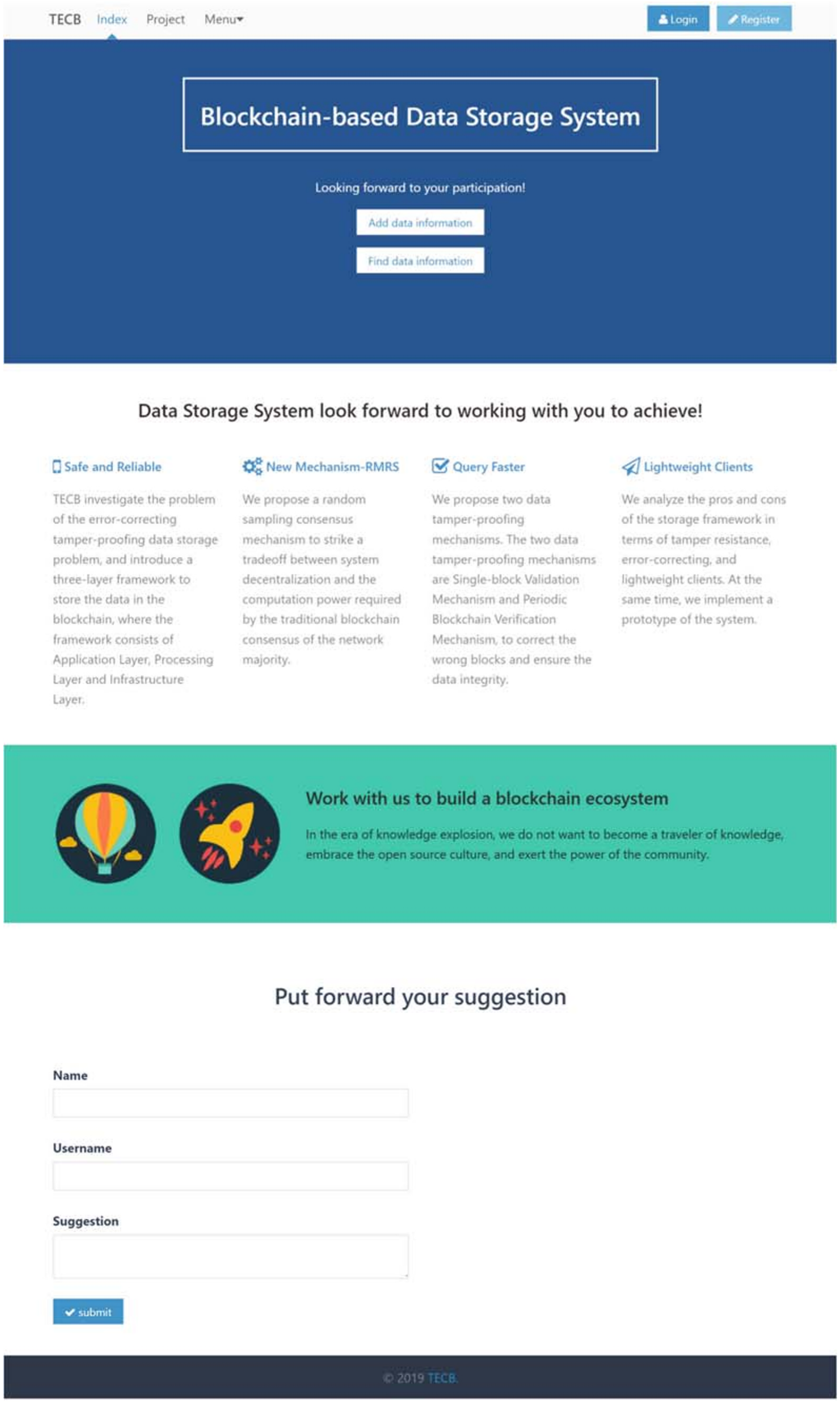
Step 3. $m = m + 1$; if $m > n - 1$, the verification is completed; otherwise, return to Step 2.

Step 4. Invoke the single-block validation module to correct the m-th block, and return to Step 2.

5 Performance analysis and evaluation

5.1 Prototype implementation

We implement a prototype of the proposed framework RMEC as shown in Fig. 8. We use laptops and desktops in LAN to simulate the blockchain servers and the portal in the infrastructure layer. The processing layer is developed based on the combination B/S and C/S modes. In the application layer, we use Java Web to implement the front-end pages. The two-dimension blockchain on each node is implemented via a two-dimension link list. Each block in the horizontal link list maintains the initial main information of a data, and the blocks in the vertical link list contain the subsequent update of the corresponding data. The block



Blockchain-based Data Storage System

Looking forward to your participation!

Add data information

Find data information

Data Storage System look forward to working with you to achieve!

Safe and Reliable

TECB investigate the problem of the error-correcting tamper-proofing data storage problem, and introduce a three-layer framework to store the data in the blockchain, where the framework consists of Application Layer, Processing Layer and Infrastructure Layer.

New Mechanism-RMRS

We propose a random sampling consensus mechanism to strike a tradeoff between system decentralization and the computation power required by the traditional blockchain consensus of the network majority.

Query Faster

We propose two data tamper-proofing mechanisms. The two data tamper-proofing mechanisms are Single-block Validation Mechanism and Periodic Blockchain Verification Mechanism, to correct the wrong blocks and ensure the data integrity.

Lightweight Clients

We analyze the pros and cons of the storage framework in terms of tamper resistance, error-correcting, and lightweight clients. At the same time, we implement a prototype of the system.

Work with us to build a blockchain ecosystem

In the era of knowledge explosion, we do not want to become a traveler of knowledge, embrace the open source culture, and exert the power of the community.

Put forward your suggestion

Name

Username

Suggestion

submit

© 2019 TECB.

Fig. 8 A prototype of the proposed framework RMEC

Table 1 Block information

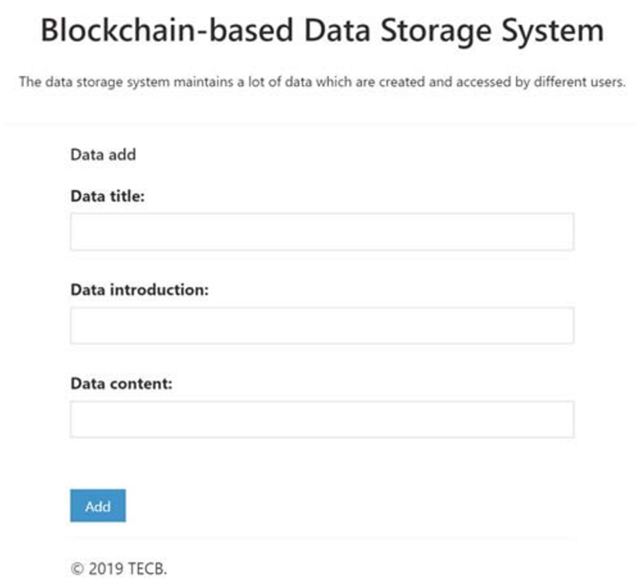
Attributes	Description
Data ID	ID of the data
Data	Data Content
Timestamp	Update time
User ID	ID of the user updating the data
Data hash	Hash of the data
Previous block hash	Hash of the previous block

information includes data ID, timestamp, user ID, the hash of the data, and previous block hash, etc., as depicted in Table 1. The client can upload or update the data through a simple webpage as shown in Fig. 9, which requires little computation and storage capability.

Assume a data has an error in a node in the system, the proposed error-correcting mechanisms can find and correct the error. For example, the data with ID 1 on node 2 has an error and the data is different from the other nodes as shown in Fig. 10. When the user submits a data query request for the data, the data query module in the processing layer will call the single-block validation module to correct the error, and hence the data can be corrected and the client can get the correct data as shown in Fig. 11.

5.2 Data tamper proof performance

The combination of distributed data storage, encryption algorithms, and consensus mechanism technologies makes the data in the blockchain computationally hard to tamper. The proposed data storage framework RMEC provides


Blockchain-based Data Storage System

The data storage system maintains a lot of data which are created and accessed by different users.

Data add

Data title:

Data introduction:

Data content:

© 2019 TECB.

Fig. 9 Data upload or update webpage

tamper-proof during the processes of data uploading, query, and storage.

(1) Data tamper-proof during uploading

Assuming the client sends a data storage request, the portal receives the data storage request and randomly selects $K(K \geq 3)$ servers as the temporary master nodes for the request. The data are uploaded to the K servers. The data are not uploaded through the portal, and hence the attacker cannot acquire or modify the data by attacking the portal. The attacker cannot know the specific servers that need to be attacked in advance since the servers are randomly selected by the portal upon each data storage request. The K servers receive the data, generate blocks, and broadcast the hash of the generated block copies to the remaining $N-1$ servers in the network. If there is a hash value with the number of duplicated times no less than $\lceil \frac{K}{2} \rceil$, each server accepts the copy and keeps requesting the generated block from one of the servers whose created block has the same hash value as the accepted one, and calculating the hash value of the received block, until the calculated hash is the same as the previously received one.

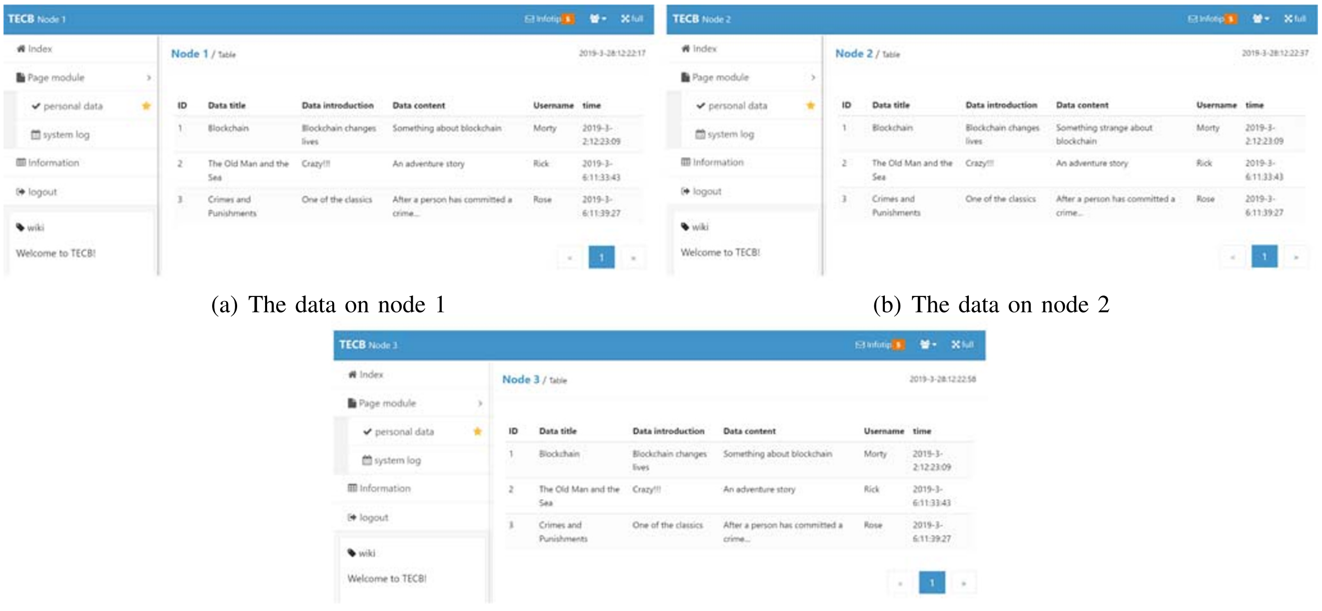
(2) Data tamper-proof during query

During data query, $K(K \geq 3)$ servers are randomly selected as the temporary master nodes, each performing the single-block validation to check the corresponding block and correct it if necessary. In addition, the data are transmitted to the client by the K servers directly without going through the portal, and hence the attacker cannot modify the data by attacking the portal. Before transmitting the data to the client, each of the K temporary master nodes may perform necessary processing, such as encryption, generating a digital signature or digital watermark, etc. At the same time, in the data query module, there is a step of calling the single-block validation module to correct the queried block, so the destroyed or falsified data does not affect the correctness of the information obtained by the user. If and only if at least $\lceil \frac{K}{2} \rceil$ nodes in the master nodes are maliciously controlled, the user will get the wrong data.

(3) Data tamper-proof during storage

The data tamper-proof during storage is guaranteed by the characteristics of the blockchain. That is, once a block is generated and added to the blockchain, no block modification or deletion is allowed. Each block in the blockchain is linked to the previous block via the hash. Therefore, if there is any modification on the m-th block, the hash value of the m-th block will be different from the previous hash value stored in the $(m+1)$-th block.

When the client uploads data, each server calculates the last block hash value and compares it with the hash value of



ID	Data title	Data introduction	Data content	Username	time
1	Blockchain	Blockchain changes lives	Something about blockchain	Morty	2019-3-2:12:23:09
2	The Old Man and the Sea	Crazy!!!	An adventure story	Rick	2019-3-6:11:33:43
3	Crimes and Punishments	One of the classics	After a person has committed a crime...	Rose	2019-3-6:11:39:27

(a) The data on node 1

ID	Data title	Data introduction	Data content	Username	time
1	Blockchain	Blockchain changes lives	Something strange about blockchain	Morty	2019-3-2:12:23:09
2	The Old Man and the Sea	Crazy!!!	An adventure story	Rick	2019-3-6:11:33:43
3	Crimes and Punishments	One of the classics	After a person has committed a crime...	Rose	2019-3-6:11:39:27

(b) The data on node 2

ID	Data title	Data introduction	Data content	Username	time
1	Blockchain	Blockchain changes lives	Something about blockchain	Morty	2019-3-2:12:23:09
2	The Old Man and the Sea	Crazy!!!	An adventure story	Rick	2019-3-6:11:33:43
3	Crimes and Punishments	One of the classics	After a person has committed a crime...	Rose	2019-3-6:11:39:27

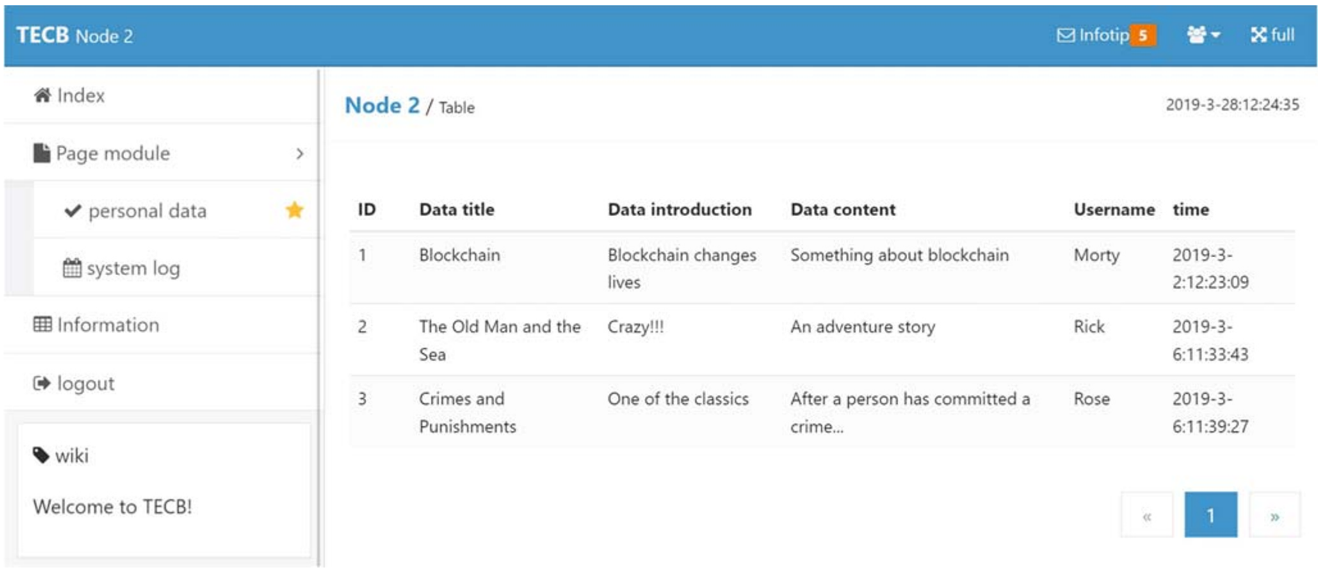
(c) The data on node 3

Fig. 10 An error of Data 1 on node 2

the previous block stored in the accepted block. If the two hash values are the same, the last block in the blockchain is correct; otherwise, the server performs the single-block validation to correct the wrong block.

Assuming that a server node has not successfully stored data or its data has something wrong, the node will synchronize or correct the data via periodic blockchain verification. Assuming that there are N servers in the system and the periodic time interval is T if the number of compromised servers is less than $\lceil \frac{N}{2} \rceil - 1$, the error can be corrected by calling the single-block validation module within time T.

Suppose the probability of each server being compromised is $p (p \ll 1)$, and there are N servers in the system within which there are K temporary master nodes. Assume the total number of compromised nodes is X which follows a binomial distribution, and the number of compromised nodes in the K master nodes is Y. The possibility that the total number of compromised nodes in the current system



ID	Data title	Data introduction	Data content	Username	time
1	Blockchain	Blockchain changes lives	Something about blockchain	Morty	2019-3-2:12:23:09
2	The Old Man and the Sea	Crazy!!!	An adventure story	Rick	2019-3-6:11:33:43
3	Crimes and Punishments	One of the classics	After a person has committed a crime...	Rose	2019-3-6:11:39:27

Fig. 11 The error is corrected upon receiving data query request

is x can be calculated by Eq. (1) and the possibility that the current number of malicious nodes in the K temporary master nodes is y can be computed by Eq. (2).

$$P(X = x) = C_N^x (1 - p)^{N-x} p^x. \tag{1}$$

$$P(Y = y) = \frac{C_x^y \times C_{N-x}^{K-y}}{C_N^K}. \tag{2}$$

Assume the nodes are compromised randomly, with each occurrence being independent of the others and following the same distribution. Only when there are at least $\lceil \frac{K}{2} \rceil$ compromised nodes in the selected K master nodes, an error occurs in the system. The probability of system security can be calculated by Eq. (3). We can guarantee the system security when $K \geq 2pN$. When $K < 2pN$, it is more difficult for the attacker to control at least half of the temporary master nodes with a larger K, and hence a larger K leads to a more secure system than a smaller K.

$$P(Y \geq \lceil \tfrac{K}{2} \rceil) = \sum_{i=0}^{\lceil \frac{K}{2} \rceil} \frac{C_{p\times N}^i \times C_{(1-p)\times N}^{K-i}}{C_N^K}. \tag{3}$$

5.3 Error-correcting mechanisms

Two data error-correcting mechanisms to ensure the integrity of data blocks are proposed: single-block validation and periodic blockchain verification. The single-block validation module is used in several scenarios, where the module detects and corrects the error. During data uploading, each server verifies the last block and correct it if it is wrong. Therefore, the last block is validated each time a new data is stored in the framework so that the correctness of the blockchain is achieved. During the data query, the temporary master nodes perform the single-block validation to check the corresponding block and correct it.

The main function of the periodic blockchain verification module is to periodically detect errors. If there is an error, the server calls the single block verification to fix it. Each server conducts the periodic blockchain verification to validate all the blocks in the local blockchain. When errors are detected, the server performs the single-block validation to correct the block. The periodic block verification enables the data storage framework to have the self-correcting capability and requires no communication overhead during the check, but only needs to calculate and compare the hash values of the local data.

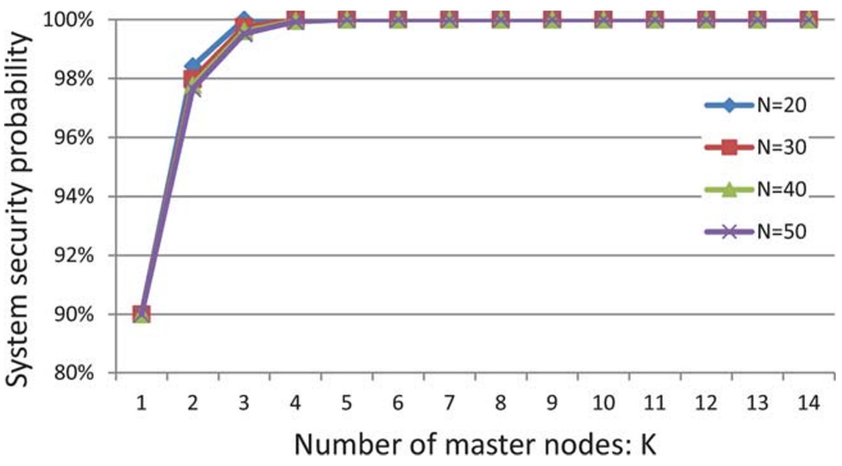
If a node has not successfully stored the data or its data contain an error, the node invokes periodic blockchain verification. Assuming the periodic time interval is T and there are N servers in the system if the number of compromised servers is less than $\lceil \frac{N}{2} \rceil - 1$, the error can be corrected within time T.

5.4 Performance of consensus mechanism

5.4.1 Performance of system security

Figure 12 shows the performance of system security under a different number of temporary master nodes and network nodes when $p = 0.1$, which means that the probability of each node to be compromised is 10%. The system security probability is the probability of the state that the system remains uncompromised. It can be observed that the security of the data storage system increases with the increase in the number of temporary master nodes. A big K indicates that it will be difficult for an attacker to control more than half of the temporary master nodes. When $K \geq 2pN$, the system can achieve 100% security. The probability of the data storage system being compromised is almost 0 when

Fig. 12 System security probability w.r.t different number of temporary master nodes for various number of nodes



$K = 7$ for all N in Fig. 12, which shows that we need only a few numbers of temporary master nodes to ensure the security of the system. A small number of total network nodes also lead to good system security performance. Given p, a small N leads to a small number of compromised nodes, and hence it is difficult to control half of the K randomly selected temporary master nodes.

Figure 13 illustrates the performance of system security in terms of security probability with respect to the number of temporary master nodes and various p (probability of each server being compromised) assuming the number of nodes is 50. In general, a smaller p outperforms a bigger p. A small p indicates that each node in the system is hard to compromise, and hence the system can be more securely protected from the attack. When $p = 0.05$, five random temporary master nodes lead to almost zero probability of the system being compromised. Since $p \ll 1$ in practice, we can achieve system security with a small number of temporary master nodes.

5.4.2 Communication overhead

The proposed framework achieves a good tradeoff between system decentralization and the amount of communication. The data tamper-proof is achieved via the decentralization of blockchain.

In the proposed framework RMEC, we introduce a portal to receive the data access requests from the clients. The portal finds the servers to accommodate the data requests. K temporary master nodes are randomly selected upon each data access request, so there is more than one master node in the system to increase the decentralization performance. The K temporary master nodes are responsible for storing the uploaded data in the blockchains and returning the query data by searching and validating the corresponding block. The data are transmitted from the client to the servers without being forwarded by the portal during the data storage process. The data are sent from the servers to the client without the portal being an intermediate node during the data query process. That is, the data are not handled by the portal. Therefore, by randomly selecting multiple temporary master nodes, we avoid having a node or some specific nodes which have all the rights to operate the data, such as block query and block broadcasting.

We evaluate the proposed consensus RMRS in terms of communication overhead against the state-of-art consensus algorithms of PBFT and YAC. PBFT requires heavy network communication during the pre-prepare, prepare and commit phases since all nodes in the system communicate with one another to achieve system security. (1) Pre-prepare process: Each node broadcasts the data to the entire network. The current master node broadcasts the data to all the nodes. Assume the size of the data is s KB, and the size of the communication overhead, including feedback, signature, timestamp, address and other information, is q KB. The communication cost of the pre-preparing process is $(N-1)(s+q)$ KB. (2) Prepare process: After each node receives the data, the hash of the new block is calculated and broadcast to the entire network. The broadcast of the block hash value consumes less overhead than the broadcast of the block itself. The communication cost of the preparation process is $q(N-1)^2$ KB. (3) Commit process: Each node broadcasts a commit message to the whole network after receiving more than $2f$ same data copies, where f is a

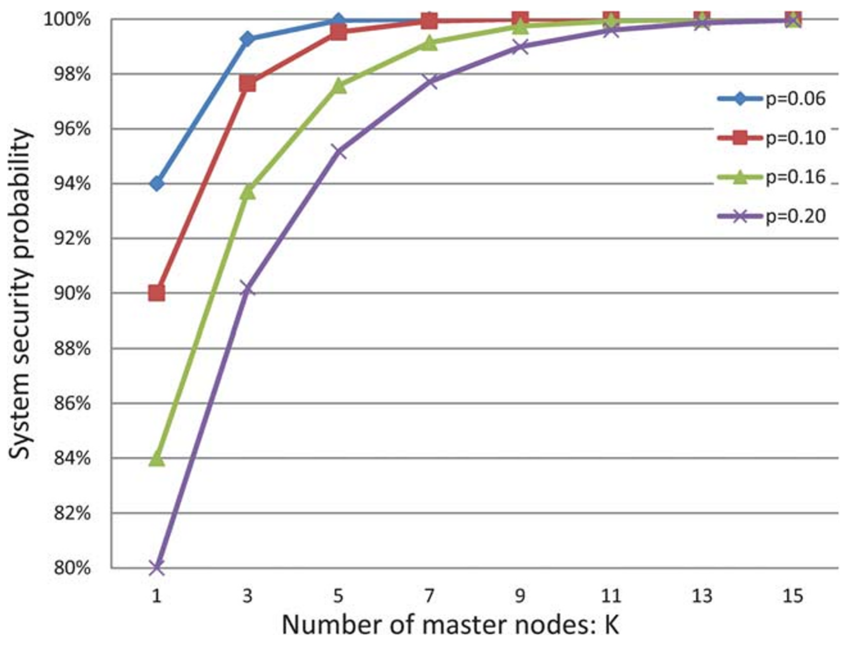
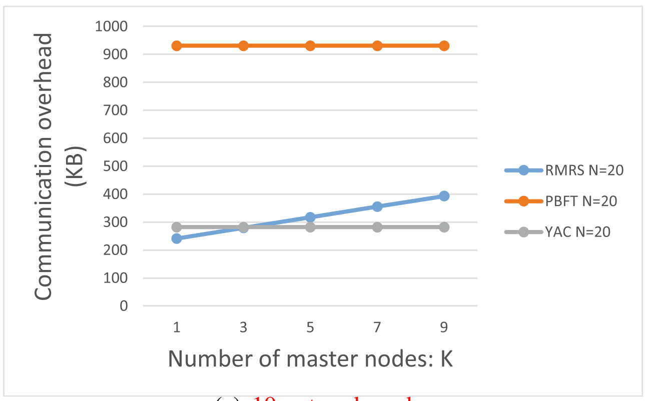


Fig. 13 System security probability w.r.t. different number of temporary master nodes for various p

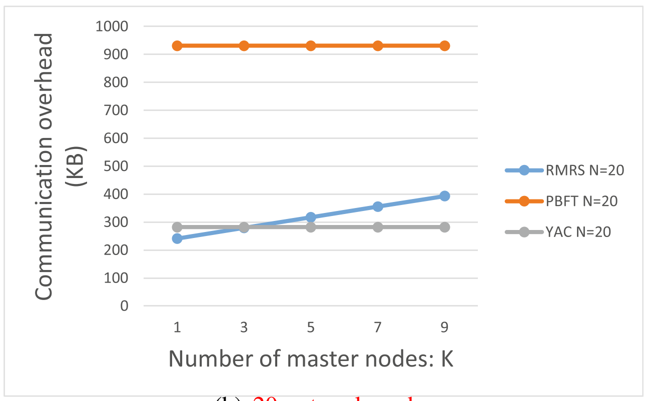
tolerable Byzantine node number. The communication cost of the commit process is $q(N-1)^2$ KB too. Therefore, the total communication overhead of PBFT is $(N-1)(s+q)+q(N-1)^2+q(N-1)^2=2q(N-1)^2+(N-1)(s+q)$ KB.

During the consensus of YAC, the communication cost that the user sends the generated transaction to a peer is $s+q$ KB. The peer consumes $s+q$ KB communication cost to send the transaction to the ordering service. It takes $N(s+q)$ KB communication cost for the ordering service to send the proposal to the peers. The communication consumption of voting across the network and the commit message broadcast is $2Nq$ KB. Therefore, the total communication overhead of YAC is $2(s+q)+N(s+q)+2Nq=(N+2)s+(3N+2)q$ KB.

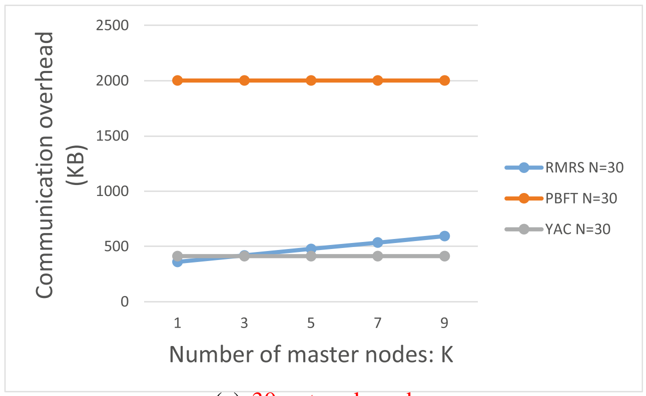
With the proposed consensus algorithm RMRS, K nodes, instead of all the N servers, are randomly selected to participate in the consensus process. Only the hash value of the block is transmitted in the network during the consensus for data uploading, search, and validation. Therefore, the amount of communication is reduced. The communication overhead between the user and the portal is $2q$ KB. The K temporary master nodes receive $K(s+q)$ KB



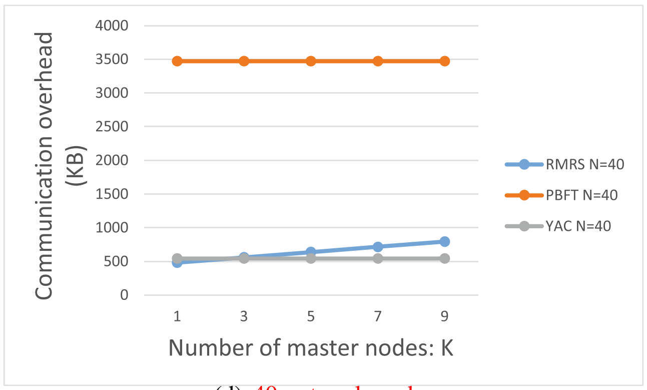
(a) 10 network nodes



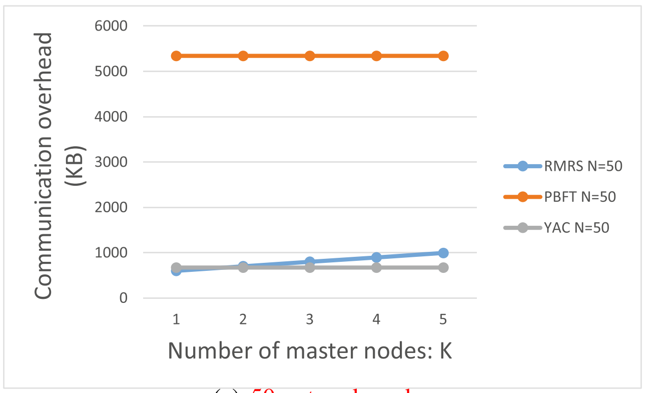
(b) 20 network nodes



(c) 30 network nodes



(d) 40 network nodes



(e) 50 network nodes

Fig. 14 Communication overhead under different number of temporary master nodes

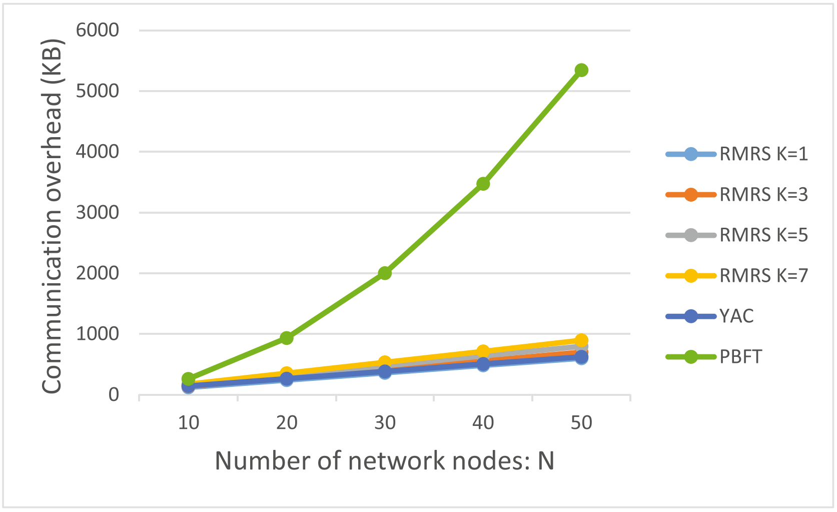


Fig. 15 Communication overhead under a various number of network nodes with a different number of temporary master nodes

data and broadcast the hash of the data with the size of $Kq(N-1)$ KB. In the process of requesting the block, the communication cost is $(s+q)(N-K)$ KB. Therefore, the total communication overhead of RMRS is $2q+K(s+q)+Kq(N-1)+(s+q)(N-K)$ KB.

For example, if $N=5$, $K=3$, $s=2$KB, and $q=1$KB, the communication overhead of PBFT, YAC, and RMRS is 44 KB, 31 KB, and 29 KB, respectively.

Figure 14 depicts the communication overhead of PBFT, YAC, and RMRS, under different numbers of temporary master nodes with various numbers of network nodes, assuming s=10KB and q=1KB. In general, the communication overhead of RMRS increases as the number of temporary master nodes increases. More temporary master nodes result in more broadcast information in the network, which increases the communication overhead. The communication overhead of PBFT and YAC keeps stable with the change in the number of temporary master nodes. PBFT always consumes more communication overhead than RMRS. All the nodes participate in the broadcast process with PBFT, and hence a big number of network nodes result in large communication overhead. When $K=1$, that is, when both RMRS and YAC has only one master node, RMRS outperforms YAC. In most cases when $K \geq 3$, YAC performs better than RMRS since YAC uses one ordering service entity to reduce the communication cost, while RMRS depends on K random master nodes to achieve consensus. However, it is hard to guarantee the security and honesty of the centralized ordering service, and the centralized point is in contradiction with the decentralization rationale of blockchain.

Figure 15 illustrates the communication overhead of PBFT, YAC, and RMRS, under various numbers of network nodes with different numbers of temporary master nodes, assuming s=10KB and q=1KB. In general, the communication overhead of all the three consensus algorithms increases as the number of network nodes increases, since the three consensus mechanisms require more broadcast communication to enable all the nodes to receive the same data with more network nodes. The communication overhead of the proposed consensus algorithm RMRS increases slowly with the increasing network size, which shows that RMRS is scalable. RMRS achieves much better performance than PBFT, since only part of the network nodes broadcast the data in the network, while all the nodes participate in the broadcast process with PBFT. It can also be observed that we need more communication with more temporary master nodes in RMRS since the communication overhead is closely related to the number of temporary master nodes. In most cases when $K \geq 3$, YAC performs better than RMRS. However, when there is only one temporary master node in the network, which is the same as the assumption of YAC having the centralized ordering service, RMRS outperforms YAC. Note that even when $K=1$, the temporary master node is chosen randomly, while the ordering service entity in YAC is fixed in the network.

6 Conclusions and discussion

In this paper, we proposed a three-layer framework to store data in the blockchain and a two-dimensional chain data structure that consists of horizontal and vertical chains. The horizontal chain stores the basic information of the data and maintains a vertical chain for the data. When the data are updated, a new block is generated to store the updated data in the vertical chain. All the blocks containing the updated information of the data form a vertical blockchain. A Rotating Multiple Random Sampling consensus mechanism was devised to balance decentralization and the computation and communication cost. Two error-correcting mechanisms (single-block validation and periodic blockchain verification) were developed to correct the wrong blocks and ensure data integrity. Our experiments demonstrated that the framework could achieve data tamper-proof and effectively reduce the communication cost.

The framework proposed in this paper can ensure the system security with a few numbers of temporary master nodes, assuming that all the nodes have the same probability of being compromised. Note that the number of temporary master nodes depends on both the number of network nodes and the probability of each node being compromised when the probabilities of the nodes to be compromised are different. In the future, we will optimize the number of temporary master nodes and develop a mechanism to adaptively adjust the number of temporary master nodes in different application scenarios to achieve data tamper-proof and efficient system costs. Traditionally, the blocks are generated sequentially due to the consensus process and distributed ledger. The sequential block generation limits the blockchain throughput since all system participants have to wait for the current consensus process to finish so as to start a new consensus for a new block. The proposed consensus mechanism allows different multiple temporary master nodes of different consensus processes to exist simultaneously, and the portal can select the temporary master nodes for multiple concurrent consensus processes. The blocks generated for different vertical chains do not interfere with each other using the proposed two-dimension blockchain structure. Therefore, the proposed consensus mechanism and blockchain structure in the paper make it possible to generate multiple blocks simultaneously, which can increase the system throughput. In our future work, we will discuss how to select the temporary master nodes to enable simultaneous block generation without degrading the system security and data consistency.

Acknowledgments This work was partly supported by the National Natural Science Foundation of China (61701162) and the open project of State Key Laboratory of Complex Electromagnetic Environment Effects on Electronics and Information System (CEMEE2018Z0102B).

References

1. Chao X, Sun Y, Luo H (2017) Secured data storage scheme based on block chain for agricultural products tracking. In: 3rd International conference on big data computing and communications (BIGCOM), Chengdu
2. Nakamoto S Bitcoin: a peer-to-peer electronic cash system. https://bitcoin.org/bitcoin.pdf. Accessed 20 September 2019
3. Zhou G, Zeng P, Yuan X, Chen S, Choo K-KR (2017) An efficient code-based threshold ring signature scheme with a leader-participate model. Security and Communication Networks 2017 Article: 1915239. https://doi.org/10.1155/2017/2041842
4. Yuan X, Gomathisankaran M (2016) Secure medical image processing for mobile devices using cloud services. In: Agaian S, Tang J, Tan J (eds) Mobile imaging for healthcare applications. SPIE Press. https://doi.org/10.1117/3.2204748.ch6
5. Elhoseny M, Elminir H, Riad A, Yuan X (2016) A secure data routing schema for WSN using elliptic curve cryptography and homomorphic encryption. J King Saud Univ - Comput Inform Sci 28(3):262–275
6. Davidson E (2015) Hive mentality or blockchain bloat? Scientist 228(3043):52–52
7. Godsiff P (2015) Bitcoin: bubble or blockchain. Smart Innov Syst Technol 38:191–203
8. Fukumitsu M, Hasegawa S, Iwazaki J, Sakai M, Takahashi D (2017) A proposal of a secure P2P-type storage scheme by using the secret sharing and the blockchain. In: IEEE 31st International conference on advanced information networking and applications, Taipei
9. Sukhodolskiy I, Zapechnikov S (2018) A blockchain-based access control system for cloud storage. In: IEEE Conference of Russian young researchers in electrical and electronic engineering, St. Petersburg
10. Kraft D (2016) Difficulty control for blockchain-based consensus systems. Peer-to-Peer Network Appl 9(2):397–413
11. Chen Y, Li H, Li K, Zhang J (2017) An improved P2P file system scheme based on IPFS and blockchain. In: 2017 IEEE international conference on big data, Boston
12. Tosh DK, Shetty S, Liang X, Kamhoua C, Njilla L (2017) Consensus protocols for blockchain-based data provenance: challenges and opportunities. In: 2017 IEEE 8th annual ubiquitous computing electronics and mobile communication conference, New York
13. Buterin V Ethereum: a next-generation smart contract and decentralized application platform. https://genius.com/Ethereum-ethereum-whitepaper-annotated. Accessed 20 September 2019
14. Delegated proof-of-stake consensus. https://bitshares.org/technology/delegated-proof-of-stake-consensus/. Accessed 20 September 2019
15. Castro M, Liskov B (2002) Practical byzantine fault tolerance and proactive recovery. ACM Trans Comput Syst 20(4):398–461
16. Chen J, Micali S (2017) Algorand. https://arxiv.org/pdf/1607.01341.pdf. Accessed 15 December 2019
17. Muratov F, Lebedev A, Iushkevich N (2018) YAC: BFT consensus algorithm for blockchain. https://arxiv.org/abs/1809.00554. Accessed 15 December 2019
18. Dai M, Zhang S, Wang H, Jin S (2018) A low storage room requirement framework for distributed ledger in blockchain. IEEE Access 6:22970–22975
19. Anglano C, Gaeta R, Grangetto M (2017) Securing coding-based cloud storage against pollution attacks. IEEE Trans Parallel Distrib Syst 28(5):1457–1469
20. Kiskani MK, Sadjadpour H (2017) Secure and private cloud storage systems with random linear fountain codes. arXiv:1706.05604
21. Buttyan L, Czap L, Vajda I (2011) Detection and recovery from pollution attacks in coding-based distributed storage schemes. IEEE Trans Emerg Topics Comput 8(6):824–838
22. Gossip data dissemination protocol. http://hyperledger-fabric.readthedocs.io/en/latest/gossip.html. Accessed 20 September 2019

23. Liu Y, Chen H, Hu F (2017) A blockchain-based verification for sharing data securely. In: 2017 IEEE international conference on big data, Boston
24. Wilkinson S, Lowry J Metadisk a blockchain-based decentralized file storage application. https://storj.io/metadisk.pdf. Accessed 20 September 2019
25. Sun D, Zhao H, Cheng S (2016) A novel membership cloud model-based trust evaluation model for vehicular ad hoc network of T-CPS. Secur Commun Netw 9(18):5710–5723
26. Zhao H, Sun D, Yue H, Zhao M, Cheng S (2017) Using CSTPNs to model traffic control CPS. IET Softw 11(3):116–125
27. Zhao H, Sun D, Yue H, Zhao M, Cheng S (2018) Dynamic trust model for vehicular cyber-physical systems. Int J Netw Secur 20(1):157–167
28. Baza M, Nabil M, Bewermeier N, Fidan K, Mahmoud M, Abdallah M (2019) Detecting sybil attacks using proofs of work and location in VANETs. arXiv:1904.05845
29. Standards for efficient cryptography SEC 2: recommended elliptic curve domain parameters. http://www.secg.org/sec2-v2.pdf. Accessed 20 September 2019
30. Vries AD (2018) Bitcoin's growing energy problem. Joule 2(5):801–805
31. Zyskind G, Nathan O, Pentland AS (2015) Decentralizing privacy: using blockchain to protect personal data. In: IEEE security & privacy workshops. San Jose
32. Baza M, Nabil M, Lasla N, Fidan K, Mahmoud M, Abdallah M (2019) Blockchain-based firmware update scheme tailored for autonomous vehicles. In: IEEE Wireless communications and networking conference (WCNC). Marrakech
33. Baza M, Lasla N, Mahmoud M, Abdallah M (2019) B-ride: ride sharing with privacy-preservation, trust and fair payment atop public blockchain, arXiv:1906.09968
34. Baza M, Nabil M, Ismail M, Mahmoud M, Serpedin E, Rahman M (2018) Blockchain-based charging coordination mechanism for smart grid energy storage units. arXiv:1811.02001
35. Amiri WA, Baza M, Banawan K, Mahmoud M, Alasmary W, Akkaya K (2019) Privacy-preserving smart parking system using blockchain and private information retrieval. arXiv:1904.09703

Yuqi Fan is an associate professor at the School of Computer Science and Information Engineering at Hefei University of Technology, China. He received his Ph.D. in Computer Science and Engineering from Wright State University, Dayton, OH in 2009. He received both B.S. and M.S. degrees in computer science and engineering from Hefei University of Technology in 1999 and 2003, respectively. His research interests include blockchain, computer networks, cloud computing, and cyber-physical systems. Hefei University of Technology, 193 Tunxi Rd., Hefei, 230009.

Xiaohui Yuan received the B.S. degree in electrical engineering from the Hefei University of Technology, China, in 1996, and the Ph.D. degree in computer science from Tulane University in 2004. After his graduation, he was with the National Institutes of Health until 2006. He is currently an Associate Professor with the University of North Texas. His research interests include computer vision, data mining, machine learning, and artificial intelligence. He was a recipient of the Ralph E. Powe Junior Faculty Enhancement Award in 2008 and the Air Force Summer Faculty Fellowship in 2011, 2012, and 2013. University of North Texas, 3940 N. Elm, Denton, TX 76207.