Available online at https://www.cnki.net

October 2021, 28(5): 46 – 58

The Journal of China Universities of Posts and Telecommunications

https://jcupt.bupt.edu.cn

# Multi-level sharded blockchain system for edge computing

Liu Qiao, Tang Bihua (✉), Chen Xue, Wu Fan, Fan Wenhao

School of Electronic Engineering, Beijing University of Posts and Telecommunications, Beijing 100876, China

**Abstract**

Blockchain technology is used in edge computing (EC) systems to solve the security problems caused by single point of failure (SPOF) due to data loss, task execution failure, or control by malicious nodes. However, the disadvantage of blockchain is high latency, which contradicts the strict latency requirements of EC services. The existing single-level sharded blockchain system (SLSBS) cannot provide different quality of service for different tasks. To solve these problems, a multi-level sharded blockchain system (MLSBS) based on genetic algorithm (GA) is proposed. The shards are classified according to the delay of the service, and the parameters such as the shard size of different shards are different. Using the GA, the MLSBS obtains the optimal resource allocation strategy that achieves maximum security. Simulation results show that the proposed scheme outperforms SLSBS.

**Keywords** edge computing (EC), blockchain, sharding, security, latency

## 1 Introduction

EC servers provide computing and storage resources for terminals with limited capabilities and process terminal offloading tasks with low latency, which is essential for compute-intensive and delay-sensitive applications. Compared with centralized cloud computing, EC systems improve availability and reduce latency by outsourcing data processing to the edge of the network, making data processing closer to users [1]. However, the EC system has security problems [2]. If the EC server goes offline, users will not be able to access the data stored on the server, and offloaded tasks cannot be processed. Besides, the EC server may be attacked, the data stored in it may be modified or abused by attackers, and the computational tasks offloaded by the terminal may be executed incorrectly or denied, so the reliability of the computing task result cannot be guaranteed.

To address the above issues, blockchain is widely considered as a promising solution that can build a secure and efficient environment for data storing, processing, and sharing [3]. Blockchain is essentially a distributed ledger shared on the whole blockchain system [4], and can provide data integrity, availability, and reliability, which enable various valuable Internet of things (IoT) applications, such as, Internet of vehicles, content delivery, etc. [5 – 6]. Although blockchain technology brings significant benefits, it is challenging for blockchain systems to provide the necessary scalability to meet the high throughput and low latency requirements of EC systems. The poor scalability of blockchain became an obstacle to its application in EC systems [7]. Throughput and latency are considered to be important indicators for measuring the scalability of the blockchain. The scalability of the existing public

Received date: 21-06-2021

Corresponding author: Tang Bihua, E-mail: bhtang@bupt.edu.cn

DOI: 10.19682/j.cnki.1005-8885.2021.0031

blockchain is still insufficient to deal with high-frequency transactions and delay-sensitive scenarios. Bitcoin generates a block every ten minutes, and the transaction throughput is 7 transactions per second (tx/s), and the throughput of Ethereum is 15 tx/s. Many methods were proposed to improve scalability, which can be divided into two categories. One is the on-chain scaling solution, with improvements in the data layer [8], consensus layer [9], and network layer [10 – 15]. The other is the off-chain scaling solution that aims at reducing the redundancy of the main chain, such as Lightning Network [16], Plasma [17], etc.

Sharding is considered one of the most effective horizontal scaling solutions so far. By dividing the nodes into different groups, called shards, all peer nodes in a group undertake three types of resources of shard-communication, data storage, and computing, while each participating node undertakes all overheads in the traditional non-sharded blockchain. Each shard will be working on a different set of transactions, rather than the entire network processing the same transactions, thus reducing redundancy. Multiple shards working in parallel increase transaction throughput and reduce latency, yet potentially compromise the security. Many studies were carried out to apply sharding technology to blockchain EC systems to improve throughput and security [18 – 20]. In the case of sharding, scalability can be increased, but security may be compromised due to a single shard takeover attack, where the percentage of malicious nodes in a shard exceeds the maximum allowable percentage. Besides, tasks offloaded to EC servers are sensitive to delay and have different delay requirements, so the existing single-level sharding scheme is not applicable. Therefore, while meeting the delay requirements of different tasks, it is essential to find a suitable solution to improve the safety of the offloaded tasks as much as possible.

To solve the above problem, MLSBS was proposed to optimize security while meeting the various delay requirements of offloaded tasks. The main contributions of this paper are described as following.

1) A MLSBS supporting EC is proposed. For tasks with different delay requirements in the IoT, the MLSBS provides multiple levels of the shards to serve tasks with different delay thresholds. Simulation results show that the MLSBS outperforms the existing single-level sharding scheme.

2) Latency and security of the MLSBS were mathematically analysed. The queuing model is built to get the latency of the task being processed by the system. Simulation results show that there is a tradeoff between latency and security.

3) Security of the MLSBS was compared with GA and allocating resources based on traffic proportions. Simulation results show that the GA can achieve higher security.

The remaining parts of this paper are organized as follows. Sect. 2 describes the MLSBS for EC. A performance analysis of the MLSBS is given in Sect. 3. Sect. 4 describes the GA used to allocate the EC servers to each level of shards. Sect. 5 compares the performance of the MLSBS and SLSBS. Moreover, Sect. 5 compares the performance of resource allocation scheme, based on GA and that based on the proportion of traffic. Finally, conclusions are drawn in Sect. 6.

## 2 System model

In this section, the structure of the MLSBS supporting EC is introduced.

### 2.1 MLSBS supporting EC

Based on the three-tier architecture of the EC system [21], MLSBS supporting EC is proposed, as shown in Fig. 1.

In Fig. 1, the structure is divided into three layers. The lowest layer is composed of the terminals of various application sectors (e. g., smart factory, smart home, smart vehicle, etc.). Tasks generated by the terminals are offloaded to edge nodes for processing.

The middle layer consists of base stations (BS) equipped with EC servers. To efficiently handle significantly large amounts of tasks offloaded from the IoT devices, a sharded blockchain is used to process numerous tasks in parallel manner. Different tasks have various delay requirements, so edge nodes are clustered into different shard groups and divided into multiple levels with service rates as classification standards. The shard parameters of different levels (such as shard size, block size, etc.) are different to provide services for tasks with different delay thresholds. Each shard independently creates blocks and verifies the blocks through intra-shard consensus. The tasks that are offloaded to the EC server are allocated to the corresponding shards according to the delay requirements of the tasks. The computational tasks are processed and packaged with the results into blocks. The new block is broadcast to the shard's internal network, and after the nodes reach a consensus, it is linked to the sharded blockchain.

The upper layer is composed of cloud servers. The processed tasks at the EC server layer are uploaded to the cloud servers for backup to ensure that all data are available when the dynamic sharding scheme is used. The shards should be reshuffled every epoch to prevent malicious validators from participating in a particular shard for a long period, which may damage the security of the shard.

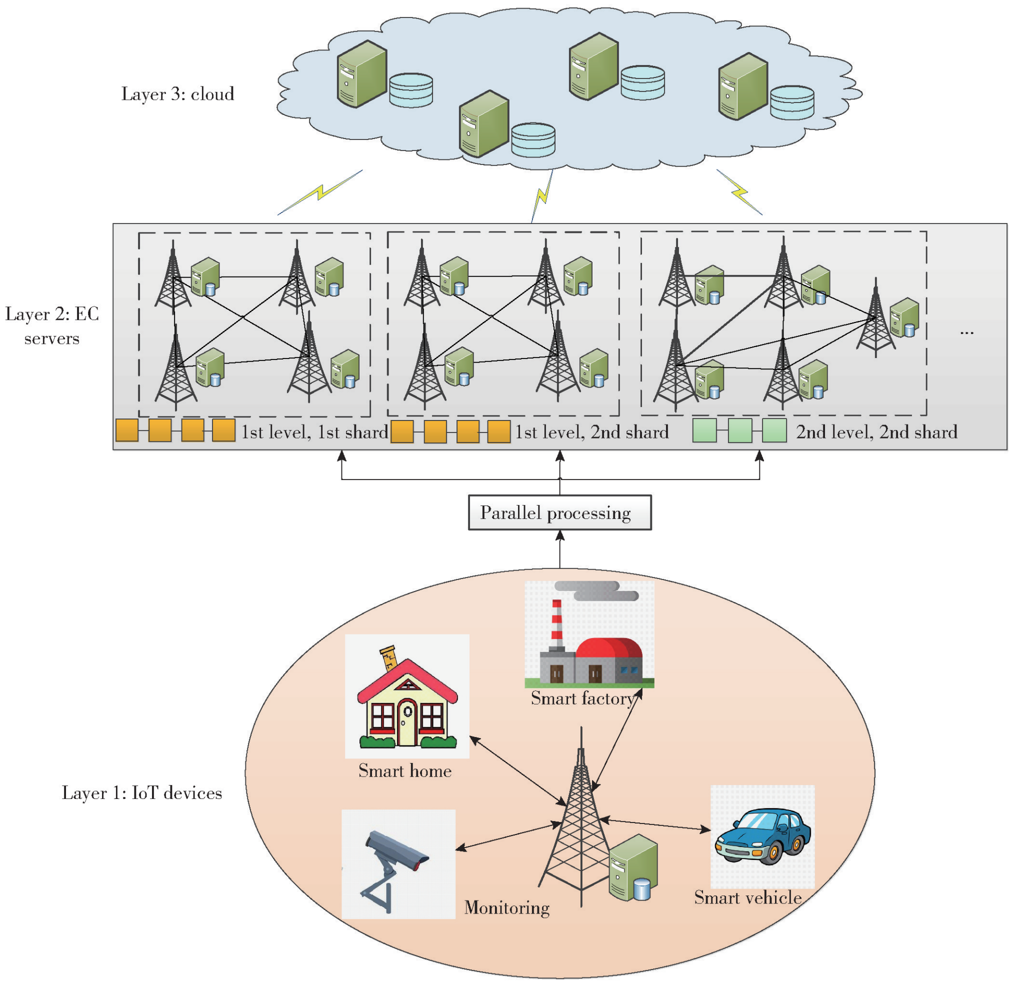

**Fig. 1** MLSBS for EC

## 2.2 Identity setup and shard clustering

All EC servers act as peers in the blockchain network. All peer nodes should be clustered into different shard groups and responsible for serving a disjoint subset of tasks. Compared with the non-sharded blockchain, the number of peer nodes in each shard is reduced, reducing the attacker's attack cost and increasing the probability of successful attack. Moreover, if the result of shard clustering is predictable, the malicious nodes will collude to initiate a single shard takeover attack, which degrades the security of the blockchain. Therefore, a shard clustering architecture is adopted in this paper to allocate EC servers to shards in a random way.

Firstly, each peer node generates its identity (ID) by using a verifiable random function (VRF). VRF is an encryption scheme that maps the input to a verifiable pseudo-random output, which is unpredictable and verifiable. Each edge node takes the public random number of the whole network and its private key as the input of VRF, and obtains a random value as its ID and a proof. With zero-knowledge proof, the legitimacy of the node ID can be guaranteed through the proof of the node's public key and its VRF output, that is, it can be proved that the random value is generated by the node without knowing the node's private key.

Secondly, the shard number of each node is obtained according to the ID set of all nodes and the number of shards required. Each node knows its shard number after the ID computations, and nodes of the same shard should set up a point-to-point connection. The shard clustering process is shown in Fig. 2.

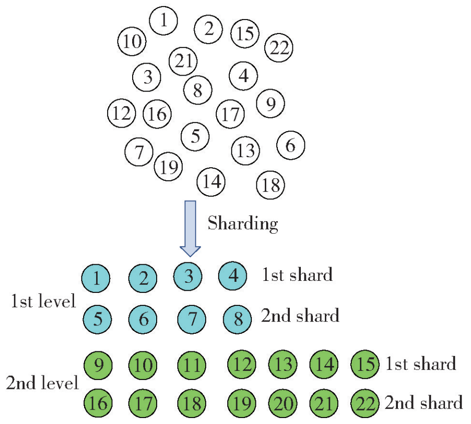

**Fig. 2** The shard clustering process

## 2.3 Intra-shard consensus

After the shard clustering, each node receives its task pool. Tasks are evenly distributed to the corresponding level of shards according to the latency requirements. Each shard processes tasks independently, creates shard blocks and performs a local consensus process. The internal structure of a single shard is shown in Fig. 3.

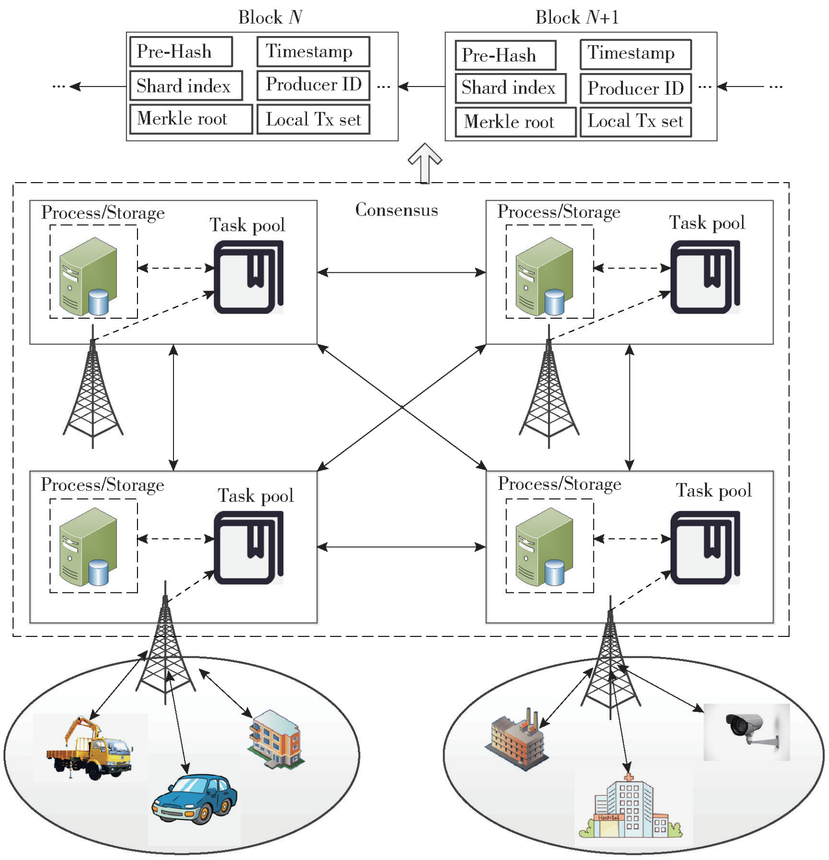

**Fig. 3** Structure of a single shard

The existing consensus protocols are mainly divided into two categories, competition-based consensus algorithms, and Byzantine fault tolerance (BFT)-based consensus algorithms. Competition-based consensus algorithms cannot guarantee the finality of a new block and may fork the blockchain. Multiple forks must be temporarily stored until the node determines which fork is valid ultimately. The competition-based consensus protocol chooses a longer chain to preserve the vitality of the blockchain. The BFT-based consensus protocol will exchange data between a small group of authenticated nodes (called replica nodes) before validating a new block and making a final decision, and designate a node to generate the new block to ensure consistency. Typical is the practical BFT (PBFT) consensus algorithm, widely used in many blockchain systems. The PBFT consensus algorithm can include at most $F$ malicious nodes in $N$ nodes based on the relationship of $(3F+1)\leqslant N$ to ensure the security of the consensus process [22]. Compared with the competition-based mechanism, the PBFT consensus algorithm has the advantages of non-fork, low resource consumption, and high throughput.

Because the consensus process is the main reason for the increase in latency of the blockchain-enabled EC system compared with the existing EC system. Therefore, under the premise of ensuring certain security, each shard executes the PBFT consensus process to reduce delay in the MLSBS. Intra-shard consensus process [23] is shown in Fig. 4.

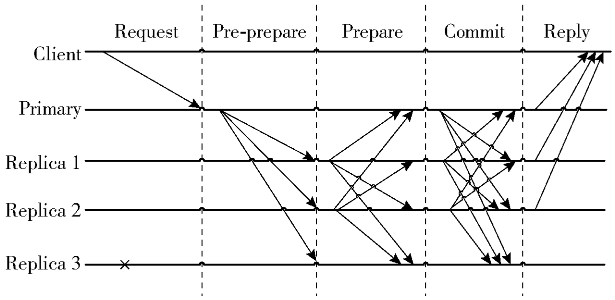

**Fig. 4** Intra-shard consensus process

## 3 Performance analysis

The latency and security of the MLSBS are analyzed in this section. Task delay refers to the time from when the terminal generates a task to when the edge network processes the task and returns the result to the terminal. A secure task refers to the fact that the processing result of a task can reach a consensus among the honest nodes in the shard. Therefore, this paper defines the security of a task as the probability that the task is assigned to a secure shard.

### 3.1 Security

A shard is unsafe which means that the number of malicious nodes in the shard exceeds the malicious nodes limit. The hypergeometric distribution proposed in Ref. [24] described the number of malicious nodes in each shard to obtain its probability distribution. Let $X_i$ denote the number of malicious nodes in shard $i$ and $P(X_i=f_i)$ denote the probability that shard $i$ contains $f_i$ malicious nodes.

Suppose the total number of EC servers in the whole network is $N$, the ratio of malicious nodes is $a$, that is, the total number of malicious nodes is $F=aN$. MLSBS split $N$ nodes into $K$ shards, and shard $i$ is allocated $n_{s,i}$ nodes. In one epoch, if some edge nodes are not assigned to a certain shard, they will not participate in the consensus process of any shard in this period, and can only have the opportunity to join the shard when this epoch is over and the shard is reorganized and start the next epoch.

The distribution of the number of malicious nodes in the first shard can be modelled by the hypergeometric distribution with the parameters, $N$, $F$ and $n_{s,1}$.

$$P(X_1=f_1)=h(N,F,n_{s,1},f_1)=\frac{C_{N-F}^{n_{s,1}-f_1}C_F^{f_1}}{C_N^{n_{s,1}}} \tag{1}$$

According to the law of total probability, the distribution of the number of malicious nodes in the second shard can be expressed as

$$P(X_2=f_2)=\sum_{f_1=0}^{f_1=\min\{n_{s,1},F\}}[P(X_1=f_1)\cdot P(X_2=f_2|X_1=f_1)]=\sum_{f_1=0}^{f_1=\min\{n_{s,1},F\}}[P(X_1=f_1)\cdot h(N-n_{s,1},F-f_1,n_{s,2},f_2)]=\sum_{f_1=0}^{f_1=\min\{n_{s,1},F\}}\left[\frac{C_{N-F}^{n_{s,1}-f_1}C_F^{f_1}}{C_N^{n_{s,1}}}\frac{C_{N-(F-f_1)}^{n_{s,2}-f_2}C_{F-f_1}^{f_2}}{C_{N-n_1}^{n_{s,2}}}\right] \tag{2}$$

Similarly, the distribution of the number of malicious nodes in shard $i$ can be expressed as

$$P(X_i = f_i) = \sum_{f_1=0}^{f_1=\min\{n_{s,1},F\}} \cdots \sum_{f_{i-1}=0}^{f_{k-1}=\min\{n_{s,k-1},F\}} \Big[ h(N,F, n_{s,1}, f_1) \ldots h\Big(N - \sum_{1}^{i-1} n_{s,i}, F - \sum_{1}^{i-1} f_i, n_{s,i}, f_i\Big)\Big] \quad (3)$$

A task is considered unsafe when it is offloaded to an unsafe shard. Since tasks are randomly assigned to shards that meet latency requirements, the security of tasks is closely related to the proportion of secure shards in the system.

There are $L$ delay thresholds for tasks offloaded in the EC system, and the delay level $l$ contains $H_l$ shards. Let $Y_l$ denote the random variable corresponding to the number of safe shards in the delay level $l$ and $P(Y_l = h_l)$ denote the probability that the subsystem $l$ contains $h_l$ safe shards. To consider all possible outcomes, the joint hypergeometric distribution is considered, which is expressed as

$$P(Y_l = h_l) = \sum \cdots \sum P\left(X_{i_1} \leqslant \left\lfloor \frac{n_{i_1} - 1}{3} \right\rfloor, \ldots, X_{i_2} \leqslant \left\lfloor \frac{n_{i_2} - 1}{3} \right\rfloor, \ldots \right); \quad l \in \{1,2,\ldots,L\}, \quad i_1 \neq i_2, i_1, i_2 \in \{1,2,\ldots,H_l\} \quad (4)$$

where $X_{i_1}$ and $X_{i_2}$ are the number of malicious nodes in the shard $i_1$ and $i_2$, respectively. The right side of Eq. (4) represents the probability that there are $H_l$ secure shards in level $l$ after all nodes are randomly sharded.

Finally, the success probability of tasks in the delay level $l$ can be expressed as

$$P_{s,l} = \sum_{h_l=0}^{H_l} \left[ \frac{h_l}{H_l} P(Y_l = h_l) \right] \quad (5)$$

## 3.2 Latency

The task delay is the time required from the task generation to the sharded blockchain returning the result to the terminal. The total time consumption $D_t$ for completing a task includes transmitting the offloading request and the response $D_{Tx}$, processing tasks including consensus and packaging $D_p$. The total delay for a task is

$$D_t = D_{Tx} + D_p \quad (6)$$

1) The time consumption for transmitting the offloading request and response. The total time consumed by transmitting the offloading request with data of size $b$ from the terminal to the BS and the response with data of size $q$ from the BS to the terminal is expressed as

$$D_{Tx} = \frac{b}{R_{MB}} + \frac{q}{R_{BM}} \quad (7)$$

where $R_{MB}$ denotes the average uplink data transmission rate from the terminal to the BS, and $R_{BM}$ denotes the average downlink data transmission rate from the BS to the terminal.

2) The time consumption for processing tasks. Based on the blockchain queuing model, the process includes two stages: packing it into a new block $D_{PK}$ and executing the consensus procedure $D_c$.

$$D_p = D_{pk} + D_c \quad (8)$$

The primary node validates the received offloading request and calculates the task, packs the task, and results as a transaction to create a new block, which is then broadcast to other peers in the shard for consensus. The blocks are linked together in a first-come, first-served (FCFS) order to form the shard chain.

Queueing model with two queueing systems is shown in Fig. 5. Each shard is responsible for the tasks generated by $m$ terminals. The tasks generated by the $m'$th terminal follow Poisson distribution with rate $\lambda_m$, and the task generation process for terminals is mutually independent. According to the synthesis of Poisson distribution, the synthesis of $m$ Poisson distribution processes is still Poisson distribution, and its rate is $\lambda_{pk} = \lambda_1 + \lambda_2 + \ldots + \lambda_{m'} + \ldots \lambda_m$.

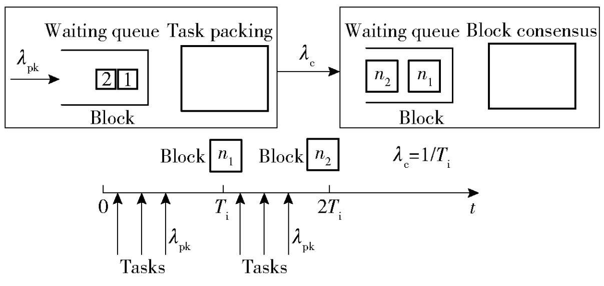

**Fig. 5** Queueing model for tasks processed in a blockchain network

When a task begins to be executed and packed at the primary node, it must wait in a queue, which includes all pending tasks that arrived before it. According to queueing theory [25], the procedure was modelled on an M/M/1 queueing system.

When the offloading requests are received, the primary node verifies the signature of each request and executes the tasks. Assuming that this procedure follows exponential distribution with rate $\mu_1$, the time consumed for the primary node to process and pack a task is

$$\frac{1}{\mu_1}=\frac{\beta+\theta}{c} \tag{9}$$

where, $\theta$ and $\beta$ denote central processing unit (CPU) cycles consumed by verifying the signature and executing the tasks. $c$ denotes the computing power of EC servers.

Therefore, the time consumed by executing and packing a task is

$$D_{\mathrm{pk}}=\frac{1}{\mu_1-\lambda_{\mathrm{pk}}} \tag{10}$$

During the consensus process to verify a new block, the block producer creates a block, for each block interval period $T_{\mathrm{i}}$. Therefore, the block arrives the consensus network with rate $\lambda_{\mathrm{c}}=1/T_{\mathrm{i}}$ and the number of tasks packaging in a block is $M=\lambda_{\mathrm{pk}}T_{\mathrm{i}}$. If the average transaction size is $b$, the block header size is $B_{\mathrm{H}}$, then the size of a block is $B=Mb+B_{\mathrm{H}}$.

The consensus process consists of message propagation, message generation/verification, and task execution for checking the calculation results. Therefore, the total consensus time consists of the following factors.

$$T_{\mathrm{c}}=\frac{1}{\mu_2}=T_{\mathrm{p}}+T_{\mathrm{v}}+T_{\mathrm{e}} \tag{11}$$

where $T_{\mathrm{p}}$, $T_{\mathrm{v}}$ and $T_{\mathrm{e}}$ are the message propagation, message validation, and task execution delay of the intra-shard consensus process, respectively. $T_{\mathrm{p}}$ and $T_{\mathrm{v}}$ refer to the delay analysis method proposed in Ref [24].

**Step 1** Pre-prepare. At the start of the initial consensus in each shard, the primary node packages $M$ tasks into a block and broadcasts the block to the replica nodes. Assuming that a shard contains $n$ peer nodes. After the primary node completes the packing step, $n-1$ messages authentication code (MACs) are generated and the pre-prepare message is sent to the replica nodes.

**Step 2** Prepare. Each replica node receives the pre-prepare message and verifies a single MAC and executes these $M$ tasks contained in the block. If the execution results of the replica node are the same as that in the block, $n-1$ MACs are generated, and a prepare message is generated and sent to other peers.

**Step 3** Commit. All nodes exchange messages to block verification. The primary node receives $n-1$ prepare messages while each replica node receives $n-2$ prepare messages and all nodes process MAC operations of all messages. Only when the node receives the prepare message sent by more than 2/3 of the nodes, it generates $n-1$ MACs, and sends a commit message to other nodes.

**Step 4** Reply. All nodes verify $n-1$ MACs. When receiving a commit message sent by more than 2/3 of the nodes means that most of the nodes in the network have reached a consensus on the block, nodes update their sharded blockchain and send task execution responses to the clients and send the shard block to the cloud server.

In summary, the primary node performs a total of $4(n-1)$ MAC operations while a replica node performs $M$ tasks execution and $4(n-1)$ MAC operations. Then, the time consumption for validating message is

$$T_{\mathrm{v}}=\frac{4(n-1)\alpha}{c} \tag{12}$$

The time consumption for executing tasks is

$$T_{\mathrm{e}}=\frac{M\beta}{c} \tag{13}$$

where $\alpha$ and $\beta$ represent central processing unit (CPU) cycles consumed by task calculation and MAC operation respectively.

Next, the message propagation delay is the time that it takes for a message to reach the destination node during the consensus steps including pre-prepare, prepare and commit. Note that a timeout $\zeta$ is set in each step to prevent unresponsive nodes from delaying the consensus process excessively. Replica nodes that do not respond within the timeout $\zeta$ are considered to have a rejected opinion of the corresponding consensus step. Then, the propagation delay of the request for each intra-shard consensus step is

$$T_{\mathrm{p}}=\min\left\{\max_{n_{\mathrm{d1}}\neq n_{\mathrm{d2}}}\frac{B}{R_{n_{\mathrm{d1}},n_{\mathrm{d2}}}},\zeta\right\}+\min\left\{\max_{n_{\mathrm{d2}}\neq n_{\mathrm{d3}}}\frac{B}{R_{n_{\mathrm{d2}},n_{\mathrm{d3}}}},\zeta\right\}+\min\left\{\max_{n_{\mathrm{d1}}\neq n_{\mathrm{d3}}}\frac{B}{R_{n_{\mathrm{d1}},n_{\mathrm{d3}}}},\zeta\right\}\tag{14}$$

where $n_{\mathrm{d1}}$ and $n_{\mathrm{d2}}$ represent the $n_{\mathrm{d1}}$th and $n_{\mathrm{d2}}$th node, respectively. $R_{n_{\mathrm{d1}},n_{\mathrm{d2}}}$ denotes data transmission rate between node $n_{\mathrm{d1}}$ and $n_{\mathrm{d2}}$.

Finally, the total consensus time of a shard is

$$T_{\mathrm{c}}=\frac{1}{\mu_2}=T_{\mathrm{p}}+T_{\mathrm{v}}+T_{\mathrm{e}}=\min\left\{\max_{n_{\mathrm{d1}}\neq n_{\mathrm{d2}}}\frac{B}{R_{n_{\mathrm{d1}},n_{\mathrm{d2}}}},\zeta\right\}+\min\left\{\max_{n_{\mathrm{d2}}\neq n_{\mathrm{d3}}}\frac{B}{R_{n_{\mathrm{d2}},n_{\mathrm{d3}}}},\zeta\right\}+\min\left\{\max_{n_{\mathrm{d1}}\neq n_{\mathrm{d3}}}\frac{B}{R_{n_{\mathrm{d1}},n_{\mathrm{d3}}}},\zeta\right\}+\frac{4(n-1)\alpha}{c}+\frac{M\beta}{c}\tag{15}$$

Let $A$ denotes the number of blocks that have reached a consensus within a block interval $T_{\mathrm{i}}$. By assuming the time consumed by consensus follows exponential distribution with rate $\mu_2$, the number of completed blocks within a block interval $A$ satisfies the Poisson distribution with rate $\mu_2 T_{\mathrm{i}}$, and can be expressed as

$$s_k=P\{A=k\}=\frac{(\mu_2 T_{\mathrm{i}})^k}{k!}\mathrm{e}^{-\mu_2 T_{\mathrm{i}}}\tag{16}$$

$E_w$ denotes the number of the $k$th blocks that stay in the consensus network when the block producer generates the $w$th block. $E_w$ is taken as the state and a discrete-time Markov chain is established. The state transition occurs when a new block is generated. The probability of $E_w=j$ is related to $s_{n-1}$ and $A$. The relationship is

$$E_w=E_{w-1}+1-A;\quad 0\leqslant A\leqslant E_{w-1},w\geqslant 1\tag{17}$$

The initial state of the consensus subsystem is 0 (i. e. $E_0=0$). At this time, the verification subsystem starts to pack and create new blocks, but there is no block to be processed in the consensus subsystem. When the first block is generated, the state of the consensus subsystem transitions to 1 (that is, $E_1=1$). If the consensus subsystem does not finish processing this block within the second block interval, then when the second block is generated, there are two blocks in the consensus subsystem (i. e., $E_2=2$); otherwise, there is one block in the consensus subsystem (i. e., $E_2=1$). By analogy, the state transition of the discrete-time Markov chain is shown in Fig. 6.

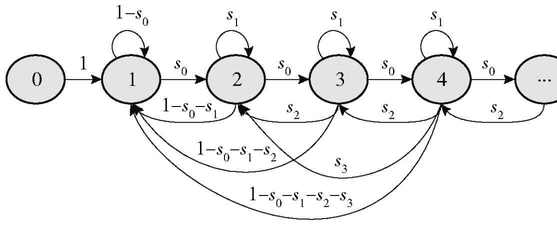

**Fig. 6** State transition of the consensus network

The steady-state equation of the discrete-time Markov chain is

$$\left.\begin{aligned}
&P_w=\sum_{k=w-1}^{\infty}(s_{k-w+1}P_k);\quad w>1\\
&P_1=\sum_{k=1}^{\infty}\left(1-\sum_{w=0}^{k-1}s_w\right)P_k\\
&\sum_{k=0}^{\infty}P_k=1
\end{aligned}\right\}\tag{18}$$

According to the steady-state equation, the stationary distribution of the number of blocks in the consensus network is obtained. Then, the average number of blocks in the waiting queue in the consensus subsystem is obtained as

$$L_{\mathrm{q}}=\sum_{w=1}^{\infty}(w-1)P_w=\sum_{w=1}^{\infty}wP_w-1+P_0\tag{19}$$

According to Little's theorem, the average waiting time can be obtained as

$$T_{\mathrm{aw}}=\frac{L_{\mathrm{q}}}{\lambda_{\mathrm{c}}}=L_{\mathrm{q}}T_{\mathrm{i}}\tag{20}$$

Finally, the average time consumed for a block consensus can be expressed as

$$D_{\mathrm{c}}=T_{\mathrm{aw}}+\frac{1}{\mu_2}=T_{\mathrm{aw}}+T_{\mathrm{c}}\tag{21}$$

## 4 Optimal resource allocation algorithm

### 4.1 Optimization problem

The aim is to maximize the total security of tasks gained by all level shards in the blockchain-enabled EC system. It needs to define an appropriate system revenue function to judge the pros and cons of the resource allocation strategy to comprehensively consider the security of the tasks with multiple delay thresholds in the system. When the total number of EC servers is limited, the higher the frequency of arrival tasks with a certain delay threshold $D_{\text{th}}$, the more EC servers the system should allocate to this level of shards for improving the total security of tasks.

The weighted sum of the security of tasks with multiple delay thresholds is used as the overall optimization goal, and the generation rate of tasks is used as the weight. The task with a high task arrival rate has a larger weight. Objective function is

$$\left.\begin{aligned}&\max_{N} S_{\max} = \sum_{l=1}^{L} \frac{\lambda_l}{\lambda} S_l \\ &\text{s. t.} \\ &\qquad \sum_{l=1}^{L} N_l = \sum_{l=1}^{L} n_l H_l \leqslant N \\ &\qquad t_l \leqslant D_{\text{th}};\quad l = 1,2,\ldots,L\end{aligned}\right\} \tag{22}$$

where $N$ denotes the number of EC servers. $n_l$ and $H_l$ are the size of a shard and the number of shards of the level $l$, respectively. Then, the total resources allocated to level $l$ denoted by $N_l$. Let $\lambda_l$ denote the arrival rate of shard level $l$, and $\lambda$ is the total arrival rate of all tasks with different delay thresholds, namely $\lambda = \sum_{l=1}^{L} \lambda_l$. $S_l$ denotes the security of tasks offloaded to shard level $l$.

The first constraint indicates that the number of EC servers allocated to all levels cannot exceed the total resources available in the whole network. By defining the actual delay of the task offloaded to level $l$ as $t_l$ and the delay threshold of the task as $D_{\text{th}}$, the second constraint restricts the actual delay of tasks processed cannot exceed the maximum latency allowed.

## 4.2 Optimization algorithm using GA

The optimization problem belongs to a mixed-integer optimization problem because the value of $N_l$ is integer and ranges from 0 to $N$. GA is a random search algorithm that solves complex problems by imitating biological evolution [26]. There are multiple levels of the shard and the total resources are limited in the MLSBS. Allocating resources to multiple subsystems to optimize security is a boxing problem in a typical combinatorial optimization problem. GA is adopted in the evolution process of each population, which iteratively searches for the solution of the problem, updates the population, and makes the individuals (called chromosomes) in the population denser around the optimal solution. Chromosomes are potential solutions to the problem, and gradually get improved through selection, crossover, mutation, and fitness evaluation in every iteration. Finally, the optimal solution (or near-global solution) can be found after multiple iterations.

1) Chromosome representation

A chromosome $\boldsymbol{a}_{q'} = (N_1, N_2, \ldots, N_L)$, $q' = 1, 2, \ldots, Q'$, is a sequence, which is serially filled with the number of EC servers allocated to the shard level $l$ denoted by $N_l$. Note that $N_l$ is constrained by the total available resources. The value of $N_l$ is expressed by using $R$ bits, which can be scaled based on the total number of EC servers.

2) Population initialization

A population $\boldsymbol{Q}$ including $Q'$ chromosomes is initialized at the start of GA. The chromosomes are randomly generated, but they must satisfy all constraints.

3) Fitness evaluation

The fitness evaluation aims to evaluate the quality of the chromosomes in the population. The fitness of a chromosome is the value computed from a fitness function. Note that, the higher fitness is, the higher the quality of the chromosome will be.

The fitness function is derived from Eq. (22) of the optimization problem attached by a barrier mechanism to punish the chromosomes which violate any of the constraints. The fitness function is

$$z = \sum_{l=1}^{L} \frac{\lambda_l}{\lambda} S_l - \left\{ \min\left\{0, N - \sum_{l=1}^{L} N_l\right\}\delta + \sum_{l=1}^{L} \min\{0, D_{\text{th}} - t_l\} \right\} \tag{23}$$

where $\delta$ is a very large positive number.

4) Selection

The roulette method is used to promote the average quality of the population. The probability of a chromosome being selected is directly proportional to its fitness score. Similar to that a disk is divided into many parts evenly, the chromosomes with high fitness score occupy more shares on the disk, turn the pointer of the disk, and the individuals finally pointed to will be selected. In this way, the chromosomes with high fitness score have a better chance of being selected from the current population through the selection process.

5) Crossover

Two offspring chromosomes are generated from the crossover of two selected chromosomes. In the process of crossover, GA chooses several locations in the chromosome with a probability $P_c = 0.7$, then exchanges the two bits on the same location of the two chromosomes.

6) Mutation

In order to prevent the solutions represented by the chromosomes from converging into a local optimal point, bits at random locations of some chromosomes are turned over with a probability $P_m = 0.06$ in the process of mutation.

After the crossover and mutation, some solutions may break the limited resources constraint. Rescale the resource allocation solution to satisfy the constraints.

The workflow of the GA is shown in Algorithm 1.

**Algorithm 1** Allocation resources by using GA

```
01 initialize $\boldsymbol{Q}$;
02 carry out fitness evaluation for each chromosome in $\boldsymbol{Q}$ to get
   fitness($\boldsymbol{Q}$);
03 $\text{fitness}^{\text{current}} = \max\{\text{fitness}(\boldsymbol{Q})\}$;
04 loop
05     $\text{fitness}^{\text{last}} = \text{fitness}^{\text{current}}$; // $\text{fitness}^{\text{last}}$ and $\text{fitness}^{\text{current}}$ are
       the last iteration and current iteration variables that store
       the maximum fitness value in $\boldsymbol{Q}$
06     carry out selection using the stochastic tournament
       method;
07     carry out crossover with probability $p_c$;
08     carry out mutation with probability $p_m$, finally get a new
       population $\boldsymbol{Q}'$;
09     carry out fitness evaluation for each chromosome in $\boldsymbol{Q}'$;
10     $\text{fitness}^{\text{current}} = \max\{\text{fitness}(\boldsymbol{Q}')\}$
11         if $|\text{fitness}^{\text{last}} - \text{fitness}^{\text{current}}| / \text{fitness}^{\text{current}} \leqslant \varepsilon$ // the
               gap between $\text{fitness}^{\text{last}}$ and $\text{fitness}^{\text{current}}$ is close
               enough
12             break;
13         end if
14 end loop
15 choose the chromosome which has the maximum fitness
   value in the population, then get optimal benefit directly
   and decode the optimal resource allocation scheme.
```

## 5 Simulation results and analysis

### 5.1 Simulation settings

The settings of parameters used in simulations are shown in Table 1.

**Table 1** Simulation parameters

| Parameter | Value | Parameter | Value |
|---|---|---|---|
| $m$ | [5,200] | $\theta$/MHz | 2 |
| $\lambda_m$/s | [0.1,10] | $\alpha$/MHz | 1 |
| $R_{n_{d1},n_{d2}}$/(MB·s$^{-1}$) | [10,20] | $\beta$/MHz | [10,100] |
| $R_{MB}$/(MB·s$^{-1}$) | [1.5,3.5] | $c$/GHz | [10,30] |
| $R_{MB}$/(MB·s$^{-1}$) | [3.5,5.5] | $b$/kB | [1,10] |
| $N$ | 300 | $q$/kB | [0.2,2] |
| $B_H$/B | 80 | | |

The performance of the MLSBS is compared to the SLSBS in terms of the total security of tasks. The performance analysis is based on the following two benchmarking schemes.

1) SLSBS: the parameters of all shards are the same, and the shards' processing tasks have the same latency.

2) MLSBS with allocating resources proportionally according to the number of tasks with different delay thresholds.

Assume that the EC system contains tasks with a delay threshold of 0.1 s, 0.5 s, and 1.0 s, and the actual value is determined by the specific application. In the SLSBS, all parameters of shards are the same and the service quality is the same. The delay of all shards processing tasks must not exceed the minimum delay threshold (i.e., 0.1 s) to meet the delay requirements of all tasks.

### 5.2 Security analysis

A method of repeated trials is used to estimate the security as shown in Eq. (5). In particular, an array of $F$ malicious nodes and $N-F$ honest nodes were established and distributed to all shards randomly.

Then, the number of malicious nodes in each shard is counted. If the number of malicious nodes in a shard exceeds 1/3, the shard will be unsafe. Otherwise, the shard is safe. Once this procedure is complete, the ratio of safe shards for each level can be obtained. To consider all the possibilities, this trial is repeated a large number of times. After repeating this procedure, the estimated expectation of the proportion of safe shards for each level can be obtained.

Fig. 7 demonstrates the convergence of Algorithm 1 in solving the resource allocation problem with maximum system security denoted by $S_{\max}$ of the tasks. In Fig. 7, the total number of edge nodes is 300, each BS covers 50 terminals and the ratio of malicious nodes is 10%. The simulation results show that with the increase of iterations, the resource allocation scheme is closer to the scheme with higher $S_{\max}$, finally, the maximum value $S_{\max}$ is 0.991 5.

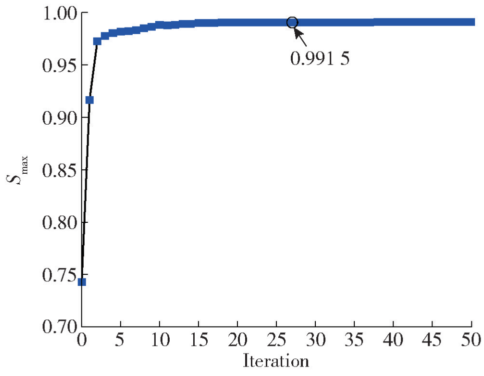

**Fig. 7** Security and convergence trend analysis

1) Impact of the ratio of the malicious nodes on security.

Fig. 8 shows $S_{\max}$ changed when the malicious nodes are injected into the blockchain validators. The task generation rate of the three delay thresholds is the same. As the ratio of the malicious nodes in the system increases, the probability of malicious nodes being selected increases, resulting in a failed shard that causes the security to drop. On the other hand, the MLSBS is superior to the SLSBS in terms of security, and as the proportion of malicious nodes increases, the advantages of MLSBS become more obvious. When the proportion of malicious nodes is 10%, the $S_{\max}$ of the proposed scheme is 4.5%, higher than that of the SLSBS. As the proportion of the malicious nodes increases to 20%, the $S_{\max}$ improvement ratio reaches 14%. Using SLSBS, when tasks with multiple delay thresholds are offloaded to edge nodes, the actual delay of the shard processing task must be less than the minimum delay threshold to meet the delay thresholds of all tasks, which leads to a smaller range of optional shard sizes. The MLSBS can meet the different delay threshold requirements of the task by providing shards of various sizes. The size of the shards for tasks with loose delay requirements can be larger. The larger the shard size, the more secure the task, so the security of the MLSBS is better than that of the SLSBS.

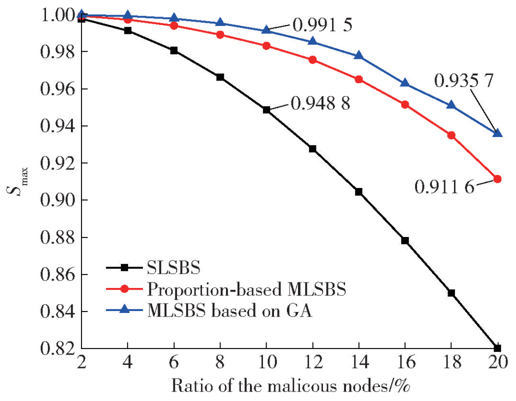

**Fig. 8** Security changes with malicious nodes injection

Fig. 8 illustrates that the MLSBS based on GA algorithm outperforms the scheme of allocating resources proportionally according to the proportion of tasks in terms of security. When the ratio of malicious nodes is 20%, the security of the MLSBS based on the GA algorithm is 2.6% higher than that of the MLSBS based on the tasks' proportion. Both of these methods can allocate resources according to the task composition of different delay thresholds in the current network, but the use of GA for resource allocation can find the optimal resource allocation scheme. When the traffic of a shard is same, security of service of the shard providing low latency service is worse than that of the shard providing high latency service. This is because the former allows the shard to contain fewer peers. The security of a certain level of shard can be improved by increasing the size of the shard. But the system needs to increase the number of shards at this level to reduce the tasks processed by each shard to ensure the delay less than threshold. When GA is used to allocate resources for each level, the security of service with the large delay threshold can be ensured, and the edge nodes of the service with the small delay threshold can be appropriately allocated to improve the security of service. Therefore, the achieved $S_{\max}$ by using GA is higher than that by allocating resources proportionally.

2) Impact of the number of terminals covered by each BS on security.

As shown in Fig. 9, the security via the three schemes with different numbers of terminals are measured. In the simulations, the number of terminals $m$ increase from 10 to 100. Each terminal generates one task per second and the ratio of malicious nodes is 20%.

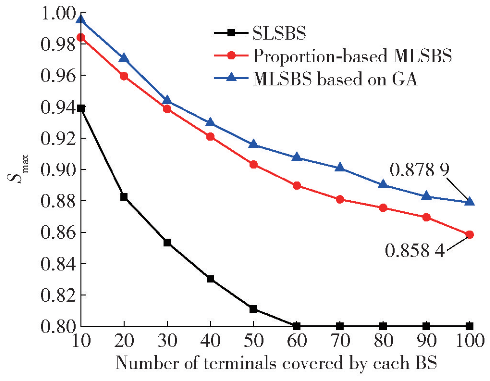

**Fig. 9** Security changes with the increase of the number of terminals

Fig. 9 shows that security of these three schemes decreases as the number of terminals increases. The number of terminals covered by each BS influences the sum of the tasks generated by the terminals and the task arrival rate of shards. The more the number of terminals, the higher the task arrival rate of shards, and the greater the queuing delay of tasks in the packaging system. Consensus delay can be reduced by reducing the nodes included in a shard to meet the delay threshold. Given the ratio of the malicious nodes, the smaller the size of a shard, the higher the probability that the ratio of malicious nodes in the shard will exceed 1/3, and the lower the proportion of safe shards, resulting in a decrease in the security of tasks. In the SLSBS, when $m = 60$, the shard size is 1 to meet the delay threshold of 0.1s, which means that the scheme degenerates into a blockchain-disabled EC scheme. The security of tasks is the ratio of honest nodes $1 - a$. Therefore, when the number of terminals continues to increase, security remains unchanged until blockchain-disabled EC solutions cannot meet the delay requirements of the tasks.

3) Impact of the ratio of delay-sensitive tasks on security.

Fig. 10 shows the security changed as the ratio of delay-sensitive (delay threshold is 0.1 s) tasks increases. As the proportion of the delay-sensitive tasks increases, the security of all tasks decreases. Delay-sensitive tasks require the blockchain to process tasks faster and allow fewer nodes in a shard, which degrades security. From Fig. 10, it can be seen that the higher the proportion of delay-sensitive tasks, the worse the security of all tasks in the system.

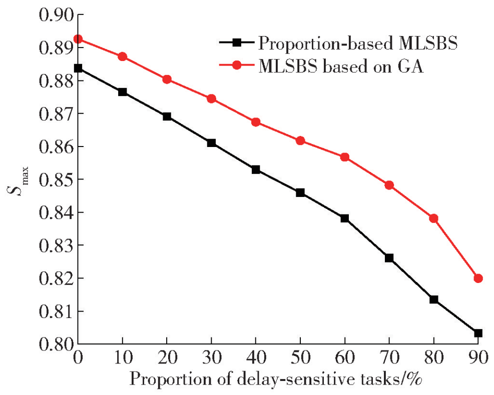

**Fig. 10** Security changes with the increase of the proportion of delay sensitive tasks

## 6 Conclusions and future work

MLSBS based on GA algorithm is proposed to support EC. This solution provides multiple services for tasks with various delay requirements, and maximizes the security of tasks by combining GA with adaptive resource allocation. Probability theory and queuing theory are used to analyze the impact of queuing delay on security and task delay, caused by a large number of tasks offloading requests. The simulation results show that the GA-based MLSBS results in an improved security performance compared to SLSBS and the scheme of allocating resources proportionally in a multiple types of task scenario. The MLSBS can be extended to maximize the scalability of other systems that also have different security requirements. Moreover, tasks in the EC system have different priority. In the case of limited resources, priority will be given to the allocation of resources to important applications. Future research needs to consider the priority of tasks and maximize system performance.

## References

1. Neumann D, Bodenstein C, Rana O F, et al. STACEE: enhancing storage clouds using edge devices. Proceedings of the 1st ACM/IEEE Workshop on Autonomic Computing in Economics (ACE'11), 2011, Jun 14 – 18, Karlsruhe, Germany. New York, NY, USA: ACM, 2011: 19 – 26
2. Yu W, Liang F, He X F, et al. A survey on the edge computing for the Internet of things. IEEE Access, 2018, 6: 6900 – 6919
3. Zheng Z B, Xie S, Dai H N, et al. An overview of blockchain technology: architecture, consensus, and future trends. Proceedings of the 2017 IEEE International Congress on Big Data (BigData Congress'17), 2017, Jun 25 – 30, Honolulu, HI, USA. Piscataway, NJ, USA: IEEE, 2017: 557 – 564
4. Zheng Z B, Xie S A, Dai H N, et al. Blockchain challenges and opportunities: a survey. International Journal of Web and Grid Services, 2018, 14(4): 352 – 375
5. Kang J W, Xiong Z H, Niyato D, et al. Toward secure blockchain-enabled Internet of vehicles: optimizing consensus management using reputation and contract theory. IEEE Transactions on Vehicular Technology, 2019, 68(3): 2906 – 2920
6. Wang W B, Niyato D, Wang P, et al. Decentralized caching for content delivery based on blockchain: a game theoretic perspective. Proceedings of the 2018 IEEE International Conference on Communications (ICC'18), 2018, May 20 – 24, Kansas City, MO, USA. Piscataway, NJ, USA: IEEE, 2018: 1 – 6
7. Asheralieva A, Niyato D. Reputation-based coalition formation for secure self-organized and scalable sharding in IoT blockchains with mobile-edge computing. IEEE Internet of Things Journal, 2020, 7(12): 11830 – 11850
8. SompolinskyY, Lewenberg Y, Zohar A. SPECTRE: a fast and scalable cryptocurrency protocol. IACR Cryptol ePrint Arch, 2016: Article 1159
9. Xiao Y, Zhang N, Lou W J, et al. A survey of distributed consensus protocols for blockchain networks. IEEE Communications Surveys & Tutorials, 2020, 22(2): 1432 – 1465
10. Kokoris-Kogias E, Jovanovic P, Gasser L, et al. OmniLedger: a secure, scale-out, decentralized ledger via sharding. Proceedings of the 2018 IEEE Symposium on Security and Privacy (SP'18), 2018, May 20 – 24, San Francisco, CA, USA. Piscataway, NJ, USA: IEEE, 2018: 583 – 598
11. Yoo H K, Yim J, Kim S. The blockchain for domain based static sharding. Proceedings of the 17th IEEE International Conference on Trust, Security and Privacy in Computing and Communications/12th IEEE International Conference on Big Data Science and Engineering (TrustCom/BigDataSE'18), 2018, Aug 1 – 3, New York, NY, USA. Piscataway, NJ, USA: IEEE, 2018: 1689 – 1692
12. Forestier S, Vodenicarevic D, Laversanne-Finot A. Blockclique: scaling blockchains through transaction sharding in a multithreaded block graph. arXiv Preprint, 2018, arXiv:1803.09029
13. Manshaei M H, Jadliwala M, Maiti A, et al. A game-theoretic analysis of shard-based permissionless blockchains. IEEE Access, 2018, 6: 78100 – 78112
14. Li S Z, Yu M C, Yang C S, et al. Polyshard: coded sharding achieves linearly scaling efficiency and security simultaneously. IEEE Transactions on Information Forensics and Security, 2020, 16: 249 – 261
15. The Zilliqa technical whitepaper. https://docs.zilliqa.com/whitepaper.pdf. 2017
16. Poon J, Dryja T. The Bitcoin lightning network: scalable off-chain instant payments. https://lightning.network/lightning-network-paper.pdf. 2016
17. Poon J, Buterin V. Plasma: scalable autonomous smart contracts. http://plasma.io/plasma.pdf. 2017
18. Liu C C, Xiao Y H, Javangula V, et al. NormaChain: a blockchain-based normalized autonomous transaction settlement system for IoT-based e-commerce. IEEE Internet of Things Journal, 2018, 6(3): 4680 – 4693
19. Liu H Y, Shen F, Liu Z Q, et al. A Secure and practical blockchain scheme for IoT. Proceedings of the 18th IEEE International Conference on Trust, Security and Privacy in Computing and Communications/13th IEEE International Conference on Big Data Science and Engineering (TrustCom/BigDataSE'19), 2019, Aug 5 – 8, Rotorua, New Zealand. Piscataway, NJ, USA: IEEE, 2019: 538 – 545
20. Yun J, Goh Y, Chung J M. DQN-based optimization framework for secure sharded blockchain systems. IEEE Internet of Things Journal, 2021, 8(2): 708 – 722
21. IEEE Std 1934$^{\text{TM}}$ – 2018. IEEE standard for adoption of OpenFog reference architecture for fog computing. 2018
22. Castro M, Liskov B. Practical Byzantine fault tolerance and proactive recovery. ACM Transactions on Computer Systems, 2002, 20(4): 398 – 461
23. Castro M, Liskov B. Practical Byzantine fault tolerance. Proceedings of the 3rd Symposium on Operating Systems Design and Implementation (OSDI'99), 1999, Feb 22 – 25, New Orleans, LA, USA. 1999: 173 – 186
24. Hafid A, Hafid A S, Samih M. A novel methodology-based joint hypergeometric distribution to analyze the security of sharded blockchains. IEEE Access, 2020, 8: 179389 – 179399
25. Gross D, Shortle J F, Thompson J M, et al. Fundamentals of queueing theory. New York, NY, USA: John Wiley & Sons, 2008
26. Holland J H. Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence. Cambridge, MA, USA: MIT Press, 1975

(Editor: Wang Xuying)