Hindawi
Security and Communication Networks
Volume 2021, Article ID 6688168, 15 pages
https://doi.org/10.1155/2021/6688168

Research Article

Efficient Concurrent Execution of Smart Contracts in Blockchain Sharding

Yan Wang,[1,2] Jixin Li,[1] Wansheng Liu,[1] and Aiping Tan [1]

[1]*College of Information, Liaoning University, Shenyang 110036, China*
[2]*State Key Laboratory of Robotics, Shenyang Institute of Automation, Chinese Academy of Sciences, Shenyang 110016, China*

Correspondence should be addressed to Aiping Tan; aipingtan@lnu.edu.cn

Received 13 November 2020; Revised 8 January 2021; Accepted 1 February 2021; Published 19 February 2021

Academic Editor: Abdelouahid Derhab

Throughput performance is a critical issue in blockchain technology, especially in blockchain sharding systems. Although sharding proposals can improve transaction throughput by parallel processing, the essence of each shard is still a small blockchain. Using serial execution of smart contract transactions, performance has not significantly improved, and there is still room for improvement. A smart contract concurrent execution strategy based on concurrency degree optimization is proposed for performance optimization within a single shard. This strategy is applied to each shard. First, it characterizes the conflicting contract feature information by executing a smart contract, analyzing the factors that affect the concurrent execution of the smart contracts, and clustering the contract transaction. Second, in shards with high transaction frequency, considering the execution time, conflict rate, and available resources of contract transactions, finding a serializable schedule of contract transactions by redundant computation and a Variable Shadow Speculative Concurrency Control (SCC-VS) algorithm for smart contract scheduling is proposed. Finally, experimental results show that the strategy increases the concurrency of smart contract execution by 39% on average and the transaction throughput of the whole system by 21% on average.

1. Introduction

Blockchain technology can be described as a distributed append-only ledger over a large peer-to-peer (P2P) network and has demonstrated great promise for utility in several fields including the Internet of Things (IoT), financial assets, the sharing economy, and copyright maintenance [1–3]. However, with the increasing transaction scale on the blockchain, the performance defects of the current blockchain platform are gradually being exposed (e.g., low throughput and lack of concurrency) [4–6], and the current platform is increasingly unable to meet the needs of large-scale applications. As an effective means to improve system performance, sharding proposals are applied to the blockchain system. Elastico is the earliest transaction sharding proposal facing the public chain. It divides the nodes in the network into multiple independent shards; each shard contains multiple nodes so that different shards can process irrelevant transactions in parallel and linearly improve the processing ability of the blockchain system. Although the introduction of sharding proposals can realize parallel processing between shards [7–9], the essence of each shard is still a small blockchain system, and a smart contract (SC) is executed in a serial way [10–12]. There is no significant improvement in the internal performance of the shard, so the SC may be limited by the performance of the shard. If the address within one shard has a high transaction frequency, the shard will generate a large amount of transaction information, which will lead to increased data conflicts while causing shard congestion [13].

To improve the performance of a single shard, we propose an SC concurrent execution strategy. First, the information of SC characteristics in a conflict is recorded, the collected information is used as an important reference factor to solve contract conflicts, and the subsequent smart contract transactions (SCTs) are clustered. Second, to implement concurrent execution of the optimized processed transaction set, we propose a Variable Shadow Speculative

Concurrency Control (SCC-VS) algorithm, which comprehensively considers the SC execution time E_t, conflict rate C_r, and available resources R, relying on redundant computation to find a serializable scheduling, which effectively solves the performance degradation problem caused by the increased number of SCTs.

We summarize the contributions of this paper as follows:

(1) We propose a feature information collection technology for SC. This method makes full use of the information resources of SC, records real-time statistics of the SC feature information that conflicts occur in the TSM-Module, and collects the feature information as the next important reference factor to resolve the SC conflict.

(2) We design a clustering technology for SCTs. First, the SCTs are initially partitioned by traversing the collected feature information. Second, the execution time E_t and conflict rate C_r of SCTs are predicted. Based on the predicted values, the aggregate function is used to divide again. Finally, we obtain three sets: Set_δ, Set_λ, and Set_μ (Set_δ, Set_λ, and Set_μ are obtained by the Concurrency Degree Optimization Processing Module, and the priority of execution in the Transaction Scheduling Management Module is from high to low). By changing the distribution of SCTs, the optimal processing of their concurrency is realized.

(3) We propose an SCC-VS algorithm for SC scheduling. This algorithm comprehensively considers the three dimensions of SCTs: execution time E_t, conflict rate C_r, and available resources R. It relies on redundant computation to find a serializable scheduling and executes the optimized SCTs concurrently. The problem of transaction blocking and restart caused by the existing methods is alleviated, and the system resources are effectively utilized.

(4) We implement the prototypes of the Concurrency Degree Optimization Processing Module (CDO-Module) and the Transaction Scheduling Management Module (TSM-Module) and apply them to the test environment. Many simulation experiments are run to verify the performance Turing improvement.

The paper is organized as follows: In Section 2, we review related work. In Section 3, we present a CDO-Module and explain its implementation in detail. In Section 4, we present a TSM-Module and analyze the theory. We present simulations and evaluations in Section 5 and conclude in Section 6.

2. Related Work

One important way to improve the performance of blockchain systems is to realize the concurrent execution of SC. In [14], Luu et al. first proposed introducing a secure sharding protocol into the public chain platform, aiming to build a sharding network structure that can achieve parallel computing. Although that approach improves the throughput to a certain extent, it uses a non-Turing complete language to create SC, and the flexibility of SC is insufficient and restricted in some cases. Dickerson et al. proposed a two-stage concurrent execution protocol of SCs based on the Lock method [15], which aimed to improve the performance of SC execution. However, since the Lock method belongs to a pessimistic concurrency control algorithm, poor scalability, serious blocking problems will occur when the conflict rate between SCTs is high; the experiment proves that the efficiency is not high. Anjana et al. proposed another SC execution framework based on the timestamp ordering method [16], allowing SCTs to be executed concurrently in an optimistic manner, optimistic concurrency control can be performed under low-conflict loads because lock synchronization is avoided. If frequent data conflicts occur between SCs, many transaction restarts will be caused. The multiversion concurrency control mechanism proposed by Zhang and Zhang [17] allows the validator to verify the consistency and certainty of blocks by executing SCTs concurrently, which accelerates the speed of block verification, but the corresponding work of the miner was not discussed in depth.

In addition, the concurrency control algorithm has a direct effect on the execution efficiency of SC. The existing Speculative Concurrency Control (SCC) algorithm is not very suitable for the blockchain sharding environment. For example, in [18], an improved SCC-2S algorithm was proposed for uncertain real-time spatial transactions, which aimed to ensure the freshness of data and did not meet the requirement to increase the concurrency. The SCC-NS algorithm introduced in [19] aimed to correct conflicts speculatively using redundant resources, but it also has a huge system overhead problem. Reference [20] provided a detailed comparison and quantitative evaluation of major sharding mechanisms, along with our insights analyzing the features and restrictions of the existing solutions. However, there was no clear description of the sharding mechanism for SCTs and no analysis of the security of efficient concurrent execution of smart contracts in the context of blockchain sharding. Reference [21] proposed a secure and effective construction scheme for blockchain networks, which built a directed acyclic graph (DAG) blockchain network through the network link protocol, and carried the sharding technology on the DAG blockchain to realize the parallel processing of transactions. However, this method did not study the performance improvement within each shard. A comparison of the main contributions between existing work is shown in Table 1.

3. CDO-Module

For clarity, this paper defines transactions on SCs as SCTs. An SCT code is executed once by the miner and multiple times by the validator. For a blockchain system in the context of shard, the maximum concurrency of transaction execution is a very important performance index, where concurrency refers to the number of SCTs executed concurrently. High concurrency can not only improve the utilization of system resources but also maximize transaction throughput. Because the existing SC execution

Table 1: A comparison of existing contributions.

Property	Sharding protocol	Lock method	Timestamp ordering method	Multiversion concurrency control mechanism	SCC algorithm
Smart contract	Available	Available	Available	Available	N/A
Concurrency	Yes	Yes	Yes	Yes	Yes
Efficiency	High	Low	Middle	High	Middle
Security	Poor	Favorable	Favorable	Favorable	Moderate
Energy saving	Partial	Partial	Yes	Partial	No

strategies are not optimized for high-concurrency services, this work constructs the Concurrency Degree Optimization Processing Module (CDO-Module). This module contains two subunits: a Feature Information Acquisition Unit (FIA-Unit) and a Classification and Monitoring Unit (CAM-Unit). With the introduction of sharding proposal, all network SCTs are mapped to different shards for processing. To reduce the performance decline caused by excessive SCTs in a single shard, the SCTs after sharding proposal will be preprocessed in the CDO-Module.

3.1. FIA-Unit. To assist the efficient operation of the TSM-Module, this work sets up FIA-Unit. This unit will record statistics of the SC feature information on conflicts in the TSM-Module in real time and use the collected feature information as an important reference factor to resolve SC conflicts. This feature information includes the corresponding SC account address and related member functions with high conflict frequency. A Feature Information Statistics Table (FIS-Table) is generated based on the mined feature information, and the table is maintained by FIA-Unit. FIS-Table records two types of data: Conflicting Contract Account Sets (C-CA Sets) and High-Conflict Rate Member Functions Sets (H-CRMF Sets).

Existing strategies adopt only a type of concurrency control method to resolve conflicts between SCs but do not consider how to make full use of the SC information resources to further improve the concurrency in their execution [22]. The essence of SC is a reusable, immutable, and automatically executed computer program that runs on the network, which cannot be actively executed. Its interaction mode is divided into the external call and internal call [23]; that is, Externally Owned Accounts (EOA) call SC and Contract Accounts (CA) call SC. Correspondingly, FIA-Unit's statistical analysis of feature information is divided into statistical analysis of C-CA Sets and H-CRMF Sets. When the TSM-Module executes the SC concurrently, it will record the new conflicts and feed them back to FIA-Unit. Among them, the conflicting SC account address is recorded in the C-CA Sets of FIS-Table, and the relevant SC functions with a higher conflict frequency are recorded in the H-CRMF Sets to ensure the continuous update of the statistics in FIS-Table. Figure 1 shows the infrastructure model of each module and unit.

3.2. CAM-Unit. Because the distribution of SCTs has a great impact on the performance of concurrent execution, this work uses the CAM-Unit. This unit divides SCTs into different sets by relevant factors to optimize the concurrency. At the same time, CAM-Unit will limit the number of SCTs executed, which fixes the calculation load, reduces the probability of conflicts, and keeps the concurrency speedup ratio S_{con} within the ideal range. In the case of n nodes, the S_{con} expression is shown as follows:

$$S_{\text{con}} = \frac{T_{\text{ser}}}{T_{\text{con}}} = \frac{W_{\text{ser}}}{W_{\text{ser}} + (W_{\text{con}}/n)} = \frac{W_{\text{ser}}/(W_{\text{ser}} + W_{\text{con}})}{(W_{\text{ser}}/(W_{\text{ser}} + W_{\text{con}})) + ((W_{\text{con}}/n)/(W_{\text{ser}} + W_{\text{con}}))} = \frac{\partial}{\partial + ((1-\partial)/n)}, \tag{1}$$

where T_{ser} represents the time for serially executing SCTs, and T_{con} represents the time for concurrently executing SCTs. W_{ser} represents the load of the serial part, W_{con} represents the load of the concurrent part, and ∂ is the proportion of the serial part; that is, $\partial = W_{\text{ser}}/(W_{\text{ser}} + W_{\text{con}})$, $1 - \partial = W_{\text{con}}/(W_{\text{ser}} + W_{\text{con}})$.

Subsequent SCTs are preliminarily classified by traversing FIS-Table to determine whether each SC has "feature information." For reclassification, the estimated execution time E_t and conflict rate C_r of SCTs must be comprehensively considered. The complete grouping process is shown in Figure 2.

3.2.1. The Value of Execution Time (E_t). The sharding design scheme can map many SCTs in the network to different shards by category. Therefore, for the value of E_t for the SCT, we assume that "similar jobs have similar execution times" and use the E_t of the completed SC to predict the E_t of similar SCs. Suppose that to estimate the E_t of a contract transaction J_{sc}, the specific steps are as follows:

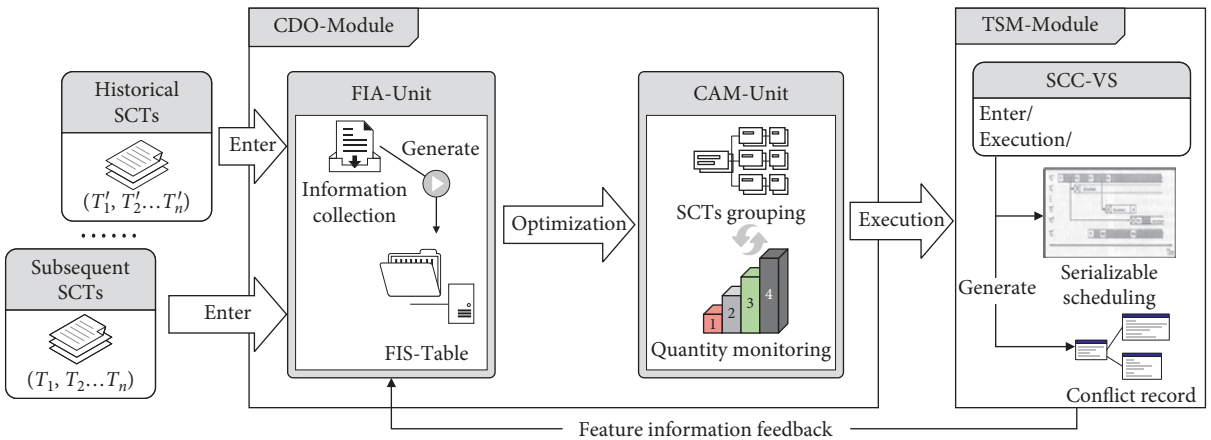


Figure 1: Infrastructure model.

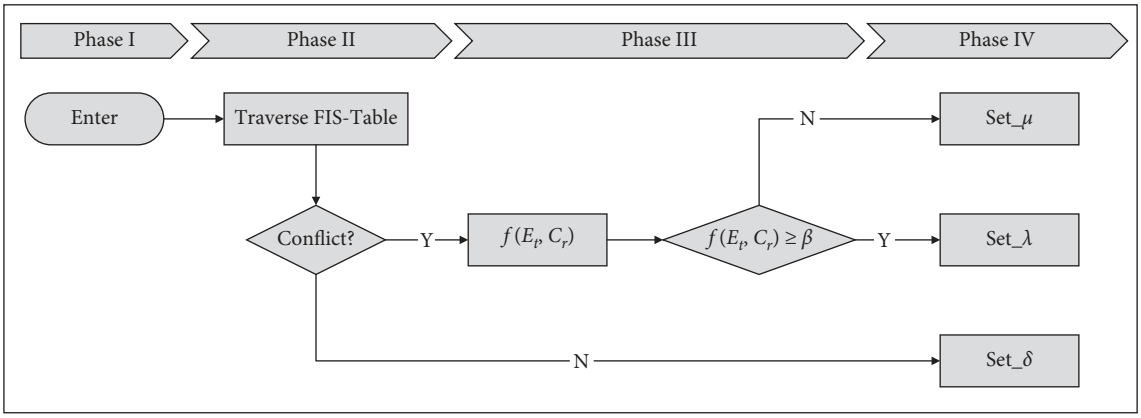


Figure 2: Subsequent contract set grouping process.

(1) Taking into account the fact that the SCTs are accompanied by Bandwidth Consumption, Storage Consumption, Calculation Consumption, etc., a template is first determined, and the above three consumption factors are regarded as three attribute values: Bandwidth Consumption (Mbps), Storage Consumption (MB), and Calculation Consumption (hash/s) as the constituent elements of the template. In this paper, B, S, and C are used, respectively, that is, $\{B, S, C\}$.

(2) A maximum of three consumption factors needs to be limited before an SCT J_{sc} is issued. According to the template $\{B, S, C\}$, select SCTs similar to J_{sc} and form a set $G^*_{\langle sc,j\rangle}$.

(3) Because the properties of the three attribute values in the template $\{B, S, C\}$ are different and usually have different dimensions and orders of magnitude, to ensure the reliability of the results, the original data needs to be standardized first; that is, the min-max standardized method is adopted to carry out the linear transformation of the original data and map it to the interval of [0,1], to eliminate the dimensional influence between different dimensions and facilitate the subsequent calculation. The sequences $B_1, B_2, \ldots, B_n, S_1, S_2, \ldots, S_n, C_1, C_2, \ldots, C_n$ are transformed by the following calculation method:

$$y_i = \frac{x_i - \min_{1\leq j\leq n}\{x_j\}}{\max_{1\leq j\leq n}\{x_j\} - \min_{1\leq j\leq n}\{x_j\}}. \quad (2)$$

(4) In set $G^*_{\langle sc,j\rangle}$, use equation (3) to calculate the numerical similarity between J^i_{sc} and J^j_{sc}, and M SCs similar to J_{sc} are selected to form the set $G_{\langle sc,j\rangle}$. Euclidean Metric is usually used to measure distance. The larger the value, the farther the distance. In this paper, the reciprocal is taken, and the farther the distance is, the closer the reciprocal value is to 0, which indicates that the similarity between SCTs is lower. $\mathrm{Sim}(J^i_{sc}, J^j_{sc})$ based on the three numerical attribute values of B, S, and C is defined as follows:

$$\mathrm{Sim}\left(J^i_{sc}, J^j_{sc}\right) = \frac{1}{1 + \sqrt{\left|B_i - B_j\right|^2 + \left|S_i - S_j\right|^2 + \left|C_i - C_j\right|^2}}. \quad (3)$$

(5) After obtaining a similar set $G_{\langle sc,j\rangle}$ of J_{sc}, the actual E_t of the SCTs in $G_{\langle sc,j\rangle}$ can be used to predict the E_t of J_{sc}.

The average method is used in this paper; that is, the average value of E_t of the SC in $G_{\langle sc,j\rangle}$ is used as the prediction time of J_{sc}, and the calculation equation is as follows:

$$E_t(J) = \frac{\sum_{i=1}^{M} R_i}{M}, \tag{4}$$

where R_i is the actual E_t of the ith SC in $G_{\langle sc,j\rangle}$.

3.2.2. The Value of Conflict Rate (C_r). The SC conflict rate refers to the probability of an SC conflicting with any other SC when it is executed, and it is ideally judged based on the current conflict situation. Because the SCs cannot be statically analyzed [24], it is impossible to know whether there will be a conflict before the SC is executed, so it is impossible to judge the probability of an SC conflict based on the system status at a certain moment. Considering that the high incidence of conflict is mainly caused by a few popular SCs in a certain period of time, we assume that "the short-term conflict rate of the transaction execution period has a greater impact on the predicted value." The C_r of an SCT is predicted by the conflict rate of the past period. Besides because the contract conflict rate has nonlinear characteristics, it is not suitable to use the linear regression equation to calculate the contract conflict rate. Therefore, this paper uses the weighted moving average method [25] based on feedback value to predict the C_r of an SCT. The specific steps are as follows:

(1) The weighted moving average method is used to calculate the original conflict rate C_r'. The weighted moving average method has the characteristics of simple logic and high prediction accuracy. It takes time as the standard and gives the larger weight to the data closer to the prediction time. This can make up for the lack of equal treatment of all data by the moving average method and sensitive response to recent trends in data. The basic calculation method of the original conflict rate C_r' is shown in the following equation:

$$C_r' = \frac{g_1 U_{n-1} + g_2 U_{n-2} + \cdots + g_i U_{n-i}}{\sum_1^i g_i}. \tag{5}$$

C_r' represents the n prediction result; U_{n-i} represents the conflict rate detected in the n_i time period; i represents the number of reference values; and g_i represents the weight value of the i reference value.

(2) The weighted moving average method has a high accuracy for short-term prediction, but the weights of the method need to be set in advance and will not change when the weights are determined, so the accuracy of the original C_r cannot be effective feedback. This also limits the accuracy of contract conflict rate prediction. In order to further improve the accuracy of the prediction, we need to calculate the feedback value, so that the results of each prediction can be fed back to the next calculation of the prediction results. The calculation method is shown in the following equation:

$$F_n = \frac{\sum_1^i v_m \times \left(C_r^{n-m}/U_{n-m}\right)}{\sum v_m}. \tag{6}$$

F_n represents the feedback value of the n time; C_r^{n-m} represents the final prediction value of the $n-m$ time; v_m is the weight value; and F_n greater than 1 predicted value is too large and less than 1 means that the predicted value is too small.

(3) We can see from equations (5) and (6) that the calculation of initial prediction value C_r' and feedback value F_n are used to calculate the weight value g and v; selecting the appropriate weights has a great influence on the final prediction results. Considering that the prediction of C_r has obvious time characteristics, the closer observation value is to the prediction point, the greater the effect is on the result. In this paper, the attenuation factor k is considered to determine the weight value. The initial value of the weight of g in equation (7) is 1. The calculation equation of weight g_i is

$$g_i = \frac{g_{i-1}}{k_1}. \tag{7}$$

The initial value of the weight in equation (8) is 1. The calculation equation of weight v_i is

$$v_i = \frac{v_{i-1}}{k_2}. \tag{8}$$

(4) The concept of feedback value is based on the calculated value of the weighted moving average method. The final C_r predicted value is equal to the ratio of weighted moving average value to feedback value. The calculation method is as follows:

$$C_r = \frac{C_r'}{F_n}. \tag{9}$$

After calculating the estimated execution time E_t and conflict rate C_r of an SCT, it is necessary to consider them comprehensively and judge the threshold value. The calculation equation of the aggregation function is as follows:

$$P(T_{sc}) = f(E_t, C_r) = \lceil \alpha \cdot \frac{w_t}{E_t} + (1-\alpha) \cdot \frac{w_r}{C_r} \rceil, \tag{10}$$

where α is a parameter; w_t and w_r are the weights of E_t and C_r, respectively.

First, by traversing FIS-Table, the SCTs without "feature information" are recorded in Set_δ. Second, the P decision must specify a threshold β. If $P \geq \beta$, the SCT is recorded in Set_λ; otherwise, it is recorded in Set_μ. Finally, the subsequent SCTs are divided into three groups: Set_δ, Set_λ, and Set_μ. The SCTs in Set_δ have the lowest conflict probability and the shortest execution time; the set is given the highest priority when the SCTs are executed, followed by Set_λ.

Set_μ has the highest conflict probability and the longest execution time, so the set is given the lowest priority. The SCTs in Set_δ, Set_λ, and Set_μ will be executed concurrently in turn by the SCC-VS algorithm of TSM-Module, while the Set_δ, Set_λ, and Set_μ will be executed serially.

4. TSM-Module

To achieve concurrent execution of the optimized SCTs, this work sets up the Transaction Scheduling Management Module (TSM-Module), which uses an improved SCC algorithm to execute SCs concurrently. The Speculative Concurrency Control algorithm is based on the optimistic method and relies on redundant computing to find a serializable scheduling. This algorithm can reduce the blocking and restart of SCTs while preprocessing the conflicts. According to SCC-NS [26], the number of shadows generated is positively related to the degree of concurrency, but at the cost of a high amount of calculation. Therefore, after balancing the two factors of concurrency and computational cost, a Variable Shadow Speculative Concurrency Control (SCC-VS) algorithm is proposed. The SCC-VS dynamically calculates the number of shadows N required by SCTs from the three aspects of execution time E_t, conflict rate C_r, and available resources R, as shown in the following equation:

$$N(T_{sc}) = f[C_r(T_{sc}), E_t(T_{sc}), R(T_{sc})] = \left\lfloor \frac{C_r(T_{sc}) + E_t(T_{sc}) + e^{(R(T_{sc}))/(R(T_{sc})_o)}}{\phi} \right\rfloor, \tag{11}$$

where ϕ is a constant coefficient, e is a constant, R_o is the average amount of idle resources in the system, $C_r(T_{sc})$ represents the conflict rate of the contract transaction T_{sc}, $E_t(T_{sc})$ represents the execution time of T_{sc}, and $R(T_{sc})$ represents the amount of idle resources available for T_{sc} execution.

Through the analysis and research of the concurrency problem of SC and the existing work, this paper adopts the two-stage concurrent execution framework of smart contracts. As shown in Figure 3 below, it takes into account the execution efficiency of the main node (miner node in the PoW, leader node in the BFT) at the same time, the playback efficiency of validation node can be guaranteed. Usually, an SCT is executed twice in its full life cycle. The first time main node creates a block; the second time validation node verifies the block. Specifically, the client will first broadcast the SCT to each node. In the first execution stage (the main execution stage), the main node collects a batch of SCTs, then uses the concurrency control algorithm to realize the concurrent execution of the SCTs, then packages the SCTs and records the conflict record in the execution process into the block. Finally, broadcast to the validation node. After receiving the consensus block, the validation node enters the second execution stage (verification stage), uses the conflict record transmitted by the main node to playback the same batch of SCTs, and deterministically calculates a new state transfer, generating the same serializable scheduling as the main node to verify the validity of the block.

The SC concurrent execution strategy based on concurrency degree optimization proposed in this paper consists of two parts. The first part is based on CDO-Module to optimize the concurrency degree of SCTs. The second part is based on the SCC-VS algorithm to execute SCTs concurrently. The complete operating mechanism is shown in Figure 4.

First, in the blockchain sharding environment, each node obtains a set of SCTs from the P2P network, and each transaction is associated with the SC function. Each SC function consists of multiple steps such as lookup, insert, and delete on shared data items. The concurrency of the SCTs is optimized with the help of CDO-Module. After entering the first stage—the main stage—to execute SCs concurrently, the miner node loads the optimized SCTs into an isolated sandbox environment in this stage, uses SCC-VS (see Algorithm 1) to identify conflicts, and allows the main node to record the conflicted relationship and specific conflict data items in real time, thus forming the conflict record. And then, the miner that executes concurrently proposes a block, which is composed of a contract transaction set, conflict records, the final status, the hash value of the previous block, and other information. It is verified by other validators in the P2P network. SCC-VS consists of five rules. The process of SCC-VS algorithm is shown in Algorithm 1.

Go later to the second stage—validation stage; the validator verifies the block proposed by the concurrent miner. The concurrent validator analyzes the conflict record in the block to identify the conflict relationship between SCTs. Because all conflicting relationships between SCTs have been identified by the miner, the validator can execute the set of SCTs in a concurrent, deterministic manner with the help of the conflicting records provided by the miner. After successful execution of the SCTs, the validator compares the calculated final state with that given by the concurrent miner. If the final state matches, then it is proved that the block proposed by the concurrent miner is valid. At this point, it is necessary to feedback the new conflict record to CDO-Module to update its maintained FIS-Table. Finally, the block is added to the blockchain and miners are rewarded accordingly.

TSM-Module uses SCC-VS, which will greatly reduce transaction blocking and restart problems. At the same time, it guarantees that the SCTs achieve higher concurrency at a lower computational cost when executed.

Moreover, we also conducted a detailed analysis on the security of the blockchain sharding model used in this paper,

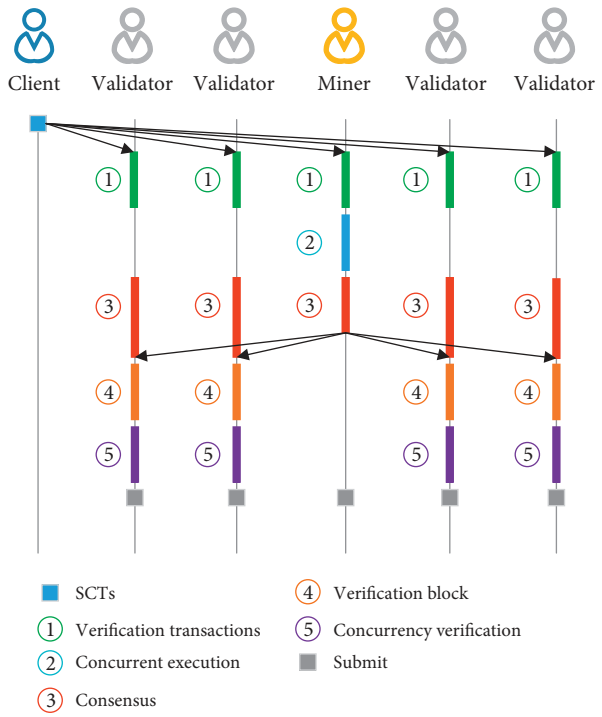
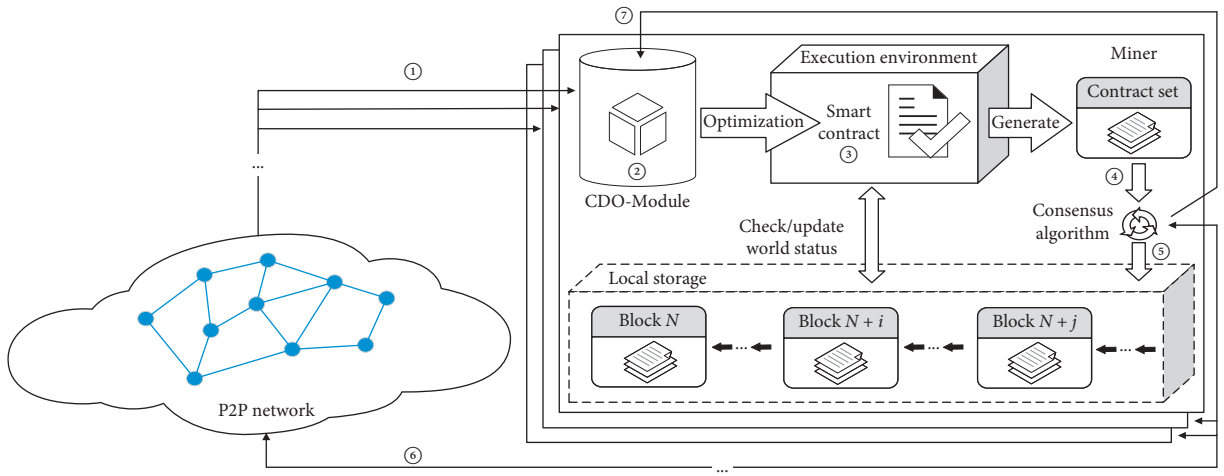


Figure 3: Two-stage concurrent execution process for smart contracts.



①: Miner receive various SCTs from the network.
②: CDO-Module optimizes the concurrency of SCTs.
③: Execute smart contracts concurrently and calculate final status.
④: Generate a contract set from the miner.
⑤: Generate a block and submit.
⑥: The new block is sent to the whole network node.
⑦: Update new conflicts to CDO-Module.

Figure 4: Logical system model.

aiming at several common attack modes in the blockchain network: Distributed Denial-of-Service Attack, 51% Attack, Empty Block Attack, and Sybil Attack.

The first is Distributed Denial-of-Service (DDoS) attack. It is a special form of denial-of-service attack based on DoS. It is a distributed, coordinated, large-scale attack. By flooding the network with a large number of useless requests, the attacker tries to overload the system, which leads to the fact that the users in the network cannot access the network resources normally and paralyzes the system. DDoS attacks are usually launched by an attacker to take control of an internal platform or to demand a ransom from the injured party. In the sharding model we used, if we want to carry out DDoS attack, we must include all nodes in the blockchain network into the attack scope, which may cause the system to crash. As we add more nodes to the blockchain network, the number of nodes will increase and the attack cost will be too much for the attacker.

(A) Start Rule: When the execution of a new transaction T_{sc} is requested, create and execution an Optimistic shadow $T^o_{\langle sc,r\rangle} \in T^o_{sc}$;
(1) Compute the number of shadows $N(T_{sc})$;
(2) Pessimistic Shadows $(T_{sc}) \leftarrow 0$;
(3) ReadSet $(T^o_{sc}) \leftarrow \varphi$;
(4) WriteSet $(T^o_{sc}) \leftarrow \varphi$;
(B) Read Rule: Whenever a transaction $T_{\langle sc,r\rangle}$ wishes to read object X, a conflict may be found out, then
(1) ReadSet $(T^o_{\langle sc,r\rangle}) \leftarrow \{X\}$;
(2) if (Pessimistic Shadows $(T_{\langle sc,r\rangle}) < N(T_{\langle sc,r\rangle}) - 1)$ then {
(2.1) Fork a new pessimistic shadow $T^p_{\langle sc,r\rangle}$;
(2.2) WaitFor $(T^p_{\langle sc,r\rangle}) \leftarrow \{(T_{\langle sc,u\rangle}), X\}$;
(2.3) Pessimistic Shadow $(T_{\langle sc,r\rangle}) \leftarrow$ Pessimistic Shadow $(T_{\langle sc,r\rangle}) + 1\}$;
(2.4) else if (Pessimistic Shadows $(T_{\langle sc,r\rangle}) \nless N(T_{\langle sc,r\rangle}) - 1)$ then{Abort $(T_{\langle sc,r\rangle})\}$;
(C) Write Rule: Whenever a transaction $T_{\langle sc,u\rangle}$ wishes to write object X, a conflict may be found out, then
(1) WriteSet $(T^o_{\langle sc,u\rangle}) \leftarrow \{X\}$;
(2) if (Pessimistic Shadows $(T_{\langle sc,u\rangle}) < N(T_{\langle sc,u\rangle}) - 1)$ then{
(2.1) Fork a new pessimistic shadow $T^p_{\langle sc,u\rangle}$;
(2.1.1) WaitFor $(T^p_{\langle sc,r\rangle}) \leftarrow \{(T_{\langle sc,u\rangle}), X\}$;
(2.1.2) Pessimistic Shadow $(T_{\langle sc,r\rangle}) \leftarrow$ Pessimistic Shadow$((T_{\langle sc,r\rangle}) + 1\}$;
(2.2) else if (existence conflict) then {
(2.2.1) Abort the shadow $T^p_{\langle sc,r\rangle}$ and replace it by a new shadow $T^p_{\langle sc,r\rangle}$;
(2.2.2) WaitFor $(T^p_{\langle sc,r\rangle}) \leftarrow \{(T_{\langle sc,u\rangle}), X\}$;
(3) else if (Pessimistic Shadows $(T_{\langle sc,u\rangle}) = N(T_{\langle sc,u\rangle}) - 1)$ then {Abort $T_{\langle sc,r\rangle}\}$;
(D) Blocking Rule: Block a pessimistic shadow T^p_{sc} at the earliest point at which it wishes to read on object X
(E) Commit Rule: whenever it is decided to commit an optimistic shadow $T^o_{\langle sc,r\rangle}$ on behalf of a transaction $T_{\langle sc,r\rangle}$, then
(1) Abort other pessimistic shadows except $T^o_{\langle sc,r\rangle}$;
(2) Deal with everything that conflicts with $T_{\langle sc,r\rangle}$;

Algorithm 1: A Variable Shadow Speculative Concurrency Control (SCC-VS).

The second is Sybil attack, which means that the attacker uses a single node to forge multiple virtual identities and make them exist in the P2P network, to reduce the robustness of the network, interfere with the normal activities of the network, and other purposes. In a blockchain sharding environment, an attacker would also need to create multiple accounts to carry out a Sybil attack. However, the sharding design scheme used in this paper can restrict the validation nodes to a certain extent; that is, the validation nodes need to pledge a certain number of tokens before entering the shard to verify the transaction, which makes it very difficult for the attacker to create a large number of identities in a short time.

Then, there is the 51% attack, which is the most famous type of attack in the blockchain. For example, in the Bitcoin network, once an attacker controls more than 51% of the computing power of the whole network, the attacker can tamper with the historical data in the network and indirectly grasp the right of keeping accounts in the Bitcoin network. In the blockchain sharding environment, the attack method can be understood as more than 51% of the validation nodes in the shard jointly commit crimes, that is, conspiracy attack. However, two conditions must be satisfied for the occurrence of conspiracy attack in shard:

(1) The number of malicious nodes in the shard should be greater than 2/3 of the total number of nodes in the shard

(2) Malicious nodes should collude together for joint evil

If more than 51% but not more than 2/3 of the validation nodes work together, there will be no consensus, i.e., consensus timeout. The sharding design scheme used in this paper will limit the number of consensus timeouts, which effectively reduces the probability of the occurrence of this type of attack. When consensus timeouts occur several times in a row, we will abandon the transaction and reassigned the transaction, so 51% attack cannot be implemented.

Finally, there is the Empty Block attack, in which miners fill the block head without verifying any transactions to solve the consensus problem as soon as possible, to be able to publish the block faster, and get the block reward during the competitive mining process. Although the empty block attack does not affect the effectiveness of the blockchain, if the frequent occurrence of empty blocks will lead to the continuous accumulation of transaction requests, the transaction memory pool continues to grow, and the average transaction confirmation time becomes longer.

This situation is similar to the 51% attack. We only need to appropriately expand the sharding scale and optimize the performance within a single shard to solve this problem. Moreover, compared with the empty block attack, there is no reason for miners to do so, which exploits the principle of economics.

From the above analysis, it can be seen that the sharding design scheme used in this paper has a certain resistance capability to several common attack modes, which can ensure the normal operation of the blockchain sharding system using the strategy proposed in this paper.

Table 2: Configuration of the experimental environment.

Deploy	Detailed description
OS	CentOS 8.0
CPU	Intel Xeon Silver 4110 CPU @2.1GHzx 16 core
Memory	160 GB
JDK version	1.8
Network adapter	Gigabit Broadcom NetXtreme II BCM5709 1000BaseT

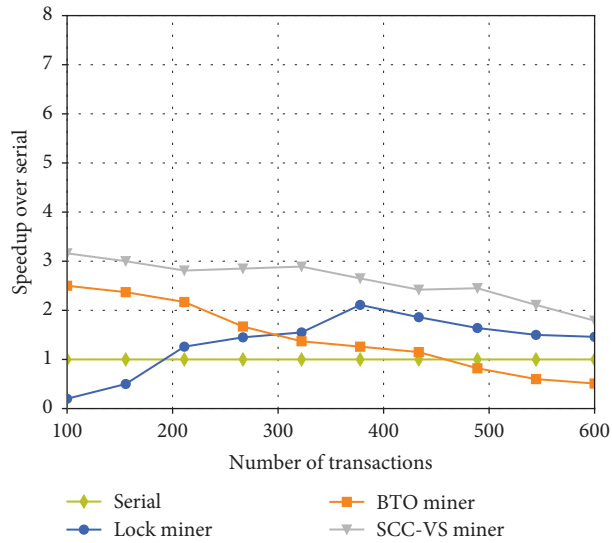


Figure 5: Accelerated changes as transaction flow increases.

5. Experiments

In this section, the performance of the proposed SC concurrent execution strategy based on concurrency degree optimization is verified by experiments. Since the existing SC models of blockchain are all single-threaded models (such as Ethereum's EVM) [27], the SC concurrent execution strategy proposed in this paper is difficult to be implemented in the real blockchain system. As a result, this experiment completes all performance reviews on a server and uses Java language to simulate real smart contract execution [28, 29]. The load generator implemented experimentally in this section synthetically considers the number of SCTs and accounts for generating the corresponding load for each set of experiments. Distribution of transaction types adopts random classes to achieve uniform distribution. The data access mode accords with the Zipfian distribution to simulate the SCTs' conflict scenario. Specifically, the larger the parameter, the higher the conflict rate between the SCTs. The server configuration required to run the experiment is shown in Table 2.

Because this paper focuses on the concurrent execution of SCs, it simplifies the POW and other related factors in the process. The experiment mainly focuses on the following aspects of performance: (1) when the SCTs increases, the comparison of acceleration changes in each method; (2) when the conflict rate increases, the comparison of acceleration changes in each method; (3) when the number of nodes increases, the comparison of throughput changes in each method; (4) as the number of shards increases, the throughput of individual shard and the whole system changes; (5) as the number of shards increases, the storage overhead of the nodes changes; (6) the conflict records in FIS-Table change as the number of SCTs increases; (7) the Cumulative Distribution Function (CDF) [30] of the estimated execution time and actual execution time of SC is achieved; and (8) the throughput result of the security experiment. All experimental results are the average values taken after multiple executions to reduce errors.

The experiment compares the smart contract concurrent execution strategy proposed in this paper with the other two traditional concurrency control algorithms and uses the results of serial execution as a baseline to simulate the average acceleration in each method. By analyzing the experimental results in Figure 5, it can be seen that when the transaction flow is low, the use of the Lock algorithm relatively does not bring about acceleration or even a slowdown. This is because the additional overhead caused by conflict processing has an impact on concurrent performance. With the continuous increase in transaction flow on the platform, the Lock algorithm starts to slow down similarly to the BTO algorithm after a period of acceleration. Based on the optimization of concurrency and the improvement in transaction blocking and restarting, the proposed strategy can mitigate the performance degradation caused by increased transaction flow.

From the analysis of the experimental results in Figure 6, it can be seen that with increasing conflict rate, the acceleration caused by the three methods shows a downward

trend. When the conflict rate is close to 68%, using the BTO algorithm to execute SCs is slower than serial execution. In contrast, the Lock algorithm, which uses the pessimistic method as an example, is more adaptable to situations with a higher conflict rate. However, due to the increased conflict rate, the probability of cross-shard interaction is also increasing, and the complexity is becoming increasingly higher [31], so the overall trend is also declining, but the overall implementation is still slightly better than the above two ways.

According to the experimental results shown in Figure 7, strategies proposed in this paper can be compatible with sharding proposals and still maintain the characteristic of linear scalability; that is, with increasing number of nodes and network volume, processing performance can be improved by parallelizing the data flow. Under the traditional method, even if SCs are executed concurrently, as more nodes are added, their trading speed will still decrease.

Under the cooperation of CDO-Module and TSM-Module, the strategy proposed in this paper improves the SCTs' execution efficiency and the whole system's throughput by optimizing the performance of each shard. We compare it with the traditional sharding blockchain, taking Elastico public blockchain as an example. Elastico first proposed to adopt the sharding model in the public blockchain system [32], which almost completes the linear expansion of the throughput of the block. Analyzing the experimental results in Figure 8, under the traditional method, although the network-wide TPS increases as the number of shards increase, the TPS of a single shard is still low, only approximately 50, and there is no significant performance improvement. While the strategy proposed in this paper guarantees the performance improvement of a single shard, the throughput of the whole system also reaches a high level.

For the overhead of sharding storage, we calculate the amount of data stored in each shard. Because cross-shard SCTs will be stored by multiple shards, we sent 5%, 10%, 15%, and 20% cross-shard SCTs in this experiment. Analysis of the experimental results in Figure 9 shows that the storage overhead of each node decreases as the total number of shards increases. In the case of the same number of shards, the more cross-shard SCTs, the greater storage overhead. Furthermore, we note that storage optimization mechanisms can be used to further reduce storage overhead.

We use the SCC-VS algorithm to implement concurrent execution of SCs in the TSM-Module. At the same time, the miner will propose a new block, which consists of information such as the set of SCTs, the conflict record, the final state, and the hash value of the previous block. TSM-Module feeds back the feature information in the conflict record to the FIA-Unit, the purpose of which is to achieve the preprocessing of SCTs. Figure 10 shows the relationship between the number of SCTs and conflict records of four types of SCs (i.e., a, b, c, and d) under different concurrent control algorithms. By analyzing the experimental results shown in Figure 10, we can see that no matter what method is adopted, the conflict records in FIS-Table will increase with increasing SCTs. However, the SCC-VS algorithm proposed in this paper shows better performance due to its application in a single shard, which diverts many SCTs into different shards and optimizes the degree of concurrency between SCTs.

Currently, because of the small number of SCTs in the block, the storage of conflict records between SCTs in the block will not consume too much space. Therefore, storing conflict records of SCTs in a block does not consume much space. Over time, if conflict records increase, more storage space will be consumed. Hence, it is important to provide the best conflict record or to properly implement concurrent execution of SCs without a conflicting record.

In the CAM-Unit, E_t for the SC must be predicted. To verify the effectiveness of the method, the Cumulative Distribution Function (CDF) of the estimated-E_t and actual E_t of four different SCs are given in Figure 11. This graph describes the probability that the estimated-E_t and actual E_t of four different types of SCs fall within any time interval. Through the execution of four SCs, we find that the actual runtime corresponding to the same probability density is slightly smaller than the E_t we calculated. Causes of this situation, on the one hand, due to the decreasing execution time, on the other hand, because we generally overestimate the runtime of the four SCs during this experiment. It can be seen in Figure 11 that the actual E_t distribution of the four SCs is smooth, while the estimated-E_t is ladder-shaped, which shows that the estimated-E_t calculated by this method is relatively rough. Therefore, for the prediction algorithm in this paper, it is easier to obtain a good prediction effect (e.g., b and d type contracts) for SCTs with a ladder-shaped actual E_t distribution.

In order to evaluate the resistance of the blockchain sharding system using the concurrent execution strategy proposed in this paper to malicious nodes, we set up 60 nodes, among which 15 nodes are malicious nodes and the other 45 nodes are honest nodes, as shown in Figure 12. The honest node confirms all transactions it receives, while the malicious node stops verifying transactions and is rejected as the master node each time it is elected. In this way, we test the security of the scheme in this paper to verify its resistance to malicious nodes in the long run.

Figure 13 is the throughput result of the security experiment. After calculation, it can be seen that the average transaction throughput of the blockchain system using traditional random sharding technology is about 412.3 TPS. The average transaction throughput of the blockchain sharding system using the strategy proposed in this paper is about 525.1 TPS, higher than that of the traditional scheme. The reason is that the malicious node stops validating the transaction and is rejected as the master node each time it is elected. At this point, the other nodes in the shard broadcast the emergency message and start the rollback program and then select a new master node. In this process, the shard stops working and the transaction cannot be verified before the new master node is elected, resulting in a rapid decline in throughput. However, overall, the overall average throughput of the strategy proposed in this paper is still better than that of the traditional scheme.

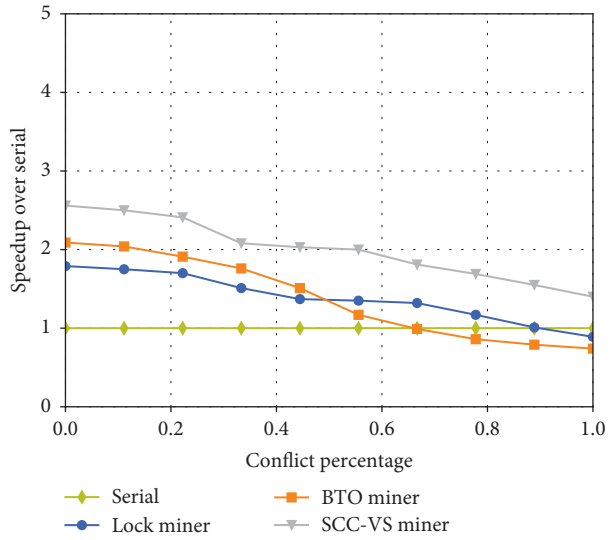


Figure 6: Accelerated changes as conflict rates increase.

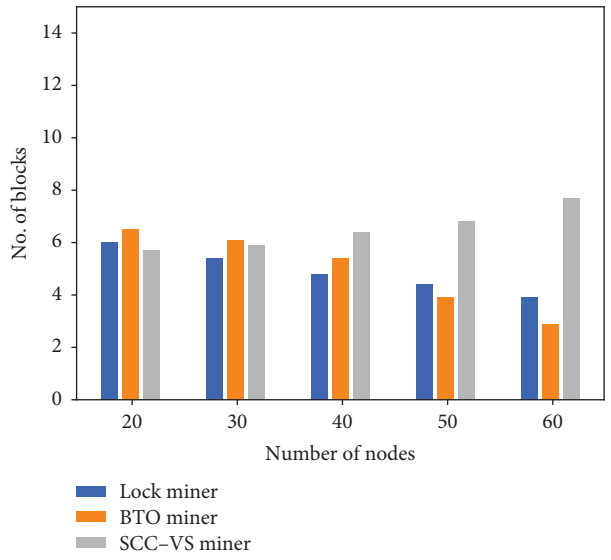


Figure 7: Throughput change with increasing number of nodes.

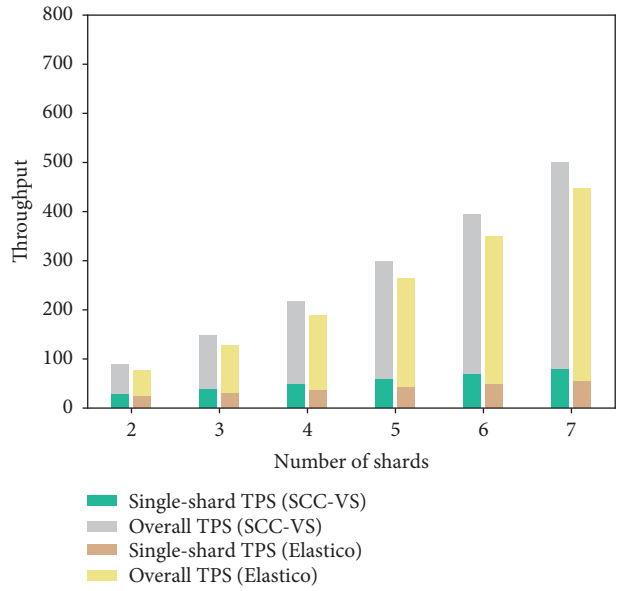


Figure 8: Single/full network TPS changes when the number of shards increases.

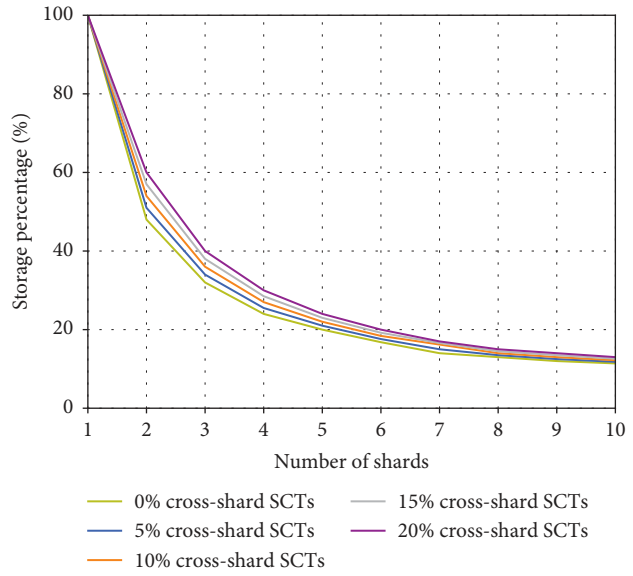


Figure 9: Storage per node decreases as the number of shards increases.

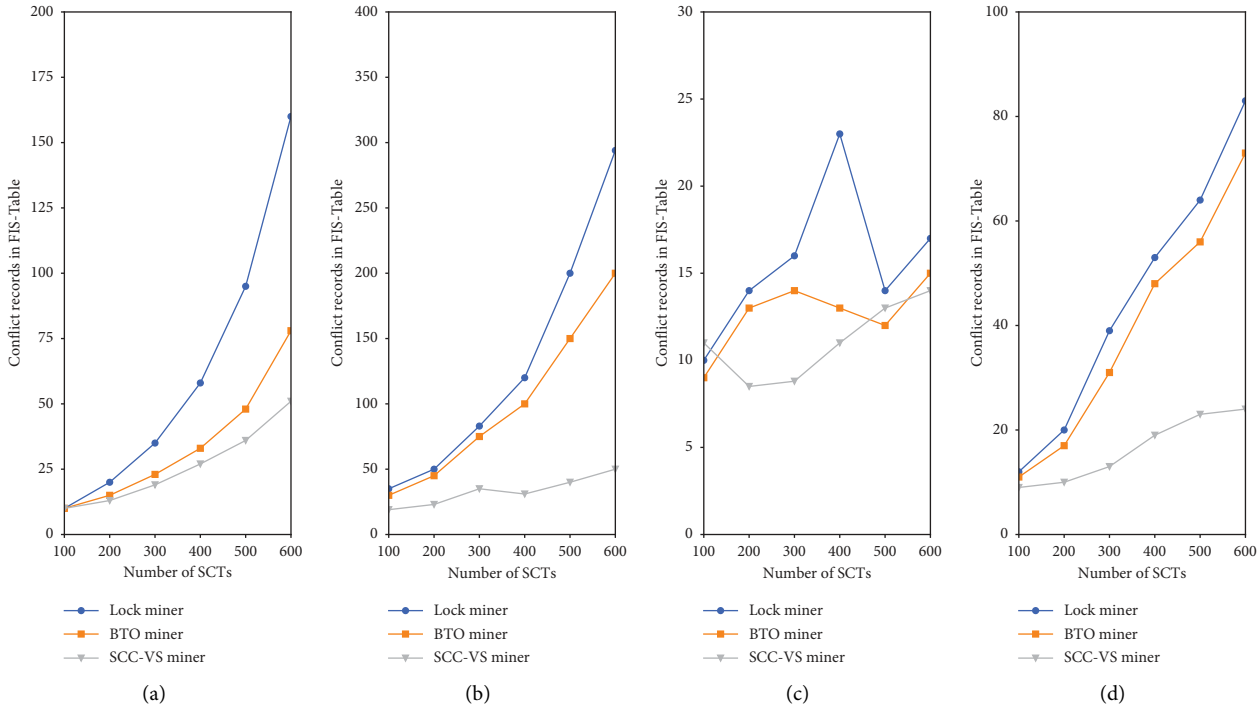


Figure 10: (Varying SCTs) Average number of conflict records in FIS-Table. (a–d) Number of SCTs.

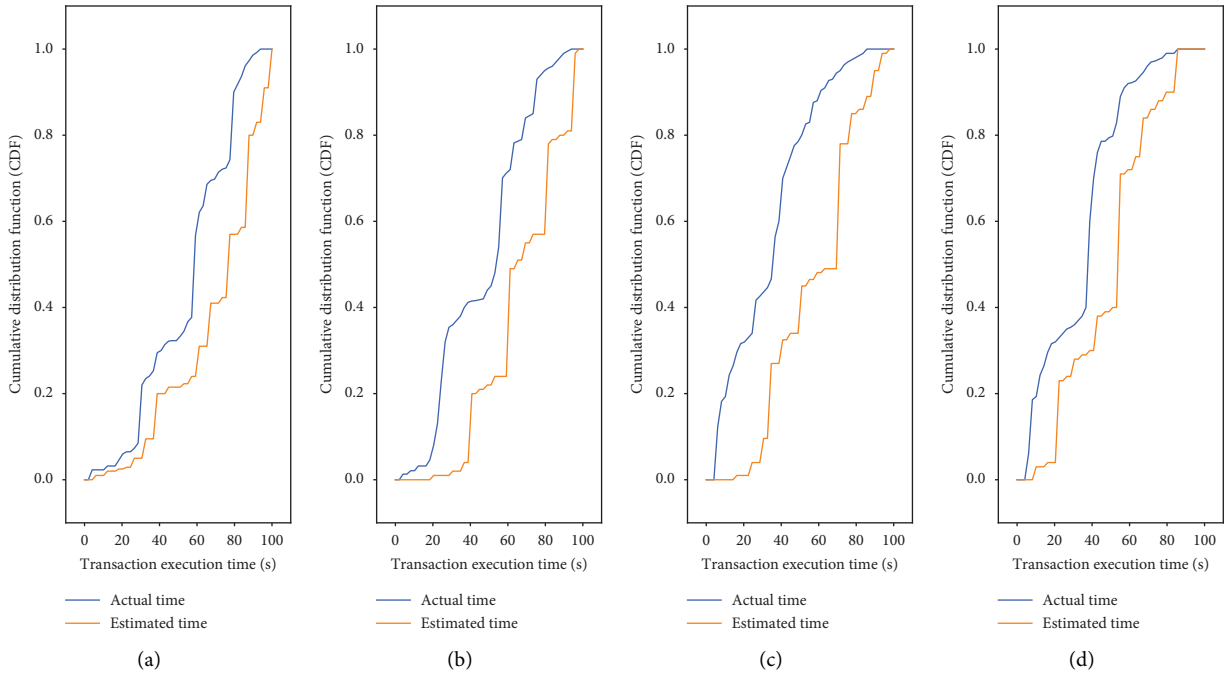


Figure 11: Distribution of estimated time and actual execution time. (a–d) Transaction execution time (s).

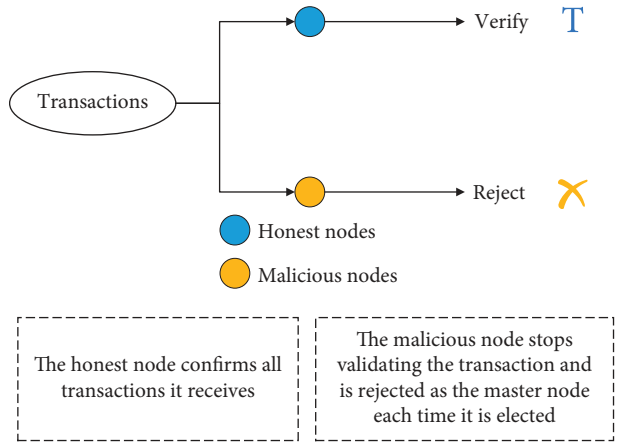


Figure 12: The behavior mode of nodes.

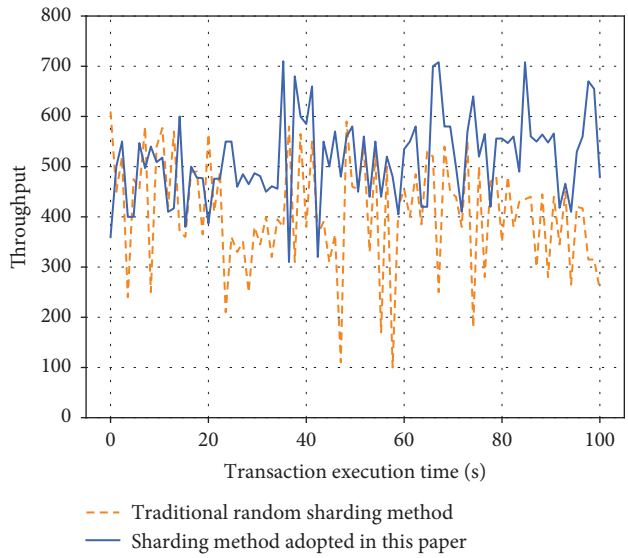


Figure 13: The results of security experiment.

6. Conclusions

In this paper, we propose a smart contract concurrent execution strategy based on concurrency degree optimization. Firstly, the CDO-Module is used to collect the feature information of conflicting SCs and carry out the concurrency degree optimization processing for the subsequent SCTs. Secondly, through the TSM-Module, the proposed SCC-VS algorithm is used to execute the SCTs after optimization. The experimental results show that the strategy ensures the execution of SCs in single shard with a higher concurrency degree, and the performance within each shard is further improved, so that the whole blockchain sharding system can meet higher transaction throughput.

Data Availability

The data used to support the findings of this study are included in this article.

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this paper.

Acknowledgments

This work was supported in part by the National Key R&D Program of China (2019YFB1406002), in part by the National Natural Science Foundation of China (61903356), and in part by Key Scientific Research Projects of Liaoning Provincial Department of Education (LZD202002).

References

[1] S. Misra, A. Mukherjee, A. Roy, N. Saurabh, Y. Rahulamathavan, and M. Rajarajan, "Blockchain at the edge: performance of resource-constrained IoT networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 1, pp. 174–183, 2021.

[2] T. Alharbi, "Deployment of blockchain technology in software defined networks: a survey," *IEEE Access*, vol. 8, pp. 9146–9156, 2020.

[3] G. Yu, T. Z. Nie, X. H. Li et al., "Distributed data management technology in blockchain system-challenges and prospects," *Chinese Journal of Computers*, vol. 12, no. 42, pp. 1–27, 2019.

[4] S. Nathan, P. Thakkar, and B. Vishwanathan, "Performance benchmarking and optimizing hyperledger fabric blockchain platform," 2018.

[5] M. Liu, R. Yu, and Y. Teng, "Performance optimization for blockchain-enabled industrial Internet of Things (IIoT) systems: a deep reinforcement learning approach," *IEEE Transactions on Industrial Informatics*, vol. 6, no. 15, pp. 3559–3570, 2019.

[6] I. Eyal, A. E. Gencer, and E. G. Sirer, "Bitcoin-NG: a scalable blockchain protocol," 2016.

[7] Y. Li, K. Zheng, and Y. Yan, "EtherQL: a query layer for blockchain system," 2017.

[8] K. Croman, C. Decker, and I. Eyal, "On scaling decentralized blockchains," 2016.

[9] C. Liu, Y. Xiao, and V. Javangula, "NormaChain: a blockchain-based normalized autonomous transaction settlement system for IoT-based E-commerce," *IEEE Internet of Things Journal*, vol. 6, no. 3, pp. 4680–4693, 2018.

[10] S. Wang, L. Ouyang, and Y. Yuan, "Blockchain-enabled smart contracts: architecture, applications, and future trends," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 11, no. 49, pp. 1–12, 2019.
[11] Q. F. Shao, C. Q. Jin, and Z. Zhang, "Blockchain technology: architecture and progress," *Chinese Journal of Computers*, vol. 5, no. 41, pp. 969–988, 2018.
[12] Y. Lian, T. Wei-Tek, and L. Guannan, "Smart-contract execution with concurrent block building," 2017.
[13] D. Jia, J. Xin, and Z. Wang, "ElasticChain: support very large blockchain by reducing data redundancy," 2018.
[14] L. Luu, V. Narayanan, and C. Zheng, "A secure sharding protocol for open blockchains," 2016.
[15] T. Dickerson, P. Gazzillo, and M. Herlihy, "Adding concurrency to smart contracts," *Distributed Comput*, vol. 33, no. 3-4, pp. 209–225, 2020.
[16] P. S. Anjana, S. Kumari, and S. Peri, "Entitling concurrency to smart contracts using optimistic transactional memory," *Computer Science*, vol. 4, no. 7, pp. 508–524, 2018.
[17] A. Zhang and K. Zhang, "Enabling concurrency concurrent execution of smart contractson smart contracts using multiversion ordering," 2018.
[18] S. Hamdi, E. Bouazizi, and S. Faiz, "A speculative concurrency control in real-time spatial big data using real-time nested spatial transactions and imprecise computation," 2017.
[19] Q. Luo and L. Zhang, "Predictable concurrency control algorithm in real-time database," *Computer Science*, vol. 31, no. 10, pp. 87–92, 2004.
[20] G. S. Yu, X. Wang, K. Yu et al., "Survey: sharding in blockchains," *IEEE Access*, vol. 8, pp. 14155–14181, 2020.
[21] C. X. Qin, B. Guo, Y. Shen et al., "A secure and effective construction scheme for blockchain networks," *Security and Communication Networks*, vol. 2020, Article ID 8881881, 2020.
[22] P. S. Anjana, S. Kumari, and S. Peri, "An efficient framework for optimistic concurrent execution of smart contracts," 2019.
[23] K. Christidis and M. Devetsikiotis, "Blockchains and smart contracts for the Internet of Things," *IEEE Access*, vol. 4, pp. 2292–2303, 2016.
[24] I. Grishchenko, M. Maffei, and C. Schneidewind, "Foundations and tools for the static analysis of Ethereum smart contracts," 2018.
[25] C.-Y. Lin, S.-H. Sheu, T.-S. Hsu, and Y.-C. Chen, "Application of generally weighted moving average method to tracking signal state space model," *Expert Systems*, vol. 30, no. 5, pp. 429–435, 2013.
[26] A. Bestavros and S. Braoudakis, "S.C.C.nS: A family of speculative concurrency control algorithms for real-time databases," 1993.
[27] Z. L. Wang, H. Jin, W. Q. Dai et al., "Ethereum smart contract security research: survey and future research opportunities," *Frontiers Comput*, vol. 15, no. 2, 2021.
[28] A. Ghaffar, M. A. Sarwar, Z. Abubaker et al., "Smart contracts for research lab sharing scholars data rights management over the Ethereum blockchain network," 2019.
[29] F. Spoto, "Enforcing determinism of Java smart contracts," 2020.
[30] H. Okagbue, M. O. Adamu, and T. A. Anake, "Closed form expression for the inverse cumulative distribution function of Nakagami distribution," *Wireless Networks*, vol. 26, no. 7, pp. 5063–5084, 2020.
[31] H. Jin, X. Dai, and J. Xiao, "Towards a novel architecture for enabling interoperability amongst multiple blockchains," 2018.
[32] Y. Dong and R. Boutaba, "Elasticoin: low-volatility cryptocurrency with proofs of sequential work," *IEEE ICBC*, vol. 2019, pp. 205–209, 2019.