Smart Contract Microservitization

Siyuan Wang, Xuehan Zhang, Wei Yu, Kai Hu, Jian Zhu*
School of Computer Science
Beihang University, Beijing 100191, China
e-mail: zhujian@buaa.edu.cn

***Abstract*—A smart contract is a computable protocol that automatically enforces contract terms in a computer, transforming real-world contract terms into digital promises of the virtual world. Early smart contracts have been stuck in the theoretical phase due to the lack of a credible execution environment and the means to control digital assets. With the emergence of blockchain technology, it has solved the problems mentioned above. Smart contracts are stored on blockchain, ensuring the credibility of contract execution through the joint execution of contracts by the various nodes in the blockchain network. However, the current technology of blockchain-based smart contracts is still not mature enough and faces many major challenges. Among them, the extensibility and performance of smart contracts are the most important and most concerned ones. This paper studies the extensibility and performance of smart contracts by combining blockchain-based smart contracts with cloud technologies to address the extensibility and performance issues of smart contracts. Combined with micro-service technology, a new type of smart contract architecture is proposed, and then the key technologies in each layer of the architecture are further studied.**

Keywords- Blockchain; BaaS; Smart Contract; Micro-service

I. INTRODUCTION

Human society is moving rapidly toward the digital society. How to transfer the economic activities in human society from the real world to the virtual world is a huge challenge. Faced with this huge challenge, Nick Szabo, an expert in cryptography, proposed the concept of "Smart Contract [1]" in 1994. He creatively proposed that "smart contracts are computable protocols that automate the execution of contract terms." Nick Szabo pointed out that computer code can replace existing machinery and is able to execute more complex digital property transactions. However, the study of smart contracts in early age have been stuck in theoretical research, and has no large-scale application. The main problems can be summarized into two aspects. One is that the smart contract lacks an effective method to controlling the assets, and the clauses of the contract-related assets cannot be guaranteed to be executed effectively. Second, the contract execution environment provided by a single computer is difficult to guarantee the terms related to these contracts can be executed properly. With the emergence of blockchain technology, problems above have been well solved.

However, the smart contract technology based on blockchain is still not mature, and there are still many problems to be solved. At present, there are two main problems:

1) *Extensibility.* With the rapid development of smart contracts, the scale of smart contract applications is gradually expanding. Similar to the problems faced by traditional software engineering [12], the development of smart contract is independent of each other and difficult to combine. The internal logic coupling of the contract is extremely high, making it difficult to re-use and expand, resulting in high contract development and maintenance costs.

2) *Performance.* Execution of a contract requires consensus [13] of the entire network, and multiple contracts within the same block need to be executed serially, resulting in excessive time overhead and poor execution performance. The performance problem of smart contracts directly limits the application of smart contracts in scenarios with real-time requirements.

At present, there are many related researches trying to solve these problems, but most of them are limited by the blockchain [2] and the smart contract technology itself, and didn't get a good result. The cumbersome construction process of the blockchain network prompted the emergence of BaaS [4]. BaaS is in the PaaS [5] layer of cloud services, providing developers with a fast blockchain development environment and reducing the cost of blockchain deployment. The emergence of BaaS technology provides not only a feasibility for the combination of smart contracts and cloud technologies, but also a new way to solve the existing problems of smart contracts. Based on the idea of BaaS, this paper solves the extensibility and performance issue by combining cloud technology with smart contract technology, we name it BaaS based smart contract. Specifically, we combine existing blockchain architecture with microservice architecture to solve problems faced by smart contracts.

Microservice architecture [3] is a new architectural concept, it uses many small functional blocks with single function and uses a modular approach to form complex largescale applications. Each microservice uses a language independent API to communicate with each other. The microservice architecture brings flexibility, extensibility and high availability to existing blockchain based smart contract architecture. We utilize the low-coupling nature of micro-services, and wrap some smart contract to smart contract micro-service, and combine complex contracts from these smart contract micro-services to solve the extensibility problem of smart contracts. Based on the research of smart contract microservices, we propose a contract parallel execution model to solve the performance issue. By implementing a smart contract service platform in the cloud,

978-1-7281-7303-0/20/$31.00 ©2020 IEEE
DOI 10.1109/COMPSAC48688.2020.00-31

the contract life cycle is effectively managed, providing a complete set of services for the development and application of blockchain-based smart contracts. Finally, with this comprehensive service, we can achieve agile development and efficient maintenance of smart contracts, and enrich the application scenarios of smart contracts, so these contracts can quickly apply in new scenarios.

II. RELATED WORK

A. Smart Contract

A smart contract is a computer contract that is digitally concluded, verified, and executed. It allows contract participants to conduct trusted transactions without the participation of a trusted third party. The history of these transactions and the status of contract execution can be tracked. And contract execution is not reversible. Early smart contracts have remained at the theoretical stage due to the lack of a credible execution environment and the means to control digital assets. Until the emergence of blockchain technology, smart contracts began to gradually move toward the implementation of the application. Although the design of smart contracts [14] and blockchains are independent, the combination of these technology ensures that the concept originally proposed by the smart contract can be realized.

Although smart contracts are rapidly developing with the support of the blockchain technology, the technology is still in its infancy, and there are still many problems to be solved. The main problems are as follows:

- The extensibility problem of smart contracts based on blockchain. Currently, the scale of smart contracts is rapidly expanding. Since the essence of smart contracts is the logical code segment running on the blockchain, it will also face the problems faced by traditional application development, the biggest one is extensibility. Currently smart contracts coupling is extremely high, and there are few mutual calls between contracts, and the reusability of contracts is extremely low, which makes the maintenance and update of contracts become cumbersome.
- Blockchain-based smart contract execution is inefficient [7]. The reason is that the smart contract execution process requires all the nodes in the blockchain network to participate. The execution of smart contract requires all the blockchain nodes to load the contract code into the contract virtual machine and reach consensus of the execution results, resulting in excessive time overhead and inefficiency.
- Security issues based on blockchain-based smart contracts [6]. Since the blockchain is essentially a decentralized ledger, the smart contracts that run on it are enforceable. Once the contract has problems, it is difficult to rollback the execution of smart contract. Therefore, if the contract has problems it will result in more serious losses than the centralized system. Currently smart contracts don't have any valid verification method, which can easily lead to code security vulnerabilities. The DAO event is a typical case of smart contract security.

B. BaaS

Cloud computing is virtualizing computing resources and providing services. These resources can be dynamically expanded and managed conveniently, which provides great convenience for users to obtain computing resources. The three mainstream cloud computing service models are IaaS [8], PaaS, and SaaS, which abstract hardware, middleware, and application software to improve the cloud service [9] ecosystem. In recent years, encrypted digital currencies such as Bitcoin [10] and Ethereum [11] have been sought after by investors. The market value of the digital money market has been pushed to a new level, and the underlying blockchain technology has received extensive attention and accelerate the development of various blockchain technologies such as public chain and alliance chain. In 2015, giants such as Microsoft and IBM proposed a strategy called blockchain as a service (BaaS), combining blockchain with cloud computing, and proposed the concept of blockchain cloud service, which is essentially a new kind of cloud service. With the new cloud services, developers can create, deploy, operate and monitor blockchain services in an efficient and easy way on the cloud service platform.

III. SMART CONTRACT MICROSERVITIZATION METHOD

At present, the main problems of smart contracts are extensibility and performance, and the emergence of blockchain cloud technology provides an opportunity for the combination of cloud technology and smart contract technology. In order to solve the problems of smart contracts with current cloud technologies, we have designed a smart contract architecture based on BaaS. Specifically, we use cloud-based micro-service technology to solve the extensibility problem of smart contracts, and use parallelization methods to solve contract performance problems. This chapter first introduces the smart contract layered architecture based on blockchain cloud, and determines the functional and non-functional requirements of each layer. Then we introduce the key technologies of the smart contract micro-service method and how to use this method to solve extensibility problem of smart contract. And we design a smart contract parallel execution model to improve the performance of contract execution. Finally, the algorithm used in the smart contract microservitization and the parallelization is described.

A. Smart Contract Architecture Based On BaaS

The smart contract layered architecture based on BaaS is mainly divided into application layer, cloud contract layer, blockchain cloud layer, smart contract microservice layer, core network layer, edge layer and rights management layer running through all levels, as shown in Fig. 1. The edge layer mainly includes a basic storage and calculation module, a module that receives the task dispatch from cloud. The core network layer mainly includes data communication and encryption modules for messages used for data transmission, providing secure and fast communication for the smart contract service layer and the blockchain cloud layer. The

blockchain cloud layer is compatible with a variety of blockchain systems, abstract blockchain basic functions, including internal functions such as encryption, building blocks, consensus, nodes, and other external functions such as transaction reception, blockchain status data query, etc. The cloud smart contract layer is based on blockchain cloud layer, which provides the function of basic smart contract lifecycle management. The smart contracts microservices layer provides contract decoupling, contract microservitization based on blockchain clouds layer and underlying smart contracts layer, and integrates functions such as contract containers management, contract combinations, and contracts parallel execution to provide efficient, scalable contract customization. The upper application layer is based on the lower blockchain smart contract service, and runs various applications based on different application scenarios. The rights management layer provides certificate authority management, identity authentication, and privacy protection for each layer

B. Contract Micronization

The smart contract is essentially the code that implements the business logic. The traditional smart contract writes all its logic in a contract; therefore, the contract is large, high coupling and can only used for certain scenario, the reusability and extensibility is very poor. Contract micronization is an important method to promote the mature development of smart contracts. How to narrow the contract logic is the first problem facing when dealing with contract micronization. We analyze the characteristics of the current smart contracts and conclude that the smart contract microservices should have the following characteristics:

- Single responsibility principle: The design of contract microservices should satisfy the principle of object orientation. To ensure the high reusability of contract microservices, the logic involved by a smart contract microservice should be single and independent.
- Low data interaction frequency: Smart contract microservices should have little data interaction with the state database in the current blockchain, that is, the input and output data of the intermediate process should be retained in the temporary environment of the contract, and should not perform data interaction with state database directly. Prevent synchronization problems due to data coupling and performance degradation caused by database read and write.
- High extensibility: After the development of the smart contract micro-service, the service can be used simply by exposing the service to other micro-contracts and users. Smart contract microservices are isolated and micro- contract collections can meet new requirements by adding new micro-contract services or updating existing micro-contract service.

C. Micro-Contract Packaging

Encapsulated the micronized contract service into an executable smart contract needs to provide a closed and secure executable environment. We use container technology to encapsulate the contract logic into a stand-alone micro-contract container, the structure of the environment is shown in Fig. 2.

It mainly contains following components:

- Contract execution module. This module includes contract language virtual machine and contract toolkit. Contract virtual machine provides the basic execution environment, including contract bytecode parsing and conversion. The contract toolkit enables efficient interaction between the micro-contract and the blockchain, and the contract container provides support for the interaction with the already packaged contract.
- Contract security module. This module provides container identity authentication and contract error handling. Container identity authentication allows

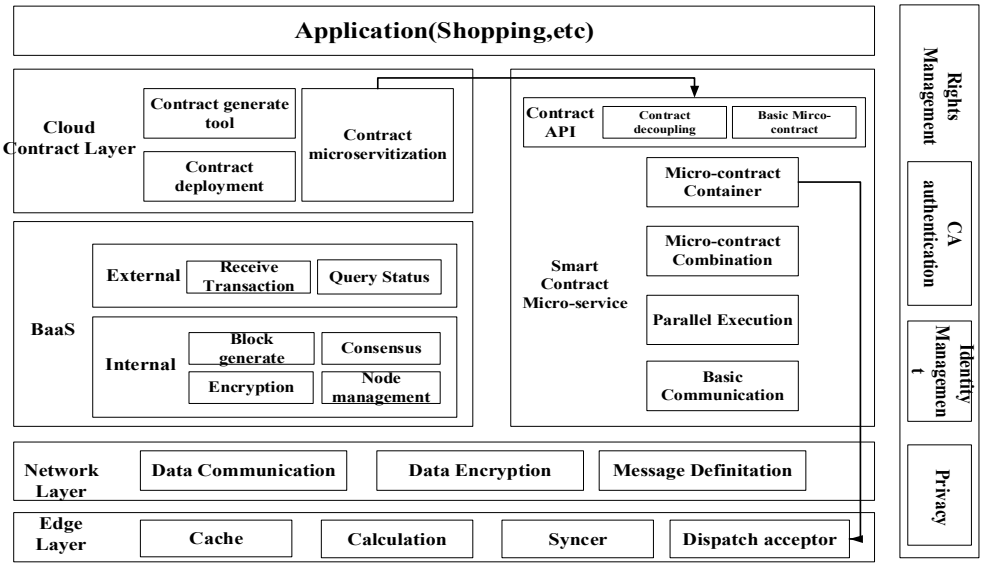

Fig. 1 Smart contract layered architecture based on BaaS

user to authenticate the visitor when accessing the container to ensure the safe execution of the micro-contract in the contract container. Contract error handling is to deal with transaction execution errors, contract container errors, and malicious container attack errors encountered during container execution.

- Communication module. This module supports internal communication between contract microservice.
- Data encryption module. This module ensures the security of data transmission between micro-contract containers by using digital signature.

D. Micro-Contract combination

The micro-contracts implement the logic of a single, independent business, but the business logic in the actual scenario is usually complex and coupled, so it is necessary to combine micro-contracts to achieve business requirements. We first define the transaction format based on the smart contract microservice architecture:

From: The sender of the data or value is represented by an address containing 20 bytes. Field name is T_f , therefore $T_f \in \mathbb{B}_{20}$ or $T_f = \emptyset$.

To: The recipient of the data or value is also represented by an address containing 20 bytes; this field needs to be empty for transactions which create smart contract microservices. Field name is T_t, therefore $T_t \in \mathbb{B}_{20}$ or $T_t = \emptyset$.

Type: The type of transaction, which distinguishes between ordinary transactions and smart contract microservice call transactions, is represented by an 8-bit binary positive integer. Field name is T_p. Therefore $T_p \in \mathbb{P}_8$, $\mathbb{P}_n$ represent n-digit positive integer.

Sign: The signature of the transaction, represented by a 32- byte signature; this field is used to identity the transaction initiator. Field name is T_s, and $T_s \in \mathbb{B}_{32}$.

Nonce: The sender has sent the number of transactions to prevent transaction replay attacks. Represented by a 32-bit binary positive integer, field name is T_n, and $T_n \in \mathbb{P}_{32}$.

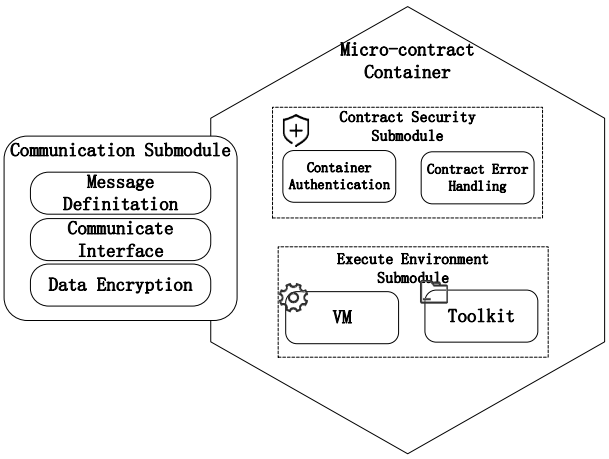

Fig. 2 Micro-contract container structure

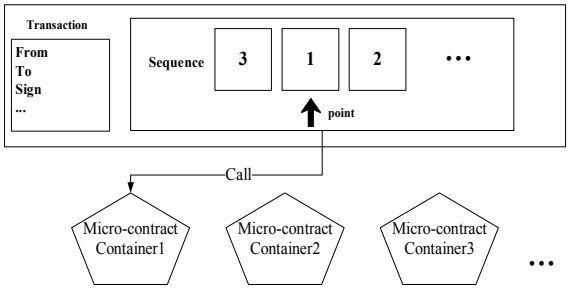

Fig. 3 Micro-contract based transaction's execution process

Value: The value of the transaction transfer, represented by a 64-bit binary positive integer. Field name is T_v and $T_v \in \mathbb{P}_{64}$.

Result: The execution result of the transaction, defined using a status code, represented by an 8-bit binary positive integer. Field name is T_r and $T_r \in \mathbb{P}_8$.

Timestamp: The timestamp of transaction, file name is T_s.

Data: Transaction data, if it is a creation transaction of a smart contract microservice, is the byte stream of the contract code. Field name is T_d.

Sequence: The order of the micro-contracts is invoked. After the combination contract is created, the sequence is generated. When user call the contract, each micro-contract logic is invoked in the sequence. Field name is T_{seq} and $T_{seq} \in \mathbb{P}_8$.

Point: A contract sequence pointer that points to the container number of the micro contract currently in use. Field name is T_p and $T_p \in \mathbb{P}_8$.

ContainerSign: The container is signed as an identifier to ensure that the container which the micro contract is located is secure. Field name is T_{csign} and $T_{csign} \in \mathbb{B}_{32}$.

Therefore, a transaction can be represented by $T = \langle T_f, T_t, T_p, T_n, T_v, T_r, T_d, T_s, T_{cs}, T_{seq} \rangle$.

As in shown in Fig. 3, a combination contract invocation transaction is executed, it uses sequence field to identify the call sequence of each micro-contract container. And the point field to get the micro-contract container which is calling now, and use it to get the transaction execution status. Since the transaction will be executed by multiple micro-contract container, so we use ContainerSign field to ensure the security of execution.

IV. PARALLEL EXECUTION MODEL BASE ON MICRO-CONTRACT

The reason why parallel execution model has a good application effect on the existing blockchain-based smart contracts is that current smart contract logic coupling is too high, and contract involves many shared variables. To ensure the security of contract execution, in most cases contract execution is downgraded to serial execution, resulting in extremely limited performance. The smart contract microservices divide the existing contract logic into pieces and encapsulate it in different contract containers. The number of shared variables in a single contract container is greatly

reduced, which greatly improves the performance of the contract execution.

V. EXPERIMENT

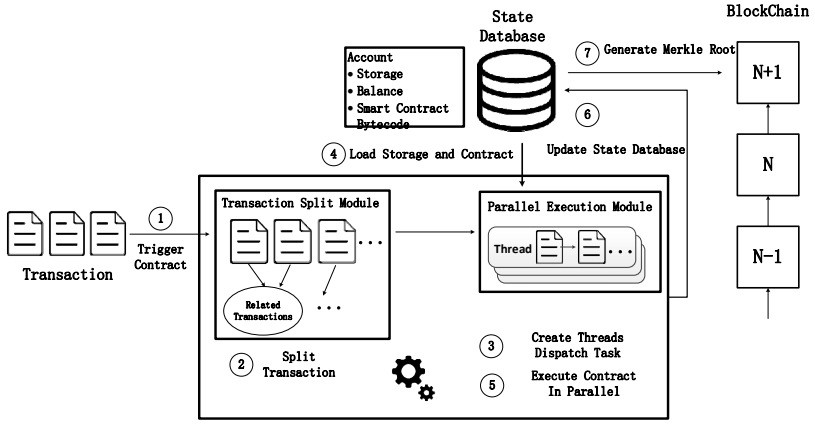

Fig. 4 Micro-contract based smart contract parallel execution model

To study the parallel execution model based on micro-contract, we first analyze the Ethereum smart contract execution model, summarizes the performance bottleneck in its model, then we design a parallel execution model base on micro-contract.

Compared to the Ethereum's smart contract model, we add two new modules. First is the transaction split module, this module processes a set of incoming transactions and split them into multiple groups, and within each group each transaction won't use any shared variable with others. Second is parallel processing module, this module will generate multiple threads to processing transactions in parallel base on work load.

Fig. 4 shows the smart contract parallel execution model. In this model, the main process of contract execution can be divided into 7 steps:

1. Query the current transaction pool to be processed, get the header transaction, and call its corresponding smart contract

2. Transaction splitting module analyzes each transaction to get information about shared variables. Then divided these transactions into different groups, within each group, no transaction will use same shared variables with others. Then these transactions will send to parallel execution module.

3. The parallel execution module assigns these transactions to worker thread.

4. Worker thread obtain contract data from the state database, such as contract code, status information, etc.

5. Worker thread execute contract code to finish business logic.

6. Contract execution result is written back to state database.

7. After all the smart contracts are executed, the blockchain system will extract a digest of state database and then record transactions and extracted status summaries into the blockchain.

First, we will introduce the preparations required before the experiment, including the construction of the experimental test environment, then perform the function and efficiency evaluation of the smart contract micro-service method; then perform the performance comparison test on the micro-contract-based smart contract parallel execution model.

A. *Experiment environment*

The test environment blockchain system needs to meet the Byzantine fault tolerance requirement. That is, in the environment with f error nodes, the total number of nodes must greater than if. In this experiment, we use minimum f 1, and the test environment have 4 nodes. In order to reduce the interference of the communication network and other factors to the experiment, 4 servers are deployed in the same LAN.

B. *Experiment and Analysis of Smart Contract Parallel Execution Model*

First, we will demonstrate that the time cost of the transaction split algorithm is positively linearly related to the number of shared variables included in the transaction and the square of the number of transactions contained in each block. Then we compare the performance of the smart contract parallel execution model with the serial execution model. Table 1 shows the default values for each variable involved in the experiment.

TABLE 1 VARIABLE DEFAULT VALUE

Type	Default value
Transaction per block	2000
Transaction correlation	10%
Shared variable in one transaction	5
Thread number in parallel execution module	12

To demonstrate the time complexity formula of the transaction split algorithm, the following experiment was performed to assist in the subsequent improvement of the model algorithm. We obtain its relationship with time overhead by changing the number of shared variables contained in each transaction.

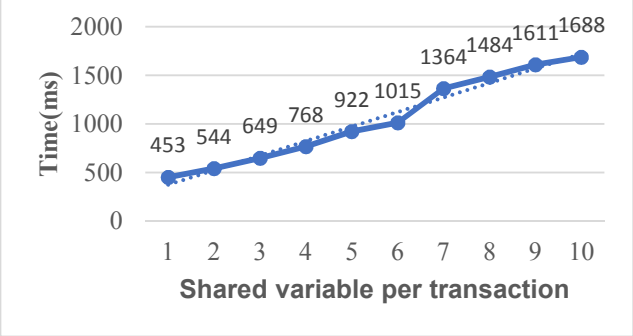

Fig. 5 Relation between shared variable per transaction and time

As shown in Fig. 5, it can be proved that the time cost of processing the transaction is positively related to the number of shared variables included in the transaction. Based on this experiment, this paper does the same experiment on the number of transactions included in the block and the time cost of processing the transaction. The experimental results are shown in Fig. 6.

Based on the current model, a serial smart contract model is designed and compared with the parallel model. Let the two models process the same transaction and compare their processing time overhead. The result is shown in Fig. 7.

According to the comparison of the two models. The results show that the parallel model can save at least 23.8% of the time overhead (in this case each block contains 3,500 transactions), and can save up to 41.9% of the time overhead

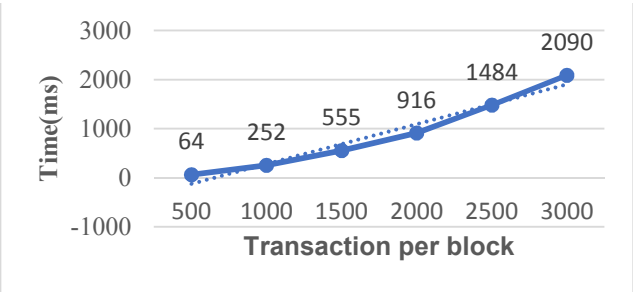

Fig. 6 Relation between transaction per block and time

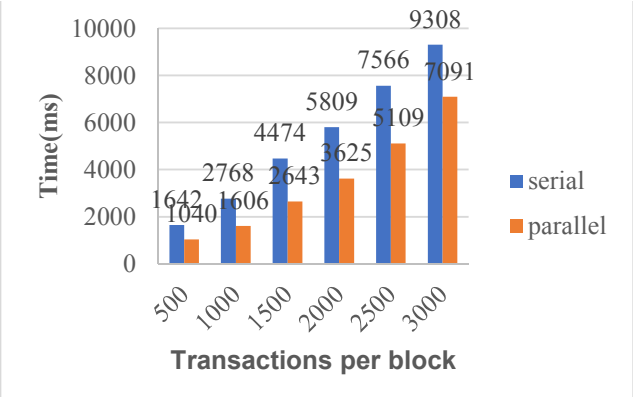

Fig. 7 Processing time for serial execution and parallel execution

(in this case each block contains 1000 transactions). By comparing the result, this paper can finally prove that the parallel model can effectively solve the problem of smart contract performance, and can effectively enrich the application scenarios of smart contracts.

VI. Conclusion

Aiming at the current scalability and performance problems of smart contracts, a new smart contract fusion architecture based on BaaS and micro-service is designed, where we have designed a new layered architecture and key technologies between the layers. Specifically, at the smart contract microservice layer, the contract microservices approach and contract parallelization techniques were proposed to improve contract scalability and performance. Subsequently, the design implements a blockchain cloud-based intelligent contract service platform, integrates the above technologies, and provides complete contract services.

Acknowledgment

This work was partially supported by the National Natural Science Foundation of China under Grant 61672074, 61672075, Project of National Key Research and Development of China under Grant 2018YFB1402702, Funding of Ministry of Education and China Mobile MCM20180104.

References

[1] Clack C D, Bakshi V A, Braine L. Smart Contract Templates: foundatio ns, design landscape and research directions[J]. 2017.

[2] Swan M. Blockchain: Blueprint for a New Economy[M]// Blockchain : blueprint for a new economy. 2015.

[3] Familiar B. Microservice Architecture[J]. 2015.

[4] D. Blockchain as a Service for IoT[C]// IEEE International Conference o n Internet of Things. 2017.

[5] Lawton G. Developing Software Online With Platform-as-a- Service Technology[J]. Computer, 2008, 41(6):13-15.

[6] Arnett M. Step by Step Towards Creating a Safe Smart Contract: Lesson s and Insights from a Cryptocurrency Lab[J]. 2016.

[7] Cook V, Painter Z, Peterson C, Dechev D. Read-Uncommitted Transactions for Smart Contract Performance[J]. 2019.

[8] Prodan R, Ostermann S. A survey and taxonomy of infrastructure as a se rvice and web hosting cloud providers[C]// IEEE/ACM International Conference on Grid Computing. 2009.

[9] Tang L.J., Jing D,Zhao Y.J., Zhang L.J.. Enterprise Cloud Service Architecture[C]. In: 2020 IEEE International Conference on Cloud Computing.

[10] Nakamoto S. Bitcoin: A peer-to-peer electronic cash system[J]. 2008.

[11] Wood G. Ethereum: A secure decentralised generalised transaction ledge r[J]. Ethereum project yellow paper, 2014, 151(2014): 1-32.

[12] Pressman, Roger S. Software Engineering: a Practitioners Approach[J]. 2001.

[13] Olfatisaber R, Fax A, Murray R M. Consensus and Cooperation in Netw orked Multi-Agent Systems[J]. Proceedings of the IEEE, 2007, 95(1):215-233.

[14] Clack C D, Bakshi V A, Braine L. Smart Contract Templates: essential requirements and design options[J]. 2016.