A Parallel Smart Contract Model

Wei YU
State Key Laboratory of Software Development Environment, Beihang University, Beijing 100191, China
(86)18811437365
964135354@qq.com

Kan LUO
State Key Laboratory of Software Development Environment, Beihang University, Beijing 100191, China
(86)18810285862
looken@buaa.edu.cn

Yi Ding
School of information, Beijing Wuzi University, Beijing 101149, China
(86)010-89534291
dingyi@bwu.edu.cn

Guang YOU
BeiJing Business Support Center, China Mobile Information Technology Co.,Ltd, Beijing 100031, China
(86) 13810354684
youguang@cmhi.chinamobile.com

Kai HU
State Key Laboratory of Software Development Environment, Beihang University, Beijing 100191, China
(86)010-82339460
hukai@buaa.edu.cn

ABSTRACT
With the rapid development of blockchain technology, blockchain becomes a good platform for execution of smart contracts. However, since smart contracts still have a low performance of transaction processing on blockchain. It can't satisfy real-time requirements in some situations. This paper proposes a parallel smart contract model on blockchain which has a better performance in transaction processing. The challenges with the proposed approach are the implementation of the parallel mode and the solution of synchronization problem of the proposed model. This paper uses multi-thread technology to implement the proposed model where transactions are executed in parallel. Then we propose a transaction splitting algorithm to resolve the synchronization problem. Finally, experimental analysis proves that this parallel model exactly makes a remarkable development of performance in transaction processing.

CCS Concepts
•Theory of computation→Models of computation→Concurrency →Parallel computing models.

Keywords
Blockchain; smart contract; parallel model.

1. INTRODUCTION
The concept of smart contract was firstly proposed by Nick Szabo in 1994 [1]. This concept implies that the smart contract is a computable trading protocol which can automatically execute the terms of the contract. Although smart contract theory is almost simultaneously proposed with the Internet technology (World Wide Web), the application practice of smart contract has been seriously behind the theory due to the lack of a clear path to realize the implementation of smart contract. Smart contracts mainly face two problems. One is that smart contracts don't have effective means to control physical assets and ensure the execution of contracts; the other is that it is difficult for a single computer to guarantee the implementation of these terms in a smart contract therefore the smart contract can't obtain the trust of contractors.

The emergence of blockchain [2] technology has resolved these problems. On the one hand, the blockchain can store digital assets (such as money or stocks) [3], thus smart contracts running on blockchain can get the control of those digital assets. On the other hand, because the blockchain has characteristics of full traceability and non-tampering so it provides smart contracts safety and trustful execution platform where smart contracts can get the trust of contractors.

However, the existing blockchain-based smart contract technology is still in a primary stage. There are still many problems needing to be resolved. An important problem is the low performance. Recent data shows that the execution time of a smart contract on blockchain can reach over 20s and the number of smart contract running on the blockchain has enormously been increasing [4] day by day. Which means that low performance of the smart contract will directly reduce the speed of transaction processing of blockchain and limit the implementation area of blockchain.

Always, smart contracts are invoked to process the transactions on blockchain and the process of transaction processing will be called as smart contract model.

This paper proposes a new model of the smart contract. It uses multi-thread technology [5] to execute smart contracts in parallel. Using this new model to process transactions can reduce the average time cost and make smart contracts get a better performance on blockchain.

1.1 Contributions
This paper makes the following contributions:

- We analyze a typical current smart contract model and summarize disadvantages of it.
- We propose a parallel smart contract model which has a better performance in transaction processing.
- We propose a transaction splitting algorithm to resolve the synchronization problem in the parallel smart contract model.

MLMI2018, September 28–30, 2018, Ha Noi, Viet Nam.

ACM ISBN 978-1-4503-6556-7/18/09…$15.00
DOI: https://doi.org/10.1145/3278312.3278321

- We test the performance of the algorithm and the parallel model that we proposed.

1.2 Organize

This paper is organized as follows:

Section 2 presents the related work; Section 3 describes the parallel smart contract model that we proposed; Section 4 introduces a transaction splitting algorithm to resolve the synchronization problem; Section 5 describes experimental analysis and Section 6 gives the conclusion.

2. RELATED WORK

There are some popular blockchain open source projects which support smart contracts. We will first analyze the smart contract model of these projects, then find out which part of the models should be modified to help us design our new smart contract model.

2.1 Components

Before we show our analysis results, some necessary components of smart contract model should be introduced first.

2.1.1 State database

State database stores all state variables, including contract code, contract storage and so on. Usually, the contract code is saved as bytecode in a low-level language. Users always make a contract by using a high-level programming language such as java, go, or Solidity [6], then compile the code by a specific compiler. For example, Ethereum users use Solidity to write their own contract, then use Ethereum Virtual Machine (EVM) to compile it into contract code [7]. Contract storage contains some variables that related to this contract. For example, a simple transfer contract storage includes the account balance and transfer amount.

2.1.2 Transaction

Usually, smart contracts are invoked by transactions. A transaction includes the name of invoking smart contract and input data for the invocation.

2.1.3 Contract executing virtual machine

Contract executing virtual machine provides smart contracts with the necessary runtime environment and compiles the contract into contract code. Moreover, when a contract is invoked by a transaction, contract executing virtual machine will execute the contract code and give the execution result.

2.1.4 Blockchain

Blockchain records all the transactions and their execution results given by smart contracts. It also records the certification of the correctness of state database.

2.2 Current Smart Contract Model

In this paper, we mainly analyze a current popular open source project: Ethereum [8], [9]. Ethereum is a public blockchain which supports the smart contract.

2.2.1 Ethereum

The Figure 1 shows the smart contract model of Ethereum. A simple process of smart contract execution can be described as follows:

- A transaction that is going to be processed invokes its corresponding contract.
- By analyzing the content of the transaction, EVM gets the corresponding contract code and contract inputs from the transaction and state database.
- Contract code is executed in EVM and the alterations of state variables will be written back into the state database.
- When all transactions in a block have already been executed, the current state of state database will be recorded in the blockchain as a certification.

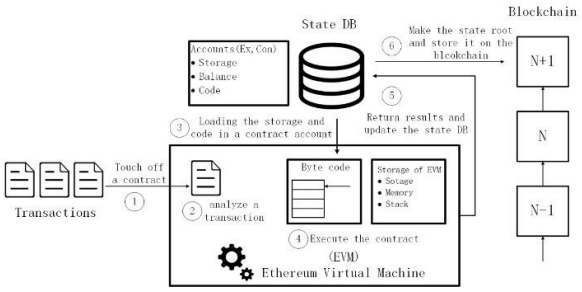

Figure 1. Ethereum smart contract model.

Ethereum uses some methods to improve the performance of its smart contract mode. First, it uses LevelDB [10] which has a high performance of random writing to reduce the I/O cost. Second, all state variables are stored in MPT [11] (Merkle Patricia tree) to reduce the cost of making the certification. Moreover, it proposes a tree pruning method [12] to cut useless data in the state database.

2.2.2 Disadvantages and Modifications

However, according to our analysis, there are some disadvantages in Ethereum. Its smart contract model is a serial mode, every operation that is going to be executed should wait for the end of previous work which will cause more time cost. Especially in the step number 3, 4 and 5 of contract process in Figure 1, there are many I/O operations executing in serial, which can sharply reduce the performance.

As a result of the description above, the current smart contract models have some disadvantages which will finally cause the decline of performance. By analyzing these disadvantages, we think that executing contract in serial is the main cause of the decline of performance. Then we propose a new model which executes contract in parallel and get a better performance in transaction processing.

3. PARALLEL SMART CONTRACT MODEL

According to the analysis of current smart contract model, we propose a parallel smart contract model which has a better performance of transaction processing.

3.1 Components

Compared to current smart contract model, there are two newly added model components:

3.1.1 Transaction splitting component

Transaction splitting component will group the transactions and make each transaction group have no shared variables.

3.1.2 Parallel processing component

Parallel processing component uses multi-thread technology to process transactions in parallel. It creates an appropriate number of threads and assigns contract executing tasks to each thread.

Other components such as state database and so on have little difference between Ethereum.

3.2 Processing Steps

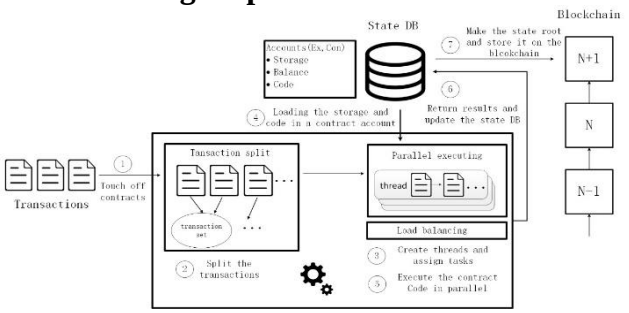

Figure 2. Parallel smart contract model.

The Figure 2 shows the parallel smart contract model that we proposed. The main process of contract execution in this new model can be described in seven steps:

Step1: When the blockchain system gets enough valid transactions from transaction pool, it will start processing these transactions.

Step2: Transaction splitting component analyzes the transactions to get the information of shared variables. Then, this component groups the transactions into different set which have no same shared variables with each other. Finally, these transaction sets will be sent to multi-thread processing component.

Step3: multi-thread processing component assigns these transaction processing work to each thread.

Step4: Threads start running and get necessary initial data such as contract code from state database. Then, the contract invoked is ready to be executed.

Step5: Contract code will be executed and then the code will complete its corresponding smart contract's business logic.

Step6: Smart contract executions will alter some related state variables and these alterations will finally be written back to state database.

Step7: When all smart contracts have been finished, the blockchain system will make the certification of state database (i.e., in Ethereum, the certification of state database is the root of MPT), then record all the processed transactions and the certification into the blockchain.

3.3 Performance Analysis

Although new components will generate an extra time cost in step 2 and step 3, processing transactions in parallel will save a huge time cost in step 4, step 5, and step 6. Moreover, the extra time cost is most generated by extra computation operations in the new components. The saved time cost is generated by not only computation operations but also the I/O operations. Since I/O operations always cost more time than computation operations, the parallel model will finally save much time in transaction processing and get a better performance than the current model. We will prove our analysis according to the experiment in section 5.

4. TRANSACTION SPLITTING ALGORITHM

Using multi-thread technology to process transactions in parallel will bring improvement on the performance, but it will also cause the synchronization problem [13]. We propose a transaction splitting algorithm to resolve this problem and it will be used in the transaction splitting component.

4.1 Descriptions of Transaction Splitting

When we used multi-thread technology, the synchronization problem should be taken into account. This problem implies that if there is no protection for a variable that is shared with different threads, the value of this variable may be different in different threads and lead some errors.

As we described above, the synchronization problem is caused by the same shared variables which exist in different transactions, we define these transactions as related transactions. Then we can get this conclusion that if the transactions in different threads are not related, there will be no synchronization problem. So, the solution to the synchronization problem is to split the transactions and let different threads process unrelated transactions. To help achieve the aim of transaction splitting, we give these definitions as follows:

We define a transaction as a two-tuples:

$$T = (I, S)$$

Among them:

- I is the information of this transaction;
- $S = \{s_1, s_2, \cdots, s_m\}$. S is the set about all shared variables of this transaction;

There are three states between two transactions as follows:

- Unrelated: $S_1 \cap S_2 = \emptyset \quad (S_1 \in T_1, S_2 \in T_2)$
- Related: $S_1 \cap S_2 \neq \emptyset \quad (S_1 \in T_1, S_2 \in T_2)$
- Potentially Related: $S_1 \cap S_2 = \emptyset, S_1 \cap S_3 \neq \emptyset, S_2 \cap S_3 \neq \emptyset \ (S_1 \in T_1, S_2 \in T_2, S_3 \in T_3)$

Unrelated means these two transactions have no the same shared variables and related means they at least have a same shared variable. Potentially related means these two transactions have at least a same related transaction. Moreover, Related and potentially related transactions should be split into the same set.

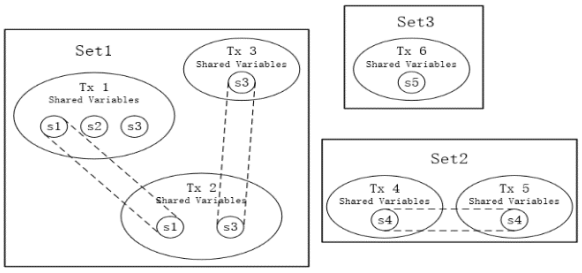

Figure 3. Process of transaction splitting.

The Figure 3 shows an example of the process of transaction splitting. Transaction Tx1, Tx2 have the same variable s1 and Tx2, Tx3 have the same variable s3. Tx1 has a potential relationship with Tx3 due to Tx2, so Tx1, Tx2 and Tx3 are grouped into set 1. Then Tx4, Tx5 also have the same variable, so they are grouped into set 2. Due to Tx6 has no same shared variable with other transactions, it is grouped into set 3 alone.

4.2 Algorithm Design

After we get the process of transaction splitting, now we will give the algorithm design of it as follows.

4.2.1 Definitions

Before we introduce the proposed algorithm, the definitions in the algorithm should be introduced first.

- The set we used in the pseudo-code does not contain duplicate elements.
- The tx is short for a transaction, the txi and txj is short for different transactions. The variable T means all transactions means total transactions of one block.
- The related set of one transaction contains its related transactions.
- The related degree means the degree of the relationship among all transactions, it can be expressed as a formula where variable n means the number of all transactions and variable m means the number of relationships among all transactions. The formula is shown as follows:

$$related\ degree = m/n^2 \ \ (\mathrm{m} \in \mathrm{N}, \mathrm{n} \in \mathrm{N})$$

4.2.2 Creating related sets

Before splitting transactions, first we should get the related set of each transaction. With the help of the related sets, we can put the related transactions into the same subset.

Table 1. Algorithm: Create related transaction sets

```
Input: T (a set of all transactions)
1   create a set R
2   forall the txi in T do
3   create a related set S for txi
4   forall the txj in T do
5    if isRelated(txi,txj)=true then
6     S ← txj
7    R ← S
8   return R
```

Table 1 shows the pseudo-code, where the function named isRelated is used to judge whether two transactions were related. By inputting the transaction set, we can finally get the related sets.

4.2.3 Transaction splitting

After we get the related sets of each transaction, we can easily convert the relationships among all transactions to an undirected graph. Then we design a splitting algorithm based BFS [14] (breadth-first search algorithm) to split related transactions into the same subset. Table 2 shows the pseudo-code. By inputting the transaction set and related sets, we can finally get the unrelated sets of transactions.

Table 2. Algorithm: Splitting transactions

```
Input: T (a set of all transactions), R (a set of related sets)
1   create set N
2   while T is not empty
3    get a transaction tx from T
4    create a set Q
5    Q ← tx
6    while Q is not empty
7     create a set M
8     get a transaction txi from Q
9     M ← txi
10   remove txi from T
11   remove txi from Q
12   forall txj in related set of txi do
13    Q ← txj
14   N ← M
15  return N
```

4.2.4 Time complexity of algorithm

The time complexity of these algorithms that we design will help us make a better experiment to test these algorithms and modify them in the future.

The time complexity formula of creating related sets is shown as follows:

$$\mathrm{O}\left(\sum_{i\geq 1, j\geq 1}^{i\leq n, j\leq n} Xi + Xj\right) = O\left(2n \sum_{i\geq 1}^{i\leq n} Xi\right) \ \ (\mathrm{n} \in \mathrm{N}, \mathrm{Xi} \in \mathrm{N})$$

Variable n means the number of all transactions and Xi means the number of shared variables belong to ith transaction. Using a set merging method, the time complexity of isRelated function is $\mathrm{O}(\mathrm{Xi} + \mathrm{Xj})$.

The time complexity of formula transaction splitting is shown as follows:

$$\mathrm{O}(\mathrm{n} + \mathrm{e}) = O\left(n + {}^1\!/_2 \sum_{i\geq 1}^{i\leq n} Yi\right) \ (\mathrm{n} \in \mathrm{N}, \mathrm{Yi} \in \mathrm{N})$$

Variable n means the number of all transactions, Yi means the number of related transactions belong to the ith transaction. The transaction splitting algorithm is based on BFS algorithm. It also has the same time complexity with BFS algorithm which is $\mathrm{O}(\mathrm{n} + \mathrm{e})$.

Then, since the related degree of transactions is low, we can get the time complexity formula of the whole algorithm as follows:

$$O(2n^2 * average) \ \ (\mathrm{n} \in \mathrm{N}, \mathrm{average} \in \mathrm{N})$$

Variable average means the average number of shared variables of each transaction and variable n expresses the number of all transactions.

Finally, according to the final formula, we get a conclusion including two parts:

- The transaction splitting algorithm's time cost has a positive linear correlation with the number of shared variables which belong to these transactions.
- The transaction splitting algorithm's time cost has a positive linear correlation with the number of transactions in each block.

5. EXPERIMENTAL ANALYSIS

To verify the parallel model, we build our own blockchain system which can support the execution of smart contracts. The Multi-threads are implemented in Java. The testing smart contracts mainly contain several simple read and write operations of shared variables. The testing transactions mainly have two parts including the name of invoked contract and the set of their shared variables.

Our goals include twofold. First, we aim to get the proof that the transaction splitting algorithm's time cost has a positive linear correlation with the number of transactions and shared variables. Second, we will prove that our parallel smart contract model has a better performance than the current smart contract model.

5.1 Default Variables in the Experiments

Table 3 shows the default values of each variable in the experiments. In our experiments, the evaluation of performance is mainly based on the total time cost. The less time the testing object costs, the better performance it has.

Table 3. Defaults of each variable in the experiments

Number of transactions in each block	2000
Transaction related degree	10%
Number of average shared variables in each transaction	5
Number of threads to be created in the parallel mode	12
Number of computer cores	8

5.2 Transaction Splitting Algorithm

Since the process of transaction splitting has an important factor affecting performance, finding out the variables which influence this algorithm's time complexity will help us modify the algorithm in the future. As a result of that, we do these experiments as follows.

We change the value of the number of shared variables in each transaction to find out its relationship with time complexity, the result is shown as Figure 4.

The trendline in Figure 4 shows that the cost time has a nearly positive linear correlation with the number of shared variables in each transaction which can prove the part one of our conclusion in section 4.

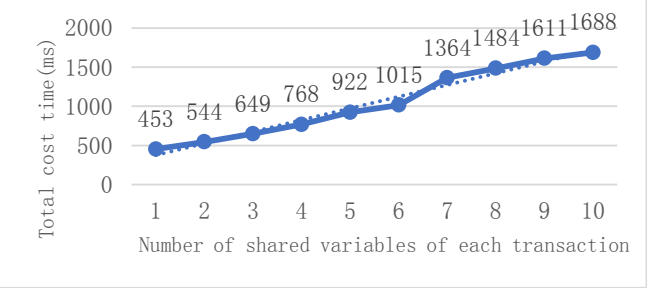

Figure 4. Relationship one.

Like the pervious experiment, we do the similar experiment of the number of shared variables. The result is shown as Figure 5. According to the trendline in Figure 5, it shows that the cost time has a nearly positive linear correlation with the number of transactions in each block which can prove the part two of our conclusion in section 4.

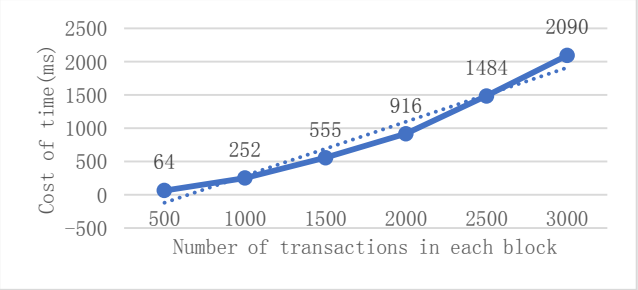

Figure 5. Relationship two.

Finally, these two experiments can prove the whole conclusion in section 4, which will help us to effectively modify the algorithm in the future.

5.3 Parallel Smart Contract Model

In this experiment, we design a serial smart contract model which is similar as the current smart contract model to be compared with the parallel model. We let the two models process the same transactions and compare their processing time cost. The result is shown as Figure 6.

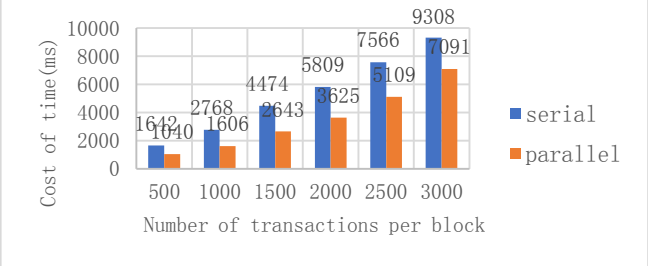

Figure 6. The time cost comparison between parallel mode and serial mode.

According to the comparison between these two modes. The result shows that the parallel smart contract model we proposed has a better performance than the serial smart contract model indeed. Furthermore, it can at least save 23.8% in time cost where each block contains 3500 transactions and can at most save 41.9% in time cost where each block contains 1000 transactions. As a result of that, we can finally prove that the proposed parallel model has a remarkable performance improvement than the current serial smart contract model.

6. CONCLUSION

This paper first introduces the current performance issues in the smart contracts running on blockchain. It analyzes the current smart contract models and summarize their disadvantages in performance, then it proposes a parallel smart contract model which avoids the disadvantages analyzed in current smart contract model. Next, it proposes a transaction splitting algorithm to resolve the synchronization problem in this proposed model. Finally, experimental results show that the proposed parallel model has a remarkable performance improvement indeed.

However, there many problems to be solved. For examples, the methods of getting shared variables in each transaction need to be given focus in terms of study.

7. ACKNOWLEDGMENTS

This work was partially supported by the National Natural Science Foundation of China under Grant 61672074 and 91538202, Funding of Ministry of Education and China Mobile MCM20160203.

8. REFERENCES

[1] Szabo N. *Formalizing and Securing Relationships on Public Networks*[J]. 1997, 2(9).

[2] Swan M. *Blockchain: Blueprint for a New Economy*[M]. O'Reilly Media, Inc. 2015.

[3] Yu L, Tsai W T, Li G, et al. *Smart-Contract Execution with Concurrent Block Building*[C]// Service-Oriented System Engineering. IEEE, 2017:160-167.

[4] Luu L, Chu D H, Olickel H, et al. *Making Smart Contracts Smarter*[C]// ACM Sigsac Conference on Computer and Communications Security. ACM, 2016:254-269.

[5] Arora, Nimar S, Blumofe, et al. *Thread scheduling for multiprogrammed multiprocessors*[J]. Theory of Computing Systems, 2001, 34(2):115-144.

[6] Ethereum Foundation. *The solidity contract-oriented programming language*. https://github:com/ethereum/solidity.

[7] Delmolino K, Arnett M, Kosba A, et al. *Step by Step Towards Creating a Safe Smart Contract: Lessons and Insights from a Cryptocurrency Lab*[M]// Financial Cryptography and Data Security. Springer Berlin Heidelberg, 2016.

[8] Ethereum*: A Next-Generation Generalized Smart Contract and Decentralized Application Platform.* https://github.com/ethereum/wiki/wiki/White-Paper.

[9] Gavin Wood. *Ethereum: A secure decentralized transaction ledger*. http://gavwood.com/paper.pdf,2014.

[10]J. Dean and S. Ghemawat, *LevelDB.* http://code.google.com/p/leveldb,2011.

[11] *Merkle Patricia Trie Specification (also Merkle Patricia Tree).* https://github.com/ethereum/wiki/wiki/Patricia-Tree.

[12] *State Tree Pruning.* https://blog.ethereum.org/2015/06/26/state-tree-pruning/.

[13] Shuang-Quan L I, Chen H Y, Sun Y X. *Synchronized Mechanism for Multithread in Java*[J]. Modern Computer, 2003.

[14] Kurant M, Markopoulou A, Thiran P. *On the bias of BFS (Breadth First Search)*[C]// Teletraffic Congress. 2010:1-8.