Fully Decentralized Application Model by Peer to Peer Smart Contract of Blockchain

Hong Su [1], Bing Guo [2], Junyu Lu [2], and Xinhua Suo [2]

[1]Sichuan University
[2]Affiliation not available

October 30, 2023

Abstract

Currently, the P2P method is that its software runs on a P2P hardware environment. However, the software is not a P2P one. It is difficult to match the scenarios in which the users' requirements change often. Meanwhile, if the software is out of service, all participants are affected. Thus, in this paper, we propose the fully decentralized application model, in which a P2P software runs in a P2P hardware environment. It is based on the blockchain, as a blockchain provides a secured P2P hardware environment. We focus on its P2P software (the P2P smart contract). A P2P smart contract is formed by smart contracts from its participants instead of a third party. It allows each participant to specify its requirement in a turning-complete way and the failure of one smart contract does not affect other smart contracts. We first describe the requirement of the P2P smart contract and the dependence among them. Then, we propose different ways to pair associated smart contracts. At last, we verify the proposed P2P smart contract model, and it shows more flexibility and robustness than the centralized software method.

Fully Decentralized Application Model by Peer to Peer Smart Contract of Blockchain

Hong Su, Bing Guo, *Fellow, IEEE,* Junyu Lu, and Xinhua Suo

***Abstract*—Currently, the P2P method is that its software runs on a P2P hardware environment. However, the software is not a P2P one. It is difficult to match the scenarios in which the users' requirements change often. Meanwhile, if the software is out of service, all participants are affected. Thus, in this paper, we propose the fully decentralized application model, in which a P2P software runs in a P2P hardware environment. It is based on the blockchain, as a blockchain provides a secured P2P hardware environment. We focus on its P2P software (the P2P smart contract). A P2P smart contract is formed by smart contracts from its participants instead of a third party. It allows each participant to specify its requirement in a turning-complete way and the failure of one smart contract does not affect other smart contracts. We first describe the requirement of the P2P smart contract and the dependence among them. Then, we propose different ways to pair associated smart contracts. At last, we verify the proposed P2P smart contract model, and it shows more flexibility and robustness than the centralized software method.**

***Index Terms*—fully decentralized, peer to peer software, P2P software topic, blockchain application model.**

I. Introduction

FOR an application, its hardware and software may come from a third party or from its peers. It is called the centralized hardware(software) if the hardware (software) is from a third party, or it is called the peer-to-peer (or P2P) one if the software (hardware) is from its peers. Currently, the P2P technology refers to a P2P hardware way[1], which focuses on the distributed hardware, such as the grid computation.

There lacks the P2P software. For the centralized software, participants send parameters to invoke its interface. However, it is difficult to express a complex logic with parameters. In the P2P way, an application is composed of software "segments" from its participants, and there is no centralized software. The P2P software reduces the requirement of a third party to provide software, and it brings more flexibility as users can express their logic in a turning-complete way.

Then, there are three kinds of application models considering whether the software and hardware are P2P or not: the centralized model, the half decentralized model, and the fully decentralized model. Figure 1 shows those models. A centralized model is that both software and hardware are in a centralized way.

The half decentralized model is that either its software or its hardware is in the P2P way. It has two subtypes, type 1 (the software is in the P2P way) and type 2 (the hardware is in the P2P way). Type 2 is the currently used way, which can accumulate the computing resources to finish a relatively big task. The blockchain is in such a way; while different from other P2P methods: a blockchain provides a secure and trusted environment, and even it adopts the consensus algorithm to resolve the conflicts. In this paper, we adopt the blockchain as the P2P hardware environment of the fully decentralized model. We focus on the blockchain software (the smart contract). However, the smart contract is still a centralized method instead of a P2P one, referring to Figure 2. The reason is that the smart contract comes from a third party instead of from its participants. We call this kind of smart contract the centralized smart contract.

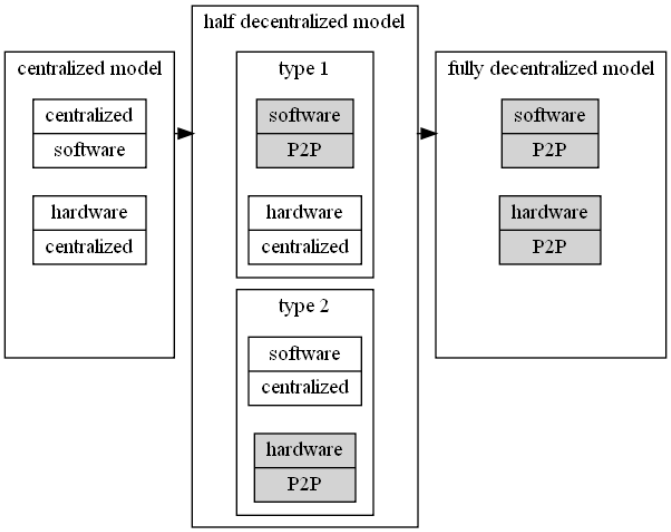

Fig. 1. Different application patterns.

There are some disadvantages of a centralized smart contract. (1) A centralized smart contract is a common contract among different participants. Participants have to use the "centralized" smart contract to interact with others. However, users often have their special requirements, which lead to different contracts for different users. The centralized way is difficult to match this requirement. (2) A centralized software only allows users to provide parameters, which is not turning-complete. If users are allowed to provide its logic, it is more flexible to express their requirements. (3) It is not conventional for the cross-chain interoperation as a centralized smart contract is different to be deployed on different blockchains.

To overcome the disadvantages of centralized smart con-

H. Su, B. Guo, J. Lu, and X. Suo are with the Department of Computer Science, Sichuan University, Chengdu, Sichuan, 610041.

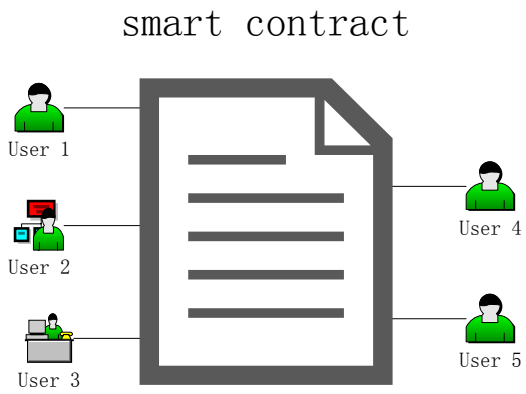

Fig. 2. Different participants cooperates through a centralized smart contract.

tracts, the P2P smart contract is introduced, which allows participants to have specific logic to customize the activities. Figure 3 shows an example of a P2P software. The advantages of P2P smart contracts are as follows.

(1) It promotes the contract among participants, as it allows each one to have its terms in the contract. In the centralized way, the cooperation logic is set by a party, and other participants have to obey it. Take a selling contract for example, which should stand for the willingness of both the seller and the customer. In the centralized way, the contract is provided by one side, which may have unfavorable terms for another side. However, in the P2P way, each participant can express its logic to form a final contract.

(2) It is more flexible for the personalized service. The service for each participant can be different and specific by the P2P software model. Suppose there is a company in which different employees have different payment methods. Their payment is automatically done when the conditions match. Then it needs the P2P smart contract, which provides the personalized payment service for different employees.

(3) It has more robustness. If a P2P smart contract fails, smart contracts from other participants still work. It is like the way that a big smart contract is divided into small parts. Then when one part of the smart contract is out of service, other parts can still work. Comparatively, if a big software is out of service, all its functions are not available.

When both the software and the hardware are in P2P methods, it is the fully decentralized model.

In this paper, we propose the fully decentralized application pattern and focus on the P2P software (the P2P smart contract). Contributions of this paper are as follows.

(1) The fully decentralized application pattern is proposed. The P2P smart contract method allows participants to specify their own logic by different smart contracts. It provides more flexibility and stronger robustness. The implementation is based on the blockchain, which helps to provide a secured P2P hardware environment. Together with the P2P software method, it forms a fully decentralized application pattern.

(2) We proposed the method to use dependence among P2P smart contracts to analyze their relationship. The relationship is how one P2P smart contract matches the requirement of another one.

(3) We proposed three ways to find cooperative smart contracts. The first one is to find cooperative smart contracts from all unmatched smart contracts. The second one is to group smart contracts first, and then it tries to find the paired one in a group. This way reduces the number of smart contracts to compare. The third one is to pair a smart contract by a specific sender or an address.

The rest of this paper is organized as follows. Section II describes the peer to peer (P2P) smart contract model. Section III describes the verification results. Section IV describes previous works, and section V concludes this paper.

II. Peer to Peer Smart Contract Model

In this section, we introduce the peer to peer smart contract that allows each participant to express its logics. A participant first launches a proposal, and then interested participants send their codes to join.

This section includes (1) the introduction to the smart contract, (2) the relationship among P2P smart contracts, and (3) how to choose cooperative smart contracts to form a related application (also called topics)

A. Peer to Peer Smart Contract

A peer-to-peer smart contract is a smart contract whose logic is not fixed by a centralized smart contract but composed of smart contracts from each participant. It is also abbreviated as the P2P smart contract. The currently used smart contract, which is based on a centralized smart contract, is called the centralized smart contract. Note a blockchain application may compose of several smart contracts, while they are from a single agency instead of from its participants, and we also call this kind of smart contract the centralized smart contract.

Figure 4 shows an interaction process by P2P smart contracts. In the beginning, a participant sends a smart contract to express its requirement. This smart contract is called the proposal smart contract. If other participants decide to join, they send their own smart contracts, which are called the participant smart contracts. A typical P2P smart contract interaction process can be divided into three steps.

(1) The proposal stage. A user sends out a proposal smart contract for new cooperation.

(2) The interaction stage. Other participants see the proposal and decide to join or not. If someone wants to join, it sends a participant smart contract. The smart contract contains the status that matches the requirement of the proposal smart contract. At the same time, this smart contract can also contain its requirement for other participants' smart contracts.

(3) The topic formation stage. When all related smart contracts match, cooperation is formed. We say those smart contracts form a topic, which is used to divide different cooperative groups of P2P smart contracts. Topics is discussed further in section II-B.

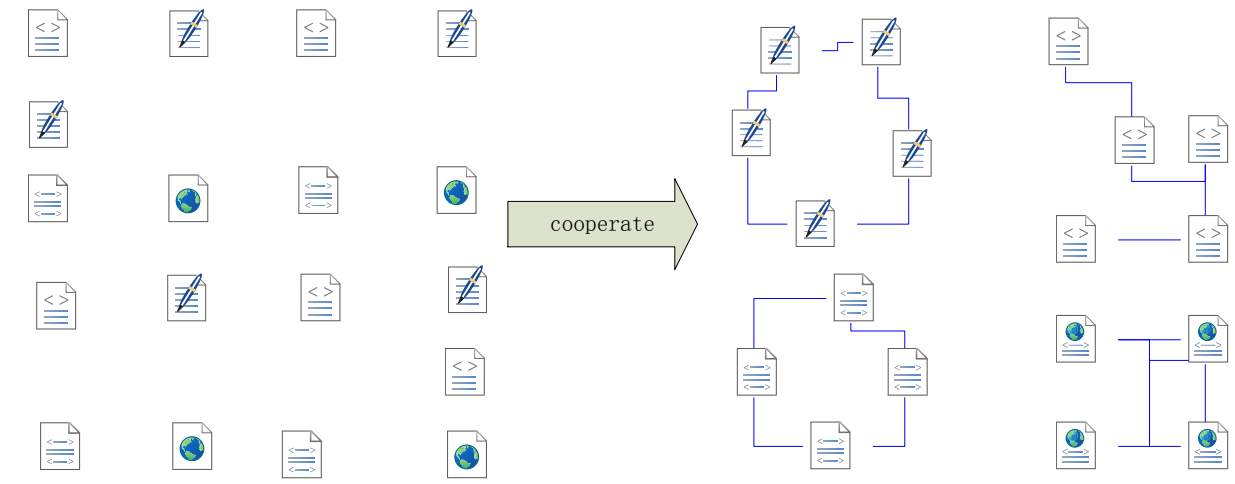

Fig. 3. P2P smart contracts. Different smart contracts can cooperate to complete different tasks. Each participant sends its smart contract to join a task.

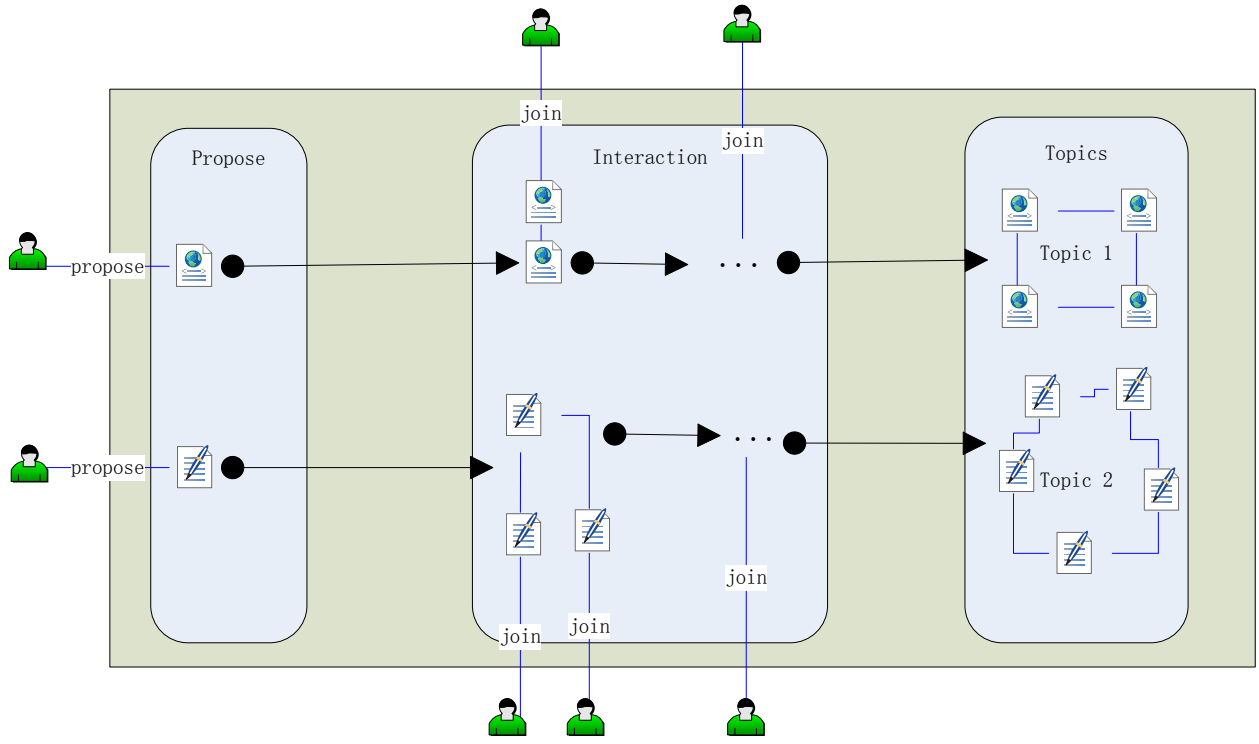

Fig. 4. The interaction process of P2P smart contract. There are three steps in this process. In the first step, a participant proposes cooperation by a smart contract. In the second step, other participants join this cooperation by its smart contract. In the third step, when related smart contracts match, a cooperation (or a topic) forms. There are two topics in the figure, topic 1 and topic 2.

1) Cooperation Requirement: P2P smart contracts cooperate to finish different tasks. However, a task does not allow any smart contract to join. A smart contract should match the condition of the proposed smart contract. Then the proposed smart contract first gives its requirement to select candidate smart contracts for cooperation. A participant P2P smart contract gives its data (or status) to show that it wants to cooperate with other smart contracts.

The candidate smart contract should match the requirement of the proposal smart contract. Now, we formally describe the relationship among P2P smart contracts.

Definition 1: A P2P smart contract (sc) requires status of other P2P smart contracts. Those required status is the *requirement set* of smart contract sc, as in (1).

$$Rsc = \{r_1, r_2, \ldots, r_n\}, r_i \in Rsc \tag{1}$$

, where r_i is a required status for other smart contract by smart contract sc.

Definition 2: A P2P smart contract (sc) provides its own status to other P2P smart contracts. The status provided to other smart contracts is the *data set* of smart contract sc, as in (2).

$$Dsc = \{d_1, d_2, \ldots, d_n\}, d_i \in Dsc \tag{2}$$

, where d_i is the data that smart contract sc provides to others.

A smart contract (sc_t) should match the condition of another smart contract sc_p if they form a cooperative group, as in (3).

$$Rsc_t \subset Dsc_p \tag{3}$$

Then a P2P smart contract has two corresponding fields, the requirement field that is used to choose a candidate smart contract (*requirement set*) and the status field that is given to match the requirement of other smart contracts (*data set*), as shown in Figure 5.

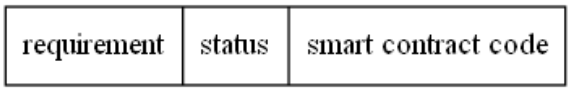

Fig. 5. P2P smart contract data structure.

The requirement field has a challenge. If a smart contract matches the requirement, will it be as expected during the whole topics process? We can see that a smart contract sets its status to match the requirement of others when validation and changes it back when real execution. In this way, other participants are cheated.

Thus, the status of a smart contract should match two requirements to avoid this kind of cheating.

(1) The status field should be verifiable. If it cannot be validated, a participant can easily fake any value in this field.

(2) The status field should ensure the correct behavior at a later time. To analyze the issue, we use the notation of D_s^t to mean the status D provided by smart contract s at time t. The status change from D to D' at a later time $t + \delta$ should be restricted. Both the later state D' and its change condition C should be limited and known by related participants, as in (4).

$$D_s^t \xrightarrow{C} {D'}_s^{t+\delta}, \ |C| \text{ and } |D'| \text{ are limited} \tag{4}$$

We introduce some typical conditions.

(a) Conditions that cannot be changed. The sender of a smart contract is such a condition, which is from a specific account and cannot be changed. Then, one smart contract may require that another smart contract should be from a specific account. Take a company for example. The leave request from an employee should be approved from a specific address (either from the address of HR or its manager).

(b) Conditions that should be changed under specific restriction as in (5).

$$status'.change(status, restrait) \tag{5}$$

The frozen digital asset is such a condition. The sender of a smart contract locks its asset to a smart contract, and the asset is called the frozen asset. The sender cannot transfer this asset back as the owner of the asset has been changed. The transfer of the frozen asset is under clear conditions (restrictions): it is transferred to a peer when the transfer condition matches or the asset is returned to the sender when there are no matching peers within a certain time.

2) Dependence among P2P Smart Contract: A P2P smart contract has three steps during the cooperation with other smart contracts, the step to wait for its requirement (st_w), the step to prepare its state to match the requirement of other smart contracts (st_s), and other execution steps (st_{oe}). We can combine the smart contract name with those components to stand for each component. Such as $f.st_w$ is the waiting step of f.

If one smart contract (SCb) waits the status of another smart contract (SCa), we say SCb depends on SCa, notated as $SCb.st_w.\mathbf{dep}(SCa.st_s)$ or simply $SCb.\mathbf{dep}(SCa)$. Figure 6 shows one example.

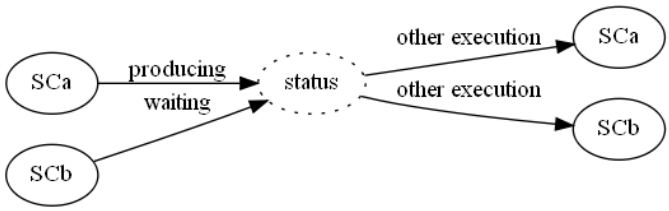

Fig. 6. The relationship among different P2P smart contract.

There are two types of smart contract dependence, direct dependence and indirect dependence considering whether there are intermediate smart contracts or not. Direct dependence means one smart contract waits for the state of another smart contract directly, as shown in the left part of Figure 7. Indirect dependence means one smart contract depends on one or more intermediate smart contracts that further depends on the target smart contract, as in the right part of Figure 7.

We use the notation of $DEP(f)$ to mean the set of all direct dependent smart contracts of smart contract f, and the notation of $DEP_{ind}(f)$ to mean the set of all indirect dependent smart contracts of smart contract f. The dependence in Figure 7 is shown in (6) and (7) separately.

$$f_a \in DEP(f_b) \tag{6}$$

$$f_a \in DEP_{ind}(f_b) \tag{7}$$

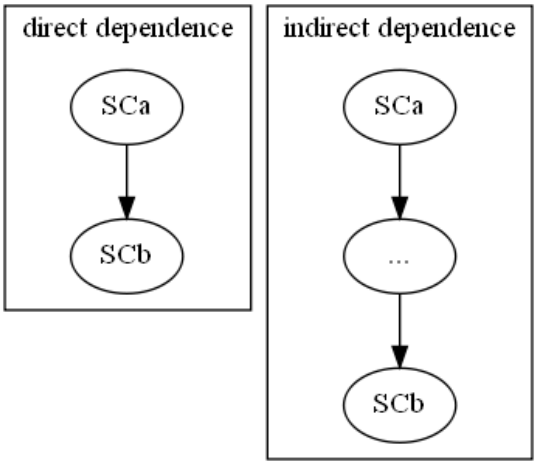

Fig. 7. Different types of dependence, direct dependence and indirect dependence. In the left part, SCb directly depends on SCa; in the right part, SCb depends on SCa indirectly as there is other P2P smart contract between them.

The dependence may be mutual. Mutual dependent smart contracts are smart contracts that depend on each other, $SCb \in DEP(SCa)$ and $SCa \in DEP(SCb)$. For example, in a blockchain exchange among two users UA and UB. UA gives its assets to UB when UB gives its assets to UA and vice versa.

Dependent smart contracts form a graph. The nodes stand for associated smart contracts. The edge is from a smart contract to its dependent smart contract. If we ignore the direction of a direct edge, we get an indirect edge. Corresponding, we use notation of $path$ as the directed path and $upath$ as the indirect path, as in (8) and (9).

$$path(SCa, SCb) \tag{8}$$

$$upath(SCa, SCb) \tag{9}$$

Examples are shown in Figure 8, in which there are different kinds of paths. For example, in part 'a', there is a direct path from SCa to SCb, $path(SCa, SCb)$ and an indirect path $upath(SCb, SCa)$. In part 'd', SCa depends on SCb and SCb depends on SCa, those dependencies form a circle. In part 'e', the dependence forms a bigger dependence circle.

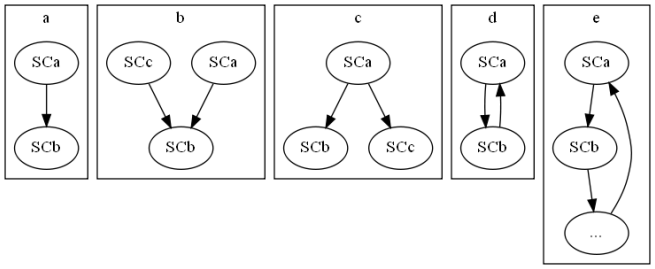

Fig. 8. Different dependence paths.

B. Topics

Different P2P smart contracts may be related or not, which depends on whether they match the requirement of other smart contracts or not. We introduce a new concept, topic, to describe the group of cooperative P2P smart contracts. Suppose there are a set of smart contracts SC as in the following.

$$SC = \{sc_1, sc_2, ..., sc_n\}, sc_i \in SC \tag{10}$$

If smart contract set SC matches the following conditions, we call it a topic.

$$\begin{aligned} &\forall sc_i \in SC, \\ &DEP(sc_i) \neq \emptyset, DEP(sc_i) \subseteq SC \end{aligned} \tag{11}$$

$$\forall sc_i, sc_j \in SC, \exists upath(sc_i, sc_j) \tag{12}$$

(1) Condition (11) shows that each smart contract has its own dependent smart contracts and all its dependent smart contracts are within the smart contract set SC. This indicates that smart contracts in a topic form a completely dependent relationship, and they do not depend on the external smart contracts.

(2) Condition (12) shows that any two smart contracts has dependence. Otherwise, there may be two or more unrelated smart contracts in a topic.

1) Topic Identification: The topic identification is the procedure that miners judge which topic a smart contract belongs to. There are different ways to do this. One simple method is to compare the target smart contract with all unmatched smart contracts. And this kind of way is called the all matching method.

The complexity of this method is n (the number of unmatched smart contracts), as in (13). The reason is that it needs to do the comparison within all smart contracts until the matched one is found. If there is a huge amount of smart contracts, it is a big work to judge whether a smart contract matches the condition of another smart contract or not.

$$\Theta = n \tag{13}$$

Then we introduce other two methods to reduce the work of topic identification, the group matching method, and the address matching method.

The group matching method. Smart contracts are first divided into different groups, and then the target smart contract is compared within its group. The comparison is much smaller

than that of the all matching method. To identify different groups, each smart contract has a group identifier flag. The group identifier is set in the proposal smart contract. Other interested smart contracts specify the same identifier to join.

The complexity of this algorithm is a constant number k, as in (14).

$$\Theta = k \tag{14}$$

We give a brief proof for the complexity as follows.

Assumptions.

(1) We suppose that users cooperate with their specific aims instead of the number of unmatched smart contracts. Then the group size is not related to the number of unmatched smart contracts, n. The group size is decided by the user aim, and it is usually not too much, we assume there exists an average cooperative number k.

(2) The position distribution (p), of which a smart contract and its paired smart contract appear in a group, is linear. Then its value in a group with size k is shown in (15).

$$p = 1/k \tag{15}$$

(3) Suppose the target smart contract is SCt, which is at the position of o_t. Suppose the paired smart contract is SCp, which is at the position of o_p.

Proof.

The matching algorithm tries to find SCp by comparison with all existing smart contracts in sequence (i.e. from 1, 2, 3, ... to n).

Case 1: $o_t >= o_p$, it means that SCt occurs earlier than SCp. As SCp does not appear, SCt compare with all the o_t smart contracts. The number of comparison is o_t.

Case 2: $o_t < o_p$, it means that SCp occurs earlier than SCt. In this case SCp is found, and the number of comparison is o_p.

Case 1 and case 2 occur with the same probability, and then the comparison count, n_{comp}, is as in (16).

$$n_{comp} = 1/2o_t + 1/2o_p \tag{16}$$

In the assumption, o_t and o_p are linearly distributed in any position of the group (with the probability, p). The average number of comparison, n_{comp}, is (17).

$$\begin{aligned} n_{comp} &= 1 * p + 2 * p + ... + k * p \\ &= (k+1) * k * (2p) \\ &= 2k + 2 \end{aligned} \tag{17}$$

Then the complexity is a constant k.

The address matching method is that a smart contract specifies the address of its paired smart contract. As the target smart contract is directly pointed out, it can be found by the address. The complexity of this algorithm is 1, as in (18).

$$\Theta = 1 \tag{18}$$

C. Topics Choice - Transaction Order Based Selection

There may be several candidates to compete for a topic. For example, a user (user UA) wants to exchange its asset, and it specifies the requirement to exchange. Two users (user UB and UC) want to exchange with UA, and each sends a P2P smart contract. Suppose both UB and UC match the condition of UA. Then there is an issue with how to select one.

In this paper, we adopt the method that regards the time at which smart contracts appear, the earlier the best. Other methods include the reward method which considers the reward given to the miner, as it reflects the willingness to join a topic.

As the sending time of a smart contract is difficult to get in some blockchains, the appearance order of a smart contract is more convenient, and then we use the smart contract order to measure the time. We begin with related definitions.

Definition 3: Smart Contract Order. All related smart contracts are sorted by the order that they are put to the blockchain. The order is first decided by the block index; if two smart contracts are in the same block, it is decided by the order in that block. In a topic, we number each smart contract from 1, which stands for the order of a smart contract. For different topics, we directly compare by this rule.

We use $order(s,t)$ to mean the order of smart contract s in topic t.

Definition 4: Different Smart Contract among Topics (DSC). If a smart contract sc belongs to one candidate topic tp_a and does not belong to another candidate topic tp_b, we call sc is a different smart contract among topics tp_a and tp_b. It helps to describe the difference among topics.

We use $d(tp_1, tp_2, i)$ to mean the i_{th} different smart contract in topic tp_1 between topic tp_1 and tp_2, and $d(tp_2, tp_1, i)$ to stand for that in topic tp_2. Some examples are as follows.

$$\begin{gathered} tp_1 = sc_1, sc_2, sc_3, tp_2 = sc_1, sc_4, sc_5 \\ d(tp_1, tp_2, 1) = sc_2, d(tp_2, tp_1, 1) = sc_4 \\ d(tp_1, tp_2, 2) = sc_3, d(tp_1, tp_2, 2) = sc_5 \end{gathered}$$

This notation can be simplified. If the comparative candidate topics are clear, we can only mention the first topic. Such as to use $d(tp_1, i)$ for $d(tp_1, tp_2, i)$ when the target topics are tp_1 and tp_2.

With this notation, we now describe different transaction order based tactics.

1) First DSC Tactic: First DSC is the first different smart contract between two candidate topics. For topic tp_1 and tp_2, the first DSC of tp_1 is $d(tp_1, tp_2, 1)$ or $d(tp_1, 1)$.

The first DSC tactic is that if there are several candidate tactics, we choose the topic tp_c whose first DSC has the minimum order. This tactic is as in (19). Algorithm 1 is the pseudocode to choose the first DSC between two topics.

$$tp_c = min(d(ti, 1)) \tag{19}$$

Algorithm 1 can apply to several topics cases. It is an iterative process among a binary tree as those comparisons are in pairs. Such as for four candidate topics tp_1, tp_2, tp_3, tp_4, we can choose the target topic as in algorithm 2 and the comparison tree is in Figure 9.

Algorithm 1 To choose the topic with the first DSC between two topics.

Require: t1 and t2 are two candidate topics
Ensure: the chosen topic between t1 and t2
1: Topic chooseCompletion(Topic t1, Topic t2)
2: **if** order(d(t1,1) <d(t2,i) **then**
3: **return** t1
4: **end if**
5: **if** order(d(t1,1) >d(t2,i) **then**
6: **return** t2
7: **end if**
8: //error case as different smart contracts has different orders
9: **return** ERROR

Algorithm 2 To choose the topic with the first DSC among four topics.

Require: t1, t2, t3, t4 are two candidate topics
Ensure: the chosen topic among them
1: Topic chooseCompletion(Topic t1, Topic t2, Topic t3, Topic t4)
2: t11 = chooseCompletion(t1, t2)
3: t12 = chooseCompletion(t3, t4)
4: **return** chooseCompletion(t11, t12);

2) Average DSC tactic: However, the above tactic only considers the first DSC. In some cases, we should consider the order of all smart contracts. Then we introduce, the average order (ao_{tp_i}) of topic tp_i, as (21).

$$ao_{tp_i} = 1/n \sum_{1}^{n} order_{sc_{ij}} \tag{20}$$

, where n is the number of smart contracts in topic tp_i; sc_{ij} is a smart contract of tp_i.

Then we choose the topic tp_c with the minimum average order value.

$$tp_c = \min(ao_{tp_i}) \tag{21}$$

3) Last Smart Contract Tactic: If we want to choose a topic that complete earliest, we choose the one that complete earliest. It is also equal to that its last smart contract complete earliest. We use sc_{last} to stands for the last smart contract. As sc_{last} is the last smart contract, it has the maximum order in a topic tp, as in (22).

$$sc_{last} = \max(order_{sc_i}), sc_i \in tp \tag{22}$$

Similarly, we choose the topic tp_c with the minimum order of sc_{last}, as in (23).

$$tp_c = min(order_{sc_{last}}) \tag{23}$$

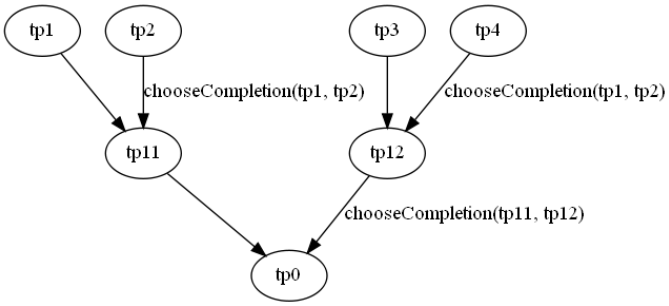

Fig. 9. Iteration tree to choose a topic among four topics, tp_1, tp_2, tp_3, and tp_4. $tp_1 1$ is the chosen topic between tp_1 and tp_2; and $tp_1 2$ is that between tp_3 and tp_4. tp_0 is the finally chosen topic.

III. VERIFICATION

A. Test environment

The test environment includes three parts, the blockchain choice, the deployment of the blockchain, and the P2P smart contract procedure.

Blockchain choice. A self-developed blockchain is used for the simulation, as the publicly available blockchains do not support the P2P smart contract. The consensus algorithm of this blockchain is PoW (Proof of Work) and the difficulty is set to the degree that the hash of a block starts with at least 6 zeros, which results in an average mining period of tens of seconds in our verification environment.

Blockchain deployment. The blockchain is deployed in 12 virtual machines of an HP workstation (i9-9900/64GB NECC/256GSSD+2TB SATA). The operating systems are Ubuntu-16.04.3. The CPUs are 1, 2, or 4 virtual CPUs, and memories are 2G or 4G. Those nodes form a P2P network that is based on socket communication. For simplification, the peer address is configured in its configuration file instead of getting peers' addresses dynamically.

P2P smart contract procedure. It contains several steps. (1) A participant sends its P2P smart contract in a transaction; the code of the smart contract is based64 encoded and is put in the data field of the carrier transaction. (2) Miners first try to validate the status part. If failed, it is not processed further. To facilitate a user to lock its asset in the status part, the blockchain provides the interface to lock an asset. With this interface, the asset transfers to the receiver if its requirement matches; otherwise, the asset is restored. (3) Miners try to find paired P2P smart contracts according to the proposed model. (4) If all paired smart contracts are found, st_{oe} parts of those smart contracts are executed.

B. Comparison between Peer to Peer Smart Contract Method with Centralized Method

In this section, we compare the p2p smart contract method (the p2p method) with the centralized smart contract method (the centralized method). The aim is to verify that P2P smart contracts express logic more feasibly and have higher error tolerance than the centralized way.

The verification scenario is asset exchanges among different participants, in which participants use their assets to exchange

TABLE I
PEER 2 PEER EXCHANGE

user	requirement	data	exchange identifier	st_{oe} code
1	5 assets	1 asset	101	log
2	6 assets	2 asset	102	log
3	7 assets	3 asset	103	log
4	8 assets	4 asset	104	log
5	1 asset	5 asset	101	log
6	2 assets	6 asset	102	log
7	3 assets	7 asset	103	log
8	4 assets	8 asset	104	log

TABLE II
TEST CASES

name	out of service(P2P)	out of service(centralized)
e1	smart contract of u1	when u1 interact
e2	smart contract of u2	when u2 interact
e3	smart contract of u3	when u3 interact
e4	smart contract of u4	when u4 interact
e5	smart contract of u5	when u5 interact
e6	smart contract of u6	when u6 interact
e7	smart contract of u7	when u7 interact
e8	smart contract of u8	when u8 interact

other's assets. The scenario can be done in either the centralized way or the P2P way. The exchange has two parts; part1, how many assets it wants to give, and, part2, how many assets it wants to get from its paired exchanger. In the centralized way, a smart contract is used to manage the exchange process. Participants send their parameters to express part1 and part2. In the P2P way, participants specify the exchange by P2P smart contracts, in which the requirement of the smart contract is part2 and the status part is part1.

There are four paired exchanges, as in table I. The requirement part is what the user wants from its peer and the data part is what the user promises to give to its peer. In the P2P way, each exchange is identified by an identifier, the exchange identifier in the table. The st_{oe} code part code is very simple, it only logs successful information when an exchange completes.

The sequence to interact with the blockchain is from user 1 to user 8. Table I shows the sequence, in which the proposal participants (i.e. users from 1 to 4) send their exchange first and then the paired participants (i.e. users from 5 to 8) send their exchange. This sequence helps to see whether exchanges affect others or not.

To make the comparison more clear, we set one of the smart contracts out of service in each test round. In the P2P smart contract way, we choose one of the smart contracts from user 1 to user 8 to be out of service. In the centralized way, the centralized smart contract is out of service when one of those users interacts with it. Then there are 8 test cases (named e1, e2, to e8) for the P2P way and the centralized way. Table II lists those cases.

The comparison is in three aspects, the affected assets, the affected user, and the number of separate processing programs. The affected assets help to understand the difference among compared smart contract methods. The affected user and the number of separate processing programs help to understand the background reason.

1) Affected Assets: The affected assets are assets that have been frozen and cannot be further processed due to the failure of the related smart contract.

Test cases from e1 to e8 have different impacts on the affected assets due to the moment that the smart contract is out of service is different. The result of the affected assets is shown in Figure 10.

From Figure 10, we know that the amount of the affected assets in the P2P way is less than that in the centralized way, with an average of 1.25 assets frozen in the P2P way and an average of 5 assets frozen in the centralized way. The reason is that an out-of-service centralized smart contract affects other uncompleted exchanges and then frozen assets cannot be processed further; while an out-of-service P2P smart contract only affects the exchange of its owner. This can be further understood from two different phases of the process.

The first phase is that the failure happens before any of the second participants of an exchange sends its P2P smart contract. Test cases e1 to e4 belong to this case. In the P2P way, there is no frozen asset. In the centralized way, 3 test cases (from e2 to e4) have the frozen asset, and the maximum frozen asset is 6 assets in e4.

The second phase is that the failure happens when one of the second participants of an exchange sends its smart contract. This phase includes test cases from e5 to e8. In the P2P way, the assets have been affected, 1 frozen asset in e5 when the smart contract from user 5 is out of service, and 2 frozen assets in e6. The centralized way has more affected assets because unrelated participants are affected. We see that 10 assets are frozen in e5, and 9 assets are frozen in e6. More details referring to table III.

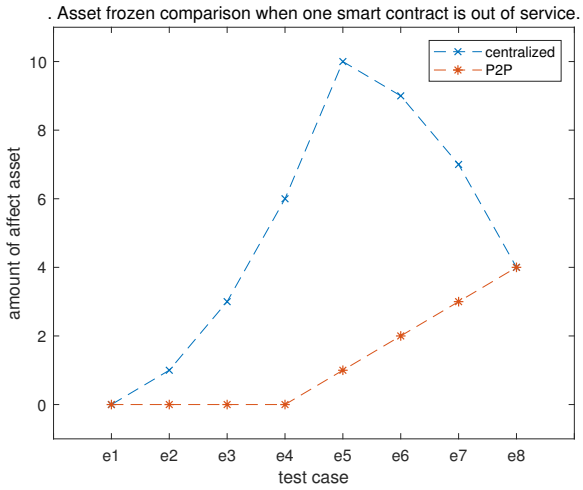

Fig. 10. Asset frozen comparison when one smart contract is out of service.

2) Affected User: The P2P method isolates the actions of different participants. The failure of one participant does not affect other participants' actions. In this section, we focus on the affected users. The affected users are the users that have interacted with the blockchain; while those users cannot process further due to that its related smart contract is out of service. The results of the number of affected user (in test cases from e1 to e 8) are shown in Figure 11 and table IV.

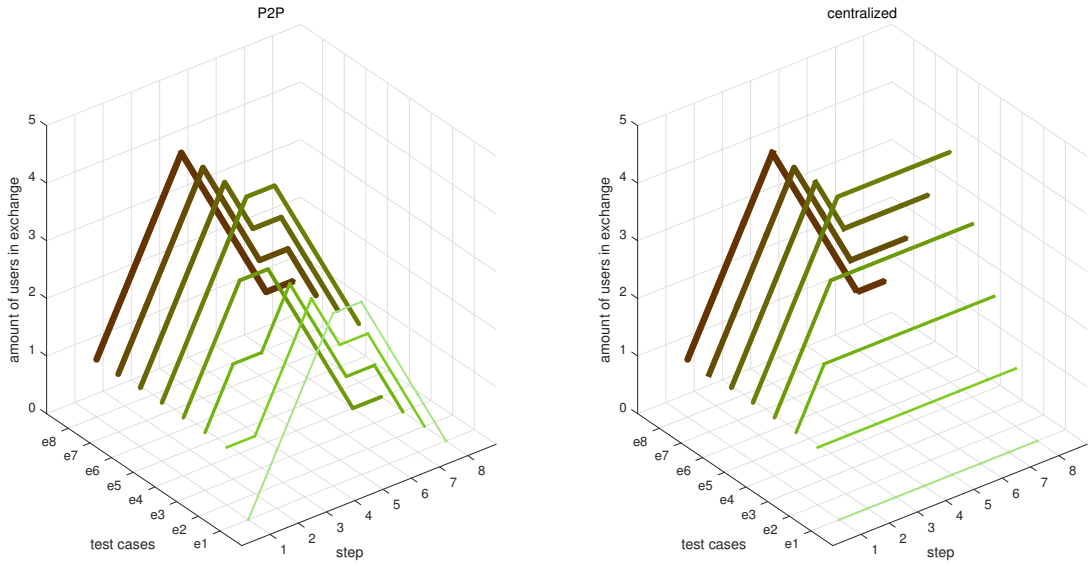

Fig. 11. The number of affected user during the process. In P2P method, failure of a participant only affect its paired exchanger. In the centralized method, all successive participants are affected when the failure of the centralized smart contract.

TABLE III
AMOUNT OF AFFECTED ASSETS

	e1	e2	e3	e4	e5	e6	e7	e8
centralized	0	1	3	6	10	9	7	4
P2P	0	0	0	0	1	2	3	4

TABLE IV
AMOUNT OF AFFECTED USERS

	e1	e2	e3	e4	e5	e6	e7	e8
centralized	0	1	2	3	4	3	2	1
P2P	0	0	0	0	1	1	1	1

From Figure 11, we see that there are also two obvious stages of the affected user. The first stage is the period when the fault occurs from user 1 to user 5 (test cases from e1 to e5). In the centralized method, the number of impacted users accumulated from 0 to 4. Comparatively, there is no user affected in the P2P method. Even the sender who sends the fault smart contract is not affected, as its smart contract is out of service.

The second phase is from the moment when the faults happen when user 6 joins (test cases from e6 to e8). The amount of users decreases as some exchanges complete. However, for the centralized way, when the fault smart contract happens, other accumulated users cannot exchange successfully. Then the number of affected users is 3, 2, 1 for e6, e7, and e8 separately. In the P2P method, there is 1 affected user, which is the paired exchanger of the fault smart contract. Then we get a conclusion that there are fewer affected users in the P2P method than that in the centralized method.

3) Separate Processing Programs: Separate processing programs are smart contracts that process user requests independently. The fault of one program does not affect the processing of another. The difference of the affected assets and affected users can be further understood from the difference of separate processing programs.

There are 4 exchanges in this verification. In the centralized ways, one smart contract process all exchange, which can be seen that there is only one processing program. If this program is blocked, no exchanges can continue. Comparatively, the P2P way has 8 processing programs. And paired processing smart contracts form an exchange.

In this verification, each smart contract logs to a file when starting, aiming to facilitate the count of separate processing smart contracts. The log file is cleaned at the beginning of a test case, and we count the number after each user sends its request either by a P2P smart contract or its parameters. The results are shown in Figure 12. There are more separate processing programs in the P2P method (maximum 8 programs) than that in the centralized method (1 program). Then when one separate processing program is out of service, there are other separate processing programs to handle service in the P2P way.

C. Comparison Efficiency

In this section, we try to compare the efficiency of different matching methods. The number of candidate smart contracts is different, which results in different comparison counts to find a paired smart contract, i.e. different comparison efficiency.

In the verification, there are 512 smart contracts. The all matching method and the group matching method are compared. The all matching method can be seen as only 1 group to process 512 smart contracts. In the group matching

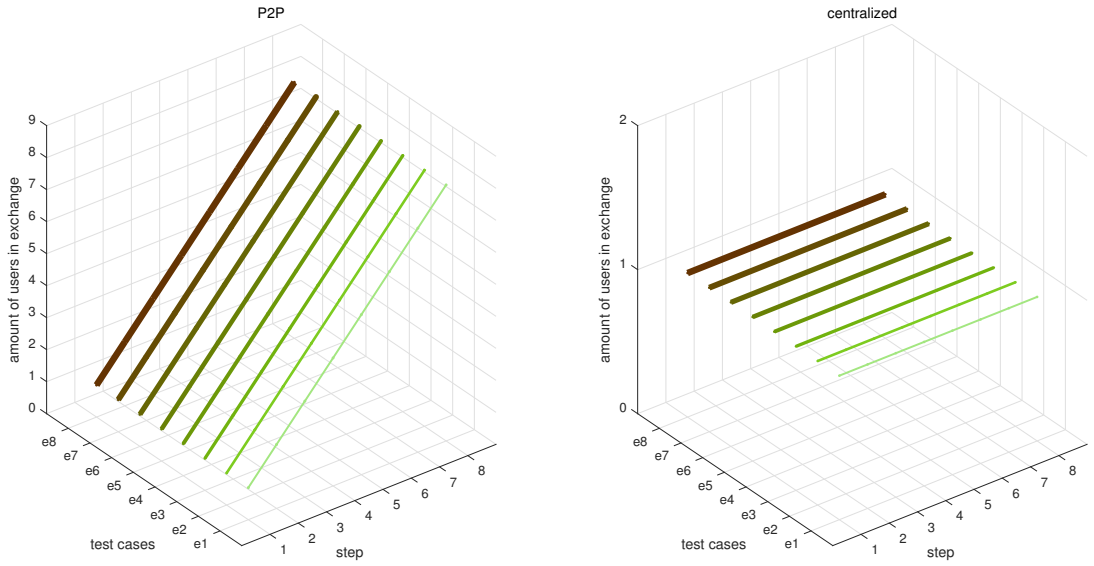

Fig. 12. Number of separate processing programs.

TABLE V
DIFFERENT MATCHING METHODS

name	number of groups
256_group	256
128_group	128
64_group	64
32_group	32
16_group	16
8_group	8
4_group	4
2_group	2
All match	1

method, we choose 8 types of groups, which results in different numbers of smart contracts to compare. The test details are in table V.

The order and group of smart contracts are random. (1) The order of the target smart contract and its paired smart contract is generated randomly. Then the target smart contracts have the random probability to be before and after the paired smart contract, which helps to cover both the scenario in which the paired smart contract can be found and the scenario in which the paired smart contract can not be found. (2) Smart contracts are randomly assigned to one of the groups. For example, in the 8-group method, each smart contract is generated and assigned with one identifier of the 8 groups. It makes the size of the group random; while some groups may be empty.

We compare candidate smart contracts one by one in its group to check whether it is paired or not. If the paired smart contract is in the group currently, the comparison count is equal to the order of the paired smart contract in this group. Otherwise, all candidate smart contracts will be compared, and the comparison count is the number of all smart contracts. We use a map to store the address of each group. Then when a smart contract specifies the group number, it is a constant time to find the corresponding group, and then we count it as 1 comparison.

We have carried out 32 test rounds for each type. Figure 13 shows the results of different matching methods (all matching method is named 1_group).

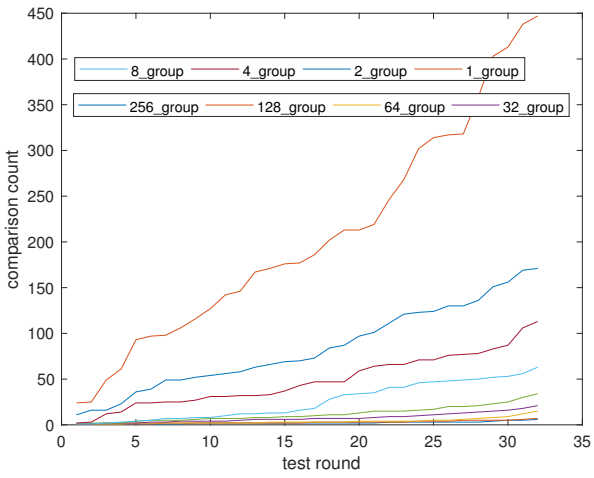

Fig. 13. The comparison counts for different group size.

From Figure 13, we see that the centralized way (1_group) has the maximum number of comparisons. The reason is that the comparison is among all unpaired smart contracts. Then the comparison count is relatively big, with an average number of comparisons of 207.2. Method 256_group has the minimum number of comparisons, with an average of 2.3 comparisons. To give a more clear tendency of the comparison count, we show the average number of comparison count in Figure 14.

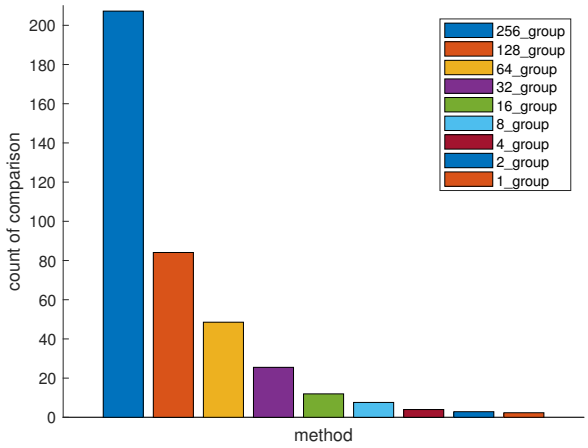

Fig. 14. The average comparison counts to find a paired smart contract.

From Figure 14, we see that the comparison counts decrease when the number of groups increases. The average counts of comparison are 2.3, 2.8, 3.9, 7.5, 11.9, 25.5, 48.5, 84.0, 207.2 for 1, 2, 4, 8, 16, 32, 64, 256 groups separately. When there are more groups, each group has fewer P2P smart contracts to compare averagely. Then the average number of comparisons is less.

From the above, we see that P2P smart contracts have the ability to form different exchange group feasibly. And it also helps to shorten the completion time of an exchange.

1) Shortening Completion Time: Group also helps to shorten the average completion time of an exchange. For example, some participants may delay the interaction time due to various reasons, and other participants have to wait for it. The more participants are in the same group, the more participants have to wait.

To measure the completion time of one group, we use the average completion time, $CTime$. Suppose there are n participants, $CTime$ is defined as (24).

$$CTime = \sum_{i=1}^{n} etime_i/n \tag{24}$$

, where $etime_i$ is the time that participant i takes in the exchange, which counts from the moment when the smart contract of a participant is sealed into the blockchain to the moment the exchange has completed.

Thus, $etime_i$ includes the time that its own takes and the time to waits for other participants (for the exchange to complete), as in (25).

$$etime_i = time_{p2p} + time_{waiting} \tag{25}$$

, where $time_{p2p}$ is the time that the P2P smart contract takes, and $time_{waiting}$ is the time to wait for other participants.

The verification for the impact of completion time is among 16 participants. There are three kinds of exchange groups, 2u, 4u, and 8u groups. 2u means that 2 users form an exchange. 4u and 8u mean that 4 and 8 users form an exchange separately. We add a sleep function in the P2P smart contract to simulate a random $time_{p2p}$ time (sleep linearly from 1 second to 10 seconds). And in each test round, a participant is chosen to have a long sleep time (100 seconds), with the aim to make other participants waiting for a long time (a long $time_{waiting}$).

The results are shown in Figure 15. From Figure 15, we see that the exchange time increases when group size increases. The average exchange time ($CTime$) is 16.0, 30.5, 54.1 seconds for groups with 2, 4, and 8 members separately. When the exchange is only between two participants, the delayed smart contract only affects another participant, then the average exchange time is the least, 16.0 seconds. However, when the group size increases, more participants in the same topic may be influenced. Thus, the average exchange time of 8u is the most (54.1 seconds). Especially, in the 8 participants' exchange, if the delayed participant appears in the latest, all other 7 participants have to wait a long time.

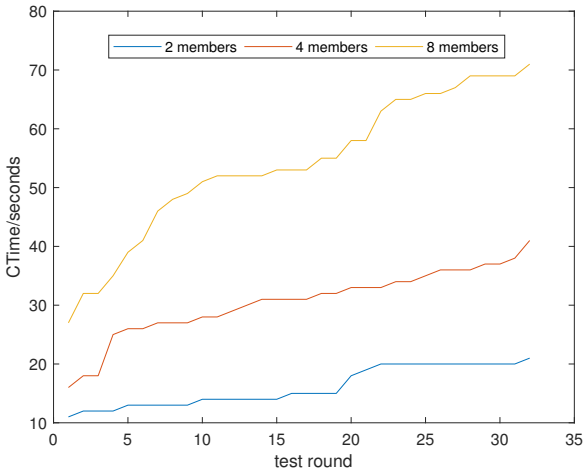

Fig. 15. The completion time of different group size.

The completion time indicates that participants should join a smaller size topic. With the P2P methods, participants can freely choose to join the topic with the desired size. Comparatively, in the centered smart contract, the logic is fixed, and it is difficult to allow participants to customize their topic size.

IV. Related Work

In the early stage of P2P technology, the software runs on the hardware of its peers, aiming to eliminate the dependency on a centered third-party. This fascinates participants to exchange data resources (such as MP3) among them. There are three major aspects of related research. One is to set up a cooperation network. Work [5] focuses on the unstructured P2P network for file sharing. Work [4] propose a structured index system that provides equality and range queries. The second aspect focuses on how to improve efficiency or scalability. Work [2] proposes a scalable data access structure. Work [3] proposes to improve efficiency and fairness by the method

of effort-based incentives. The third aspect provides common frameworks to support the development of P2P software. Work [6] summarizes the necessary to program the grids. Work [7] provides a P2P-based middleware to implement the transfer and aggravation of computing resources.

Blockchain provides a secured P2P model and a consensus algorithm on the P2P environment to eliminate possible conflicts among peers. Actions on a blockchain cannot be denied and the blockchain helps to keep the data immutability [8]. This can be seen as the second phase of the P2P cooperation. It depends on the encrypt methods and the consensus algorithm to enable cooperation among different participants. The consensus algorithm ensures that the results are deterministic [9] [10]. There is no third-party required to ensure the consensus. Blockchain provides an environment for fully decentralized applications.

The smart contract is a program on the blockchain. It allows the users to define more complex cooperation among different participants. Some research focus on how to improve the gas fee in the smart contract [11]. Some research focuses on the application of smart contracts, such as to promote resource share [12]. Even the smart contract is applied to interact with non-blockchain system [13].

However, the smart contract still needs a third-party software (a smart contract) that provides a common interaction framework among participants. This centralized smart contract runs for all participants and there will be a big loss if the centralized smart contract is broken. An example is the attack for the DAO project [14]. To enhance a smart contract, there are some ways to formally validate a smart contract before it deployed into the blockchain [15]. However, the single fault of a smart contract still blocks the whole application. At the same time, all participants provide corresponding parameters to invoke the interface of a smart contract, which is not a turning-complete method for the participants.

V. Conclusion

In this paper, we address the issue of how to achieve a fully decentralized application model, which requires both hardware and software to run in a P2P way. The P2P application model allows to customize different requirements of users and has more robustness. Our solution is based on the blockchain, as a blockchain has provided the P2P environment with the consensus algorithm. Then we focus on the P2P software (smart contract) in this paper. We first proposed the P2P smart contract that has a requirement for others and the status which provides to match other's requirement. Then we describe how to pair associated smart contracts to form a topic. We propose three matching methods considering the comparison scope. At last, we verify the P2P smart contract and compare it with the centralized methods. Overall, we provide a fully decentralized way to implement applications that provides more flexibility and robustness.

Acknowledgment

The authors thank the anonymous reviewers for their constructive comments, which help us to improve the quality of this paper. This work was supported in part by the National Natural Science Foundation of China under Grant No. 61772352; the Science and Technology Planning Project of Sichuan Province under Grant No. 2019YFG0400, 2018GZDZX0031, 2018GZDZX0004, 2017GZDZX0003, 2018JY0182, 19ZDYF1286.

References

[1] Rufino J, Alam M, Almeida J, et al. Software defined P2P architecture for reliable vehicular communications[J]. Pervasive and Mobile Computing, 2017, 42: 411-425.
[2] Aberer K, Punceva M, Hauswirth M, et al. Improving data access in p2p systems[J]. IEEE Internet Computing, 2002, 6(1): 58-67.
[3] Rahman R, Meulpolder M, Hales D, et al. Improving efficiency and fairness in p2p systems with effortbased incentives[C]//2010 IEEE International Conference on Communications. IEEE, 2010: 1-5.
[4] Crainiceanu A, Linga P, Machanavajjhala A, et al. Pring: an efficient and robust p2p range index structure[C]//Proceedings of the 2007 ACM SIGMOD international conference on Management of data. 2007: 223-234.
[5] Stutzbach D, Rejaie R, Sen S. Characterizing unstructured overlay topologies in modern P2P file-sharing systems[J]. IEEE/ACM Transactions on Networking, 2008, 16(2): 267-280.
[6] Gannon D, Bramley R, Fox G, et al. Programming the grid: Distributed software components, p2p and grid web services for scientific applications[J]. Cluster Computing, 2002, 5(3): 325-336.
[7] Shudo K, Tanaka Y, Sekiguchi S. P3: P2P-based middleware enabling transfer and aggregation of computational resources[C]//CCGrid 2005. IEEE International Symposium on Cluster Computing and the Grid, 2005. IEEE, 2005, 1: 259-266.
[8] Hofmann F, Wurster S, Ron E, et al. The immutability concept of blockchains and benefits of early standardization[C]//2017 ITU Kaleidoscope: Challenges for a Data-Driven Society (ITU K). IEEE, 2017: 1-8.
[9] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. bitcoin.org, 2008.
[10] Gramoli V. From blockchain consensus back to Byzantine consensus[J]. Future Generation Computer Systems, 2020, 107: 760-769.
[11] Marchesi L, Marchesi M, Destefanis G, et al. Design patterns for gas optimization in ethereum[C]//2020 IEEE International Workshop on Blockchain Oriented Software Engineering (IWBOSE). IEEE, 2020: 9-15.
[12] Han D, Zhang C, Ping J, et al. Smart contract architecture for decentralized energy trading and management based on blockchains[J]. Energy, 2020, 199: 117417.
[13] Su H, Guo B, Shen Y, et al. A Solution for State Conflicts of Smart Contract in Interaction with Non-blockchain[C]//IEEE INFOCOM 2020-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS). IEEE, 2020: 382-387.
[14] Mehar M I, Shier C L, Giambattista A, et al. Understanding a revolutionary and flawed grand experiment in blockchain: the DAO attack[J]. Journal of Cases on Information Technology (JCIT), 2019, 21(1): 19-32.
[15] Magazzeni D, McBurney P, Nash W. Validation and verification of smart contracts: A research agenda[J]. Computer, 2017, 50(9): 50-57.

Hong Su Hong Su received the BS and MS degrees, in 2003 and 2006, respectively, from Sichuan University, Chengdu, China. He is currently a Ph.D. candidate in the School of Computer Science and software Sichuan University, Chengdu, China. His research interests include blockchain and the Internet of Value.

PLACE PHOTO HERE

Bing Guo Bing Guo is currently a professor of Sichuan University, Ph.D. supervisor. He received the Ph.D. from University of Electronic Science and Technology of China in 2002. His current research interests include green computing, personal big data, and blockchain.

PLACE PHOTO HERE

Junyu Lu Junyu Lu is currently working toward the PhD degree in computer science in Sichuan University, Chengdu, China. He received the B.E. degree in computer science and the M.E. degree in computer technology from Sichuan University. His major research interests include Mobile Cloud, Mobile Distributed Computing and Resource Allocation.

PLACE PHOTO HERE

Xinhua Suo Xinhua Suo is a Ph.D. candidate in Sichuan University. His current research interests mainly include Software architecture, Knowledge Graph (KG), Natural Language Processing (NLP).