Fully Decentralized Model by P2P Smart Contract To Achieve High Availability in IoT

Hong Su*, Bing Guo†, Junyu Lu†, Xinhua Suo† *School of Computer Science, Chengdu University of Information Technology, Chengdu, China †College of Computer Science, Sichuan University, Chengdu, China

***Abstract*—In recent years, the number of IoT devices has increased substantially. One of the main concerns is the interaction with high availability. While peer-to-peer (P2P) technology has been used to achieve high availability by software running in P2P hardware, a single software still acts as the center for the interaction. If the software has bugs or is out of service, all devices are affected. Aiming at this issue, this paper proposes a fully decentralized model as a high-availability solution, in which P2P software runs on a P2P hardware environment. P2P software is a set of independent software from respective participants (IoT devices) that can work cooperatively without centralized software. We use blockchain as a P2P platform because it has P2P nodes and a trustless environment, so the P2P software in this paper is P2P smart contracts. To pair cooperative smart contracts, each P2P smart contract has requirements on other smart contracts. If several smart contracts matches the requirements mutually, they are treated as a work group (a topic), and then they process each other's requests within that group. When a single smart contract fails, the paired P2P software can be re-selected, resulting in high availability. Simulation results show that the proposed model can effectively eliminate the impact of the failure of any single smart contract.**

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

I. Introduction

THE number of IoT devices has increased dramatically over the past few decades. Forbes reports that the number of IoT devices will increase to 500 billion by 2030 [1]. Meanwhile, the types of devices are also increasing to adapt to more and more IoT scenarios. IoT devices need to interact with other devices, which is usually done by intermediary systems. High availability of intermediate systems is one of the major concerns, as its failure may affect a large number of IoT devices.

High availability refers to the degree to which a system can provide service when some hardware or software is out of service. The P2P method is often used to achieve high availability [2] [3]. In this method, the *same copies* of software run on the hardware of each peer, aiming to provide service if the hardware of some peers fails. However, only the hardware is in P2P mode; the software is still a single software that acts as the center for the interaction of participants.

There are some disadvantages of the currently used P2P method. If the software has bugs (such as an unwanted infinite loop or an exit due to a null pointer), the instances on each peer have the same bug because they are created from the same binary code. This makes it difficult to serve IoT devices after the error occurs.

To overcome the above disadvantages, we propose P2P software composed of peer (e.g., an IoT device) softwares. Each P2P software can work independently, and no one acts as the central software; related P2P softwares work cooperatively to provide interactive services between IoT devices. We use blockchain as a P2P platform because it has P2P nodes and a trustless environment [4] [5], which allows us to focus only on P2P software. Then the P2P software in this paper is P2P smart contracts, because the software on the blockchain is smart contracts [6] [7].

P2P smart contracts bring greater robustness [8]. The software of each peer [9] runs independently, and the software failure of some peers does not affect the software of other peers. That is other peers' smart contracts can still work when a P2P smart contract fails. It is like the way a smart contract is divided into small parts. When one part of the smart contract goes out of service, other parts can still work. By contrast, if the whole software goes out of service, all its functions cannot be used.

When both software and hardware are P2P, it is a fully decentralized model. In this paper, we propose the fully decentralized application model and focus on P2P software (P2P smart contracts). The contributions of this paper are as follows.

(1) We propose a fully decentralized application model. The P2P smart contract method allows participants to specify their own logic through different smart contracts. It provides greater robustness and flexibility than centralized software. The implementation of the model is based on blockchain, which provides a secure P2P environment. Together with the P2P software approach, a completely decentralized application model is formed.

(2) We propose P2P software. P2P software is software from all peers, and related P2P software composes application scenarios. We use the dependencies between P2P smart contracts to analyze the relationship between them. This relationship is how one P2P smart contract matches the requirements of another P2P smart contract.

(3) We propose two methods for finding cooperative smart contracts. The first is to find cooperative smart contracts from all unmatched smart contracts. The second is to group smart contracts first, and then try to find the paired one in a group, which reduces the number of smart contracts to compare.

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 and Section IV concludes the paper.

II. Peer to Peer Smart Contract Model

A. Peer to Peer Smart Contract

A peer-to-peer smart contract application consists of smart contracts (peer-to-peer smart contracts) from each participant, where no single smart contract acts as a centralized contract for others. Peer-to-peer smart contracts are also referred to as P2P smart contracts. The currently used smart contracts are called centralized smart contracts.

Figure 1 shows the interaction process of a P2P smart contract application. In the beginning, a device sends a smart contract expressing its request for cooperation. This smart contract is called a proposal smart contract. If other devices decide to join, they send their own smart contracts, called participant smart contracts.

A typical P2P smart contract interaction process can be divided into three steps.

(1) Proposal stage. A device issues a proposed smart contract for a new cooperation.

(2) Interaction stage. Other devices see the proposal and decide whether to join. If a device wants to join, it sends a participant smart contract. The smart contract contains the status that meets the requirements of the proposed smart contract. At the same time, the smart contract can also contain its requirements for other participants' smart contracts.

(3) Topic formation stage. Cooperation is formed when the requirements of all smart contracts match. We say that these smart contracts form a topic, which is used to divide different cooperative groups of P2P smart contracts. Topics are discussed further in the II-B section.

1) Cooperation Requirements: P2P smart contracts cooperate to accomplish different tasks. However, tasks do not unconditionally allow any smart contracts to join. The smart contract should match the conditions of the proposed smart contract. Therefore, the proposed smart contract first gives the requirement to select candidate smart contracts for cooperation. A participant P2P smart contract provides its data (or state) to indicate that it wants to cooperate with the smart contract.

Candidate smart contracts should meet the requirements of the proposed smart contract.

A P2P smart contract has two corresponding fields, a requirement field (*requirement set*) for selecting candidate smart contracts and a status field (*data set*) for matching the requirements of other smart contracts.

Requirement of Status. There is a challenge to the state requirements of other smart contracts. If the current state of the smart contract meets the requirements, does it meet expectations throughout the whole topics process? That is, a smart contract can set its state upon verification to match the requirements of others, and change the state back upon actual execution. In this way, other participants are deceived.

Therefore, the status of the smart contract should meet two requirements to avoid this cheating.

(1) The status field should be verifiable. A participant can easily fake any value in this field if it cannot be verified.

(2) The status field should ensure that future behavior is also correct. To analyze this problem, we use the notation of D_s^t to represent the state D provided by the smart contract s at time t. The state change from D to D' at a later time $t+\delta$ should be restricted. The following status D' and its change condition C should be known to the relevant participants.

We introduce some conditions that satisfy the above requirements.

(a) Conditions that cannot be changed. The sender of a smart contract [10] is such a condition, which comes from a specific account and cannot be changed. Therefore, one smart contract may require another smart contract to be from a specific account. Take a company as an example. An employee's leave request should be approved from a specific address (that of the Human Resources Department or its manager).

(b) Conditions that should be changed under certain constraints. An example of fulfilling requirement (b) is when digital assets are frozen. The sender of the smart contract locks its assets into a smart contract (this asset is called frozen asset [11] [12]). The sender cannot transfer the asset back because the asset owner has changed. The transfer of frozen assets has clear conditions (in the smart contract): transferring the asset to the receiver when the conditions match [13], or returning the asset to the sender when there is no matching condition within a certain period of time [14].

B. Topics

Different P2P smart contracts may be dependent [15] or not, depending on whether they match the requirements of other smart contracts. We introduce a new concept, *topic*, to describe a set of cooperative P2P smart contracts. Suppose there is a set of smart contracts SC.

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

(1)Each smart contract has its own dependent smart contracts, and all dependent smart contracts are in the smart contract set SC. That is, all dependent smart contracts are in this topic, and there is no dependence on external smart contracts.

(2)Any two smart contracts have a dependence. Otherwise, there may be two or more unrelated smart contracts in one topic, which can be divided into another topic.

1) Topic Identification: Topic identification is the process by which miners determine which topic a smart contract belongs to. There are different ways to identify a topic. An easy way to do this is to compare the target smart contract with all smart contracts. This method is called the all matching method.

The complexity of this method is $nO(n)$ (n is the number of smart contracts). The reason is that it needs to compare across all smart contracts until a matching contract is found. If there are numerous smart contracts, judging whether one smart contract matches the conditions of another smart contract is a heavy workload.

Then, we introduce another method to reduce the workload of topic identification, the group matching method.

Group matching method. Smart contracts are first divided into different groups, and then the target smart contracts are

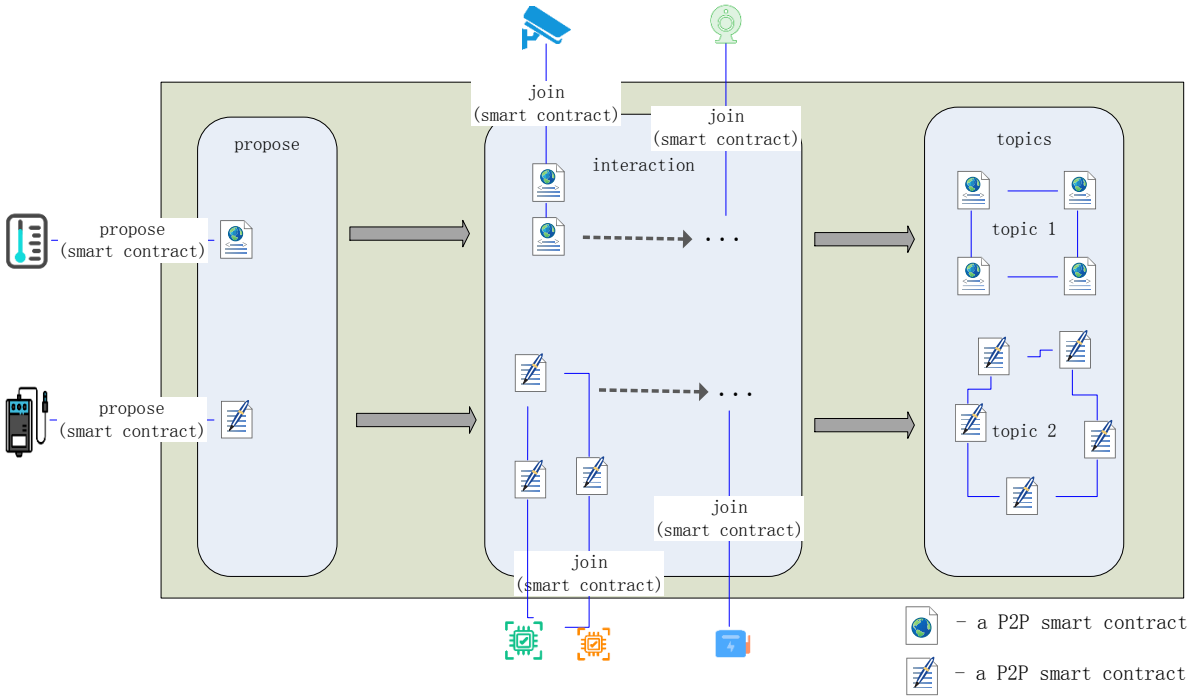

Fig. 1. The interaction process of P2P smart contracts. This process is divided into three steps. In the first step, a device proposes cooperation through a P2P smart contract. In the second step, other devices join the cooperation through their smart contracts. In the third step, cooperation (or a topic) is formed when the relevant smart contracts are matched. There are two topics in the figure, topic 1 and topic 2.

compared within their groups. The workload of the method is smaller than that of the all matching method, since the smart contract only needs to compare within its group. To identify different groups, each smart contract has a group identifier. The group identifier is set in the proposal smart contract. Other interested smart contracts specify the same identifier to join. The complexity of the algorithm is a constant $O(1)$.

C. Topics Choice - Transaction Order Based Selection

There may be several candidates competing for a topic. For example, a user (User UA) wants to exchange his assets and specifies the requirements for the exchange. Two users (users UB and UC) both want to exchange with UA, each sending a P2P smart contract. Assume that both UB and UC meet the requirements of UA. There is the question of how to choose one of them.

In this paper, the approach we take is based on when smart contracts appear (the earlier, the better). Other methods include the reward-based method, which considers the reward given to miners as it reflects the willingness to join the topic.

Since the sending time of smart contracts is difficult to obtain in some blockchains, the order of appearance of smart contracts is more convenient. Therefore, the order of smart contracts is used to measure the time when smart contracts appear. We begin with the relevant definitions.

Definition 1: Smart Contract Order. All related smart contracts are sorted in the order they were put into the blockchain. The order is first determined by the block index (rule SCO_1); if two smart contracts are in the same block, it is determined by their order in that block (rule SCO_2). In a topic, we number each smart contract starting from 1, which represents the relative order in which the smart contracts appear. For different topics, since it is difficult to number across topics, we compare them according to rules SCO_1 and SCO_2.

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

Definition 2: Different Smart Contract between 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 that sc is a different smart contracts between topic tp_a and tp_b. It is used to describe the differences between candidate topics.

We use $d(tp_1, tp_2, i)$ to represent i_{th} different smart contracts in topic tp_1 between topics tp_1 and tp_2, $d(tp_2, tp_1, i)$ represents that in topic tp_2. For example, if tp_1 is $\{sc_1, sc_2, sc_3\}$, and tp_2 is $\{sc_1, sc_4, sc_5\}$, then $d(tp_1, tp_2, 1)$ is sc_2, $d(tp_2, tp_1, 1)$ is sc_4, and $d(tp_1, tp_2, 2)$ is sc_3.

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

We now describe different strategies based on transaction order, which are called the topic-choosing algorithms.

1) First DSC Tactic: First DSC is the first different smart contract between the two candidate topics. For topics 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 multiple candidate

tactics, we choose the topic tp_c whose first DSC has the smallest order. The algorithm 1 shows pseudocode for selecting the first DSC between two topics.

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
Topic chooseCompletion(Topic t1, Topic t2){
if order(d(t1,1) <d(t2,i) then
    return t1
end if
if order(d(t1,1) >d(t2,i) then
    return t2
end if
//error case as different smart contracts has different orders
return ERROR
}
```

Algorithm 1 can be applied to scenarios between multiple topics. The algorithm first compares different topics two by two, and then compares the selected results, and so on. The selection process forms a binary tree, called a comparison tree. For example, for the four candidate topics tp_1, tp_2, tp_3, tp_4, we can select the target topic according to the algorithm 2, and the comparison tree is shown in the figure 2.

Meanwhile, we call the sequence of comparing different topic pairs as *iteration order*. For example, the iteration order of Figure 2 is (tp_1, tp_2), (tp_3, tp_4), ($tp_1 1$, tp_2).

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

```
Require: t1, t2, t3, t4 are four candidate topics
Ensure: the chosen topic among them
Topic chooseCompletion(Topic t1, Topic t2, Topic t3,
Topic t4){
t11 = chooseCompletion(t1, t2)
t12 = chooseCompletion(t3, t4)
return chooseCompletion(t11, t12);
}
```

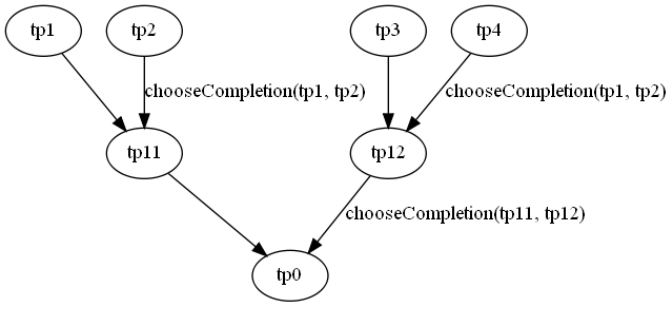

Fig. 2. Comparison tree to choose a topic among four topics, tp_1, tp_2, tp_3, and tp_4. tp_{11} is the chosen topic between tp_1 and tp_2; and tp_{12} is that between tp_3 and tp_4. tp_0 is the finally chosen topic.

2) Last Smart Contract Tactic: In some cases, we want to choose a topic that completes the earliest, which is equivalent to its last smart contract completing the earliest. We use sc_{last} to represent the last smart contract. Since sc_{last} is the last smart contract, it has the highest number in topic tp.

Similarly, we choose the topic tp_c with the smallest order of sc_{last}.

There may be cases where the last smart contract is the same between two topics, then we further compare the completion time of the penultimate smart contract, and so on, until a smart contract with a different completion time is found. Meanwhile, different topics must have a different smart contract, otherwise, the two topics are the same. This process is shown in algorithm 3, and we first introduce a related concept.

Definition 3: The $_nth$ last smart contract pair (lastPair(n)) is the $_nth$ smart contracts from the last smart contract.

For example, two topics tp_1 is $\{sc_1, ...,sc_{n-2},..., sc_{n-1}, sc_n\}$, and tp_2 is $\{sc'_1, ...,sc'_{n-2}, sc'_{n-1}, sc'_n\}$. The first last smart contract pair (**lastPair(1)**) is sc_n and sc'_n. The penultimate smart contract pair (**lastPair(2)**) is sc_{n-1} and sc'_{n-1}.

Algorithm 3 To choose the topic with the last smart contract tactic between two topics.

```
Require: t1, t2 are two candidate topics
Ensure: the chosen topic between t1 and t2
Topic chooseCompletion(Topic t1, Topic t2){
//length(tp) is a function that returns the number of smart
contract in topic tp
ln = min(length(t1), length(t2))
for i = 1 to ln do
    if order(lastPari(i, t1)) >order(lastPari(i, t2)) then
        return t2
    end if
    if order(lastPari(i, t1)) <order(lastPari(i, t2)) then
        return t1
    end if
end for
}
```

D. Deterministic Results and Turning-complete Logics of P2P Smart Contract

1) Deterministic Results:

Lemma 1: A topic-choosing algorithm is deterministic if it gives the same results from the same input.

Proof 2.1: First, the input to the topic-choosing algorithm is deterministic because when the containing blocks of smart contracts are put into the main chain, their order is fixed. Therefore, when the topic-choosing algorithm matches the conditions (gives the same result for the same input), the chosen topic is deterministic.

Lemma 2: The first DSC outputs deterministic topics regardless of *iteration order*.

Proof 2.2: Suppose the candidate tactics are tp_1, ..., tp_i, ..., tp_n. For any two candidate topics, tp_i and tp_j, at least one different smart contract exists; otherwise, they are the same tactic. The min function will return a deterministic result (the one with the lesser order) between tp_i and tp_j. This iterates through all candidate topics, and in each iteration, the first

subject with the smaller DSC is selected. Therefore, the last selected topic is the first one with the smallest DSC.

For example, there are three topics tp_1 (sc1, sc3, sc8), tp_2 (sc1, sc4, sc7) and tp_3 (sc2, sc5, sc8). If the order is $\min(\min(tp_1, tp_2), tp_3)$, tp_1 and tp_2 are compared first, and tp_3 is compared finally. The result is tp_1. If the order is $\min(\min(tp_1, tp_3), tp_2)$, tp_1, tp_3 are compared first, and tp_2 is compared last. The result is also tp_1. Similarly, for $\min(\min(tp_2, tp_3), tp_1)$, the result is tp_1. Although the iteration order is different, the results are the same.

Then no matter what the iteration order is, the first DSC algorithm outputs a deterministic topic.

Lemma 3: No matter what the *iteration order* is, the last smart contract tactic outputs a deterministic topic.

Proof 2.3: We still assume that there are several topics, tp_1, ..., tp_i, ..., tp_n. The algorithm compares the order of smart contracts in reverse order until it finds a smart contract with a different order, and picks a topic with a smaller order.

(1) Assuming the smallest one is the last smart contract, no matter what order the paired smart contracts are selected for comparison, the topic containing the smallest order will be selected.

(2) If the last smart contract in all topics is the same and we delete the last smart contract, it is the same as (1).

(3) If the last two smart contracts are the same, we can remove the last two smart contracts. The case is also similar to (1). This can be iterated if more smart contracts are the same.

(4) After deleting some smart contracts in step (2) or (3), if a topic has no elements, the topic will be selected. Meanwhile, no more than one topic will be empty at the same time; otherwise, they will be the same topic. Therefore, the results are deterministic.

Theorem 1: The proposed model outputs deterministic results.

Proof 2.4: In the proposed model, there are two topic-choosing tactics, the first DSC tactic and the last smart contract tactic. From the above analysis, we know that both are deterministic. Therefore, the results of the proposed model are deterministic.

2) Turning-complete Interaction:

Lemma 4: If a blockchain supports a Turning-complete smart contract, a user can interact with others in a Turning-complete way with a P2P smart contract.

Proof 2.5: When a blockchain supports (1) the Turning-complete smart contacts and (2) the P2P smart contract model, a user can specify its condition to interact with others in its own smart contract. And the P2P model does not conflict with any Turning-complete condition, then the user can interact with others in a Turning-complete way.

III. VERIFICATION

A. Test Environment and Test Scenarios

Currently available blockchains do not provide a running environment for P2P smart contracts, and then a self-developed blockchain is used for verification. The verification blockchain provides support for P2P smart contracts. (1) Miners check whether a smart contract is a P2P smart contract; (2) If it is a P2P smart contract, miners try to find a topic through one of the choosing algorithms; (3) If a topic is formed, then all P2P smart contracts continue to run. The consensus algorithm for this blockchain is PoW (Proof of Work), and the difficulty is set to the degree that a block's hash starts with at least 6 zeros, which results in an average mining cycle of tens of seconds in the verification environment.

TABLE I
PEER 2 PEER EXCHANGE

device id	requirement	data	exchange identifier	test case
u1	5 assets	1 asset	101	e1
u2	6 assets	2 asset	102	e2
u3	7 assets	3 asset	103	e3
u4	8 assets	4 asset	104	e4
u5	1 asset	5 asset	101	e5
u6	2 assets	6 asset	102	e6
u7	3 assets	7 asset	103	e7
u8	4 assets	8 asset	104	e8

Verification scenarios are asset exchanges between IoT devices that use their own assets to exchange other assets. These assets are different digital coins in the blockchain. This can be done in either the centralized way or the P2P way. The exchange has two parts: part 1, how much asset it wants to offer, and part 2, how much asset it wants to get from the paired exchanger. In the centralized way, one smart contract is used to manage the exchange process. IoT devices specify part 1 and part 2 of the exchange through parameters. In the P2P approach, each IoT device specifies its exchange through a P2P smart contract, where the smart contract's requirements are part2 and the status part is part1.

There are four paired exchanges (with the same exchange identifier), as shown in table I. The requirement part is what the device expects from its peer, and the data part is what the device promises to provide to its peer. In the P2P approach, each exchange is identified by an identifier, the exchange identifier. The st_{oe} code part of the code is very simple, it only logs a success message when the exchange is complete.

The sequence of interaction with the blockchain is from device u1 to device u8. When proposal devices (i.e. devices from u1 to u4) send their exchange first, paired devices (i.e. devices from u5 to u8) send their exchange. This sequence helps to see if an exchange affects others.

To compare robustness, we set a smart contract out of service in each round of testing. In the P2P smart contract approach, we choose one of P2P smart contracts from device u1 to device u8 to be out of service. In the centralized way, the centralized smart contract goes out of service when the chosen device interacts with it. Therefore, there are 8 test cases (named e1, e2 to e8) in both the P2P way and the centralized way.

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

The comparison is between the p2p smart contract approach (p2p approach) and the centralized smart contract approach

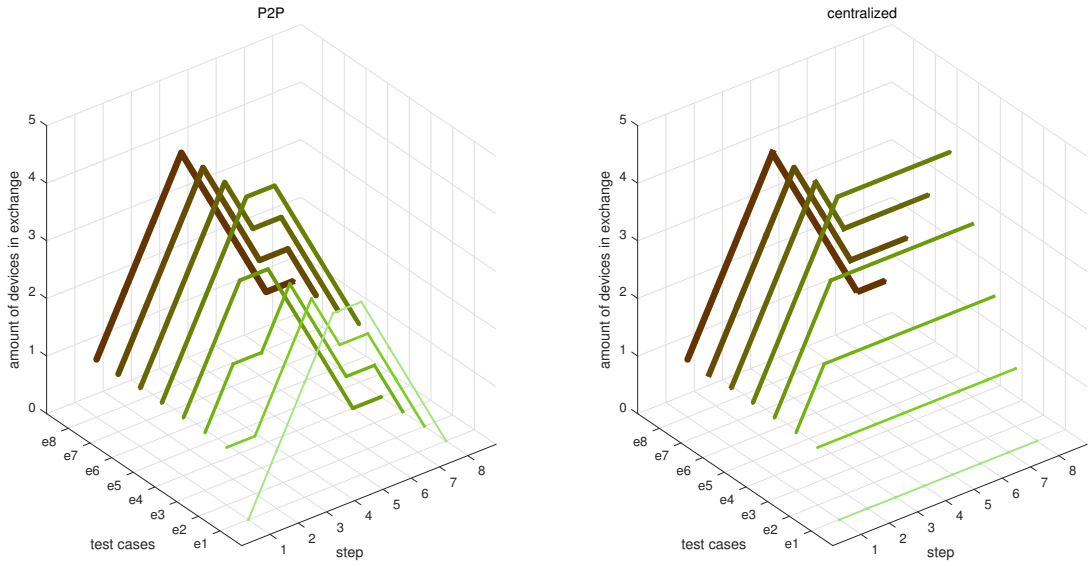

Fig. 3. The number of affected device during the process. In P2P method, failure of a participant only affect its paired exchanger. In the centralized method, all successive participants are affected when the failure of the centralized smart contract.

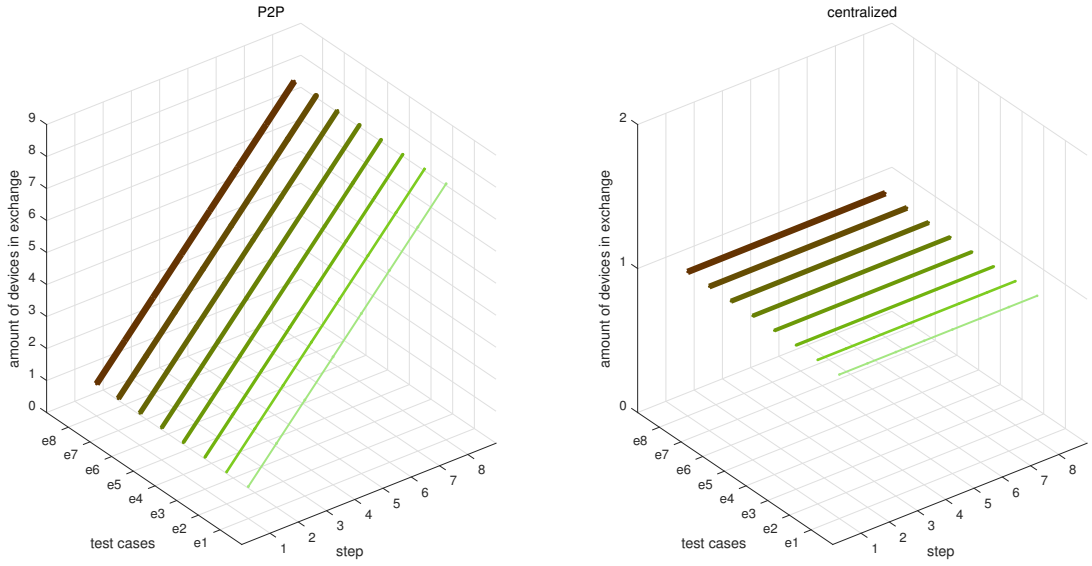

Fig. 4. Number of separately processing programs.

(centralized approach). The purpose is to verify that P2P smart contracts are more feasible to express logic and have higher fault tolerance than centralized methods.

The comparison is in two aspects, the affected device, and the number of separately processing programs. The affected device help to understand the difference between the P2P method the centralized method. The number of separately processing programs help to understand the background reason.

1) Affected Devices: P2P methods isolate the actions of different participants. The failure of one participant does not affect the actions of the other participants. In this section, we will focus on the affected devices. Affected devices are those that interact with the blockchain and cannot be processed further because their associated smart contracts are out of service. Figure 3 shows the results for the number of affected devices (in test cases from e1 to e8).

From the Figure 3, we see that the affected device also has two distinct phases. The first stage is the period when the fault occurs from device u1 to device u5 (test cases from e1 to e5). In the centralized approach, the number of affected devices

is accumulated from 0 to 4. Comparatively, no devices were affected in the P2P approach. Even the device sending the faulty smart contract will not be affected because its smart contract is out of service.

The second stage is from the moment when the faults happen when device u6 joins (test cases from e6 to e8). As some exchanges are done, the number of affected devices decreased. However, with a centralized approach, when the smart contract fails, other unfinished cannot successfully exchange. Then the number of affected devices is 3, 2, and 1 for e6, e7, and e8, respectively. In the P2P approach, there is 1 affected device, the paired exchanger of the faulty smart contract. Therefore, we conclude that fewer devices are affected in the P2P approach than in the centralized approach.

2) Separately Processing Programs: Separately processing programs are smart contracts that process device requests independently. The failure of one does not affect the processing of another. It helps to further understand the difference between affected assets and affected devices.

There are 4 exchanges in this verification. In a centralized approach, one smart contract processes all exchanges, therefore, there is only one processing program. If this one fails, no further exchanges can be made. Comparatively, the P2P approach has 8 processing programs, with paired smart contracts forming an exchange.

In this verification, each smart contract records the corresponding information to a separate log file when it starts, which is intended to facilitate counting the number of separately processing smart contracts. The log files are cleaned up at the beginning of the test case, and we count the number of log files after each device sends its request either by a P2P smart contract or its parameters. The results are shown in Figure 4. There are more separately processing programs in the P2P approach (up to 8 programs) than in the centralized approach (1 program). Therefore, when one separately processing program is out of service, there are other separately processing programs to handle service in the P2P way, which indicates higher robustness.

IV. Conclusion

In this paper, we address the issue of how to achieve high availability. The method is to use the fully decentralized application model, which requires both hardware and software to run in a P2P way. The P2P application model has more robustness and allows customizing of different requirements of devices. We implement the model by the P2P smart contracts on a blockchain. Analytical and experimental results show that we provide a fully decentralized way to achieve high availability for IoT devices.

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] Norita Ahmad, Phil Laplante, and Joanna F DeFranco. Life, iot, and the pursuit of happiness. *IT Professional*, 22(6):4–7, 2020.

[2] Asmae Blilat and Abdelali Ibriz. Design and implementation of p2p based mobile app for collaborative learning in higher education. 2020.

[3] Sumendra Yogarayan, Siti Fatimah Abdul Razak, Afizan Azman, and Mohd Fikri Azli Abdullah. A mini review of peer-to-peer (p2p) for vehicular communication. *Indonesian Journal of Electrical Engineering and Informatics (IJEEI)*, 9(1):185–197, 2021.

[4] Natarajan Deepa, Quoc-Viet Pham, Dinh C Nguyen, Sweta Bhattacharya, B Prabadevi, Thippa Reddy Gadekallu, Praveen Kumar Reddy Maddikunta, Fang Fang, and Pubudu N Pathirana. A survey on blockchain for big data: approaches, opportunities, and future directions. *Future Generation Computer Systems*, 2022.

[5] Hong Su, Bing Guo, Yan Shen, and Xinhua Suo. Embedding smart contract in blockchain transactions to improve flexibility for the iot. *IEEE Internet of Things Journal*, 2022.

[6] Daojing He, Zhi Deng, Yuxing Zhang, Sammy Chan, Yao Cheng, and Nadra Guizani. Smart contract vulnerability analysis and security audit. *IEEE Network*, 34(5):276–282, 2020.

[7] Alfredo J Perez and Sherali Zeadally. Secure and privacy-preserving crowdsensing using smart contracts: Issues and solutions. *Computer Science Review*, 43:100450, 2022.

[8] Cameron McPhail, Holger R Maier, Seth Westra, Leon van der Linden, and Jan H Kwakkel. Guidance framework and software for understanding and achieving system robustness. *Environmental Modelling & Software*, 142:105059, 2021.

[9] Ayman Esmat, Martijn de Vos, Yashar Ghiassi-Farrokhfal, Peter Palensky, and Dick Epema. A novel decentralized platform for peer-to-peer energy trading market with blockchain technology. *Applied Energy*, 282:116123, 2021.

[10] Yaoqing Liu, Guchuan Sun, and Stephanie Schuckers. Enabling secure and privacy preserving identity management via smart contract. In *2019 IEEE conference on communications and network security (CNS)*, pages 1–8. IEEE, 2019.

[11] Hong Su, Bing Guo, Jun Yu Lu, and Xinhua Suo. Cross-chain exchange by transaction dependence with conditional transaction method. *Soft Computing*, 26(3):961–976, 2022.

[12] Yan Zhu, Weijing Song, Di Wang, Di Ma, and William Cheng-Chung Chu. Ta-spesc: Toward asset-driven smart contract language supporting ownership transaction and rule-based generation on blockchain. *IEEE Transactions on Reliability*, 70(3):1255–1270, 2021.

[13] Narges Shadab, Farzin Houshmand, and Mohsen Lesani. Cross-chain transactions. In *2020 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, pages 1–9. IEEE, 2020.

[14] Qian Zhang, Sheng Cao, and Xiaosong Zhang. Enabling auction-based cross-blockchain protocol for online anonymous payment. In *2021 IEEE 27th International Conference on Parallel and Distributed Systems (ICPADS)*, pages 715–722. IEEE, 2021.

[15] Christian Gorenflo, Lukasz Golab, and Srinivasan Keshav. Xox fabric: A hybrid approach to blockchain transaction execution. In *2020 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, pages 1–9. IEEE, 2020.