Li *et al. J Wireless Com Network* (2021) 2021:168
https://doi.org/10.1186/s13638-021-02040-z

EURASIP Journal on Wireless Communications and Networking

RESEARCH Open Access

TFCrowd: a blockchain-based crowdsourcing framework with enhanced trustworthiness and fairness

Chunxiao Li, Xidi Qu and Yu Guo*

*Correspondence:
yuguo@bnu.edu.cn
School of Artificial Intelligence, Beijing Normal University, Beijing, China

Abstract

Blockchain technology has attracted considerable attention due to the boom of cryptocurrencies and decentralized applications. Among them, the emerging blockchain-based crowdsourcing is a typical paradigm, which gets rid of centralized cloud-servers and leverages smart contracts to realize task recommendation and reward distribution. However, there are still two critical issues yet to be solved urgently. First, malicious evaluation from crowdsourcing requesters will result in honest workers not getting the rewards they deserve even if they have provided valuable solutions. Second, unfair evaluation and reward distribution can lead to low enthusiasm for work. Therefore, the above problems will seriously hinder the development of blockchain-based crowdsourcing platforms. In this paper, we propose a new blockchain-based crowdsourcing framework with enhanced trustworthiness and fairness, named TFCrowd. The core idea of TFCrowd is utilizing a smart contract of blockchain as a trusted authority to fairly evaluate contributions and allocate rewards. To this end, we devise a reputation-based evaluation mechanism to punish the requester who behaves as "false-reporting" and a *Shapley value*-based method to distribute rewards fairly. By using our proposed schemes, TFCrowd can prevent malicious requesters from making unfair comments and reward honest workers according to their contributions. Extensive simulations and the experiment results demonstrate that TFCrowd can protect the interests of workers and distribute rewards fairly.

Keywords: Crowdsourcing, Reputation, Shapley value, Smart contract, Blockchain

1 Introduction

With the prosperity of sharing economy, crowdsourcing has become a considerable computing paradigm which allows people from different countries and regions to accomplish the same task together. More and more companies and individuals begin to outsource tasks and recruit freelancers all over the world through crowdsourcing platforms. Crowdsourcing platforms help companies find the right workers around the world and reduce their costs. Meanwhile, crowdsourcing platforms help many freelancers find jobs they are interested in. This is of great significance to the redistribution of resources and social stability in the world. Under this background, a lot of applications

have been developed, such as Upwork [1], CrowdFlower [2], and Amazon Mechanical Turk [3].

In traditional crowdsourcing platforms, there are three roles: requesters, workers, and brokers. The requesters publish their tasks to the broker, and the workers submit their interests to the broker. The broker matches the task requirements with the interests of workers and recommends the best matched tasks to the workers. After collecting task results from the workers, the requesters will accept the genuine solutions and reward the workers with payment.

Traditional crowdsourcing platforms are normally based on a centralized cloud-server for managing tasks and matching workers with aligned interests. Such centralized architectures are often vulnerable to single-point failure and lack operation transparency. Once the servers are attacked, the whole crowdsourcing platform will break down. For instance, Elance-oDesk encountered DDoS attacks in May 2014 [4] and its services broke down. In April 2015, Uber suffered a hardware failure that caused passengers cannot stop their orders at the end of services [5]. Meanwhile, traditional crowdsourcing platforms are notorious for their untrustworthiness and unfairness. Brokers and requesters may collude, or brokers may encounter financial problems resulting in requesters and workers unable to get back their deposits. In 2018, ofo sharing bicycles encountered financial problems and the users could not get their deposits back. To address these problems, a feasible solution is to build a crowdsourcing platform based on blockchain [6–9]. Blockchain is a shared, distributed ledger that facilitates the process of recording transactions and tracking assets in a business network. Due to the intrinsic advantages of decentralization and immutability, blockchain-based systems are more difficult to be attacked comparing with centralized systems [10–12]. Thus, we are motivated to develop a blockchain-based crowdsourcing platform.

Although a blockchain-based crowdsourcing platform is robust and reliable, there are two challenging issues yet to be solved. The first challenge is that the requester may behave as "false-reporting." If the requesters undervalue the workers' solution maliciously, the workers will not be paid even if they contribute a high-level solution. The second challenge is how to fairly evaluate and reward workers' contributions. Since the contribution of each worker is different, it would be unfair to the workers who contribute more if we reward them equally. This situation will diminish better workers' enthusiasm. Therefore, the above problems will seriously hinder the development of blockchain-based crowdsourcing platforms.

In this paper, we propose TFCrowd, a new blockchain-based crowdsourcing framework with trustworthiness and fairness. Our design utilizes the trustworthy smart contracts of blockchain to transparently calculate the requesters' reputation and allocate the workers' rewards. To prevent the "false-reporting" issue, we propose a reputation-based evaluation mechanism to punish malicious requesters. With our evaluation mechanism, in the long run, when honest workers submit high-quality results, the requesters who misreport them as low-quality results will spend more. Besides, we further explore a method based on *Shapley value* to distribute rewards for workers. Our proposed method can evaluate each worker's contribution via the smart contract and assign the rewards fairly. We conduct extensive experiments to prove that TFCrowd is effective in protecting workers' interests and distributing rewards fairly.

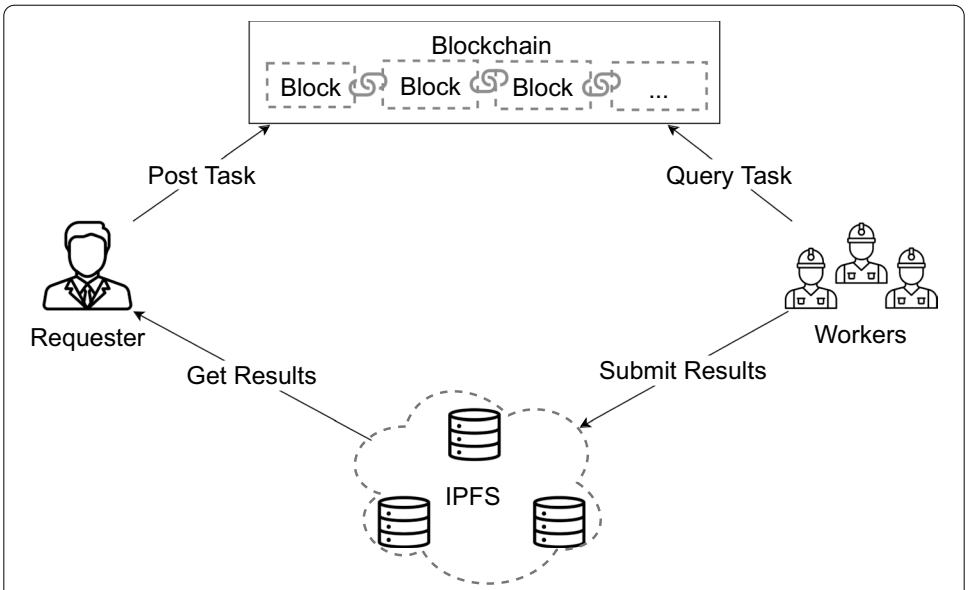


Fig. 1 Overview of TFCrowd architecture

In summary, the contributions of this paper are listed as follows:

- We propose a new blockchain-based crowdsourcing framework with trustworthiness and fairness, named TFCrowd. By utilizing smart contracts, our proposed blockchain-based framework enables credible result evaluation and fair reward distribution.
- We design a reputation-based evaluation mechanism to prevent the requester from making malicious comments. Besides, we devise a method based on *Shapley value* that can distribute the crowdsourcing rewards fairly according to the contribution of each worker.
- We implement the system prototype and conduct simulation experiments. The experimental results demonstrate that our design can effectively punish malicious requesters and reward honest workers fairly.

The reminder of this paper is organized as follows. In Sect. 2, we present the system overview and workflow. The detailed design of TFCrowd is given in Sect. 3. Experimental results are demonstrated in Sect. 4. In Sect. 5, we present the related work. We analyze and discuss the results of TFCrowd in Sect. 6. The whole paper is concluded in Sect. 7.

2 System overview

2.1 Architecture of TFCrowd

As shown in Fig. 1, the architecture of TFCrowd contains four types of components: requester, workers, blockchain, and IPFS [13]. Requesters publish their tasks on the platform. Workers query tasks those they are interested in on the platform and post their

solutions to IPFS. All the transactions are recorded on the blockchain. The detail of each component is described as follows:

- **Requester**: The requester has crowdsourcing tasks to be completed, which are hard for computer to accomplish, but easy for human, such as image labeling and environmental data collection. The requester finds suitable workers through blockchain-based crowdsourcing platform. The requester posts his task, containing task description and the deadline, to the crowdsourcing platform. To attract workers to solve the task together, the requester transfers a certain amount of deposit to the smart contract. The amount of deposit depends on the workloads of the task.
- **Blockchain**: The blockchain integrated with smart contract serves as a trusted crowdsourcing platform to match the requester and workers. Smart contract can execute the predefined contracts automatically to control the whole workflow. Specifically, it knows who is authorized to sign in and how to deal with the registrations. It calculates the quotation of the task and updates the reputation of the requester. If there is something wrong with the transaction, the smart contract serves as a judge to select miners to handle the conflicts. At the end of transaction, smart contract distributes the rewards to the workers.
- **Workers**:The workers are a group of persons who have certain skills to complete crowdsourcing tasks. They use the blockchain-based crowdsourcing platform to find tasks they are interested in. A worker will receive a message from the smart contract when a new task is published and decide whether to take the task or not.
- **IPFS**: The IPFS is a secure and distributed database system. Since transaction will generate a lot of crowdsourcing data, it is not feasible to store a large amount of data in blockchain. Thus, in our design, we use IPFS to save the task data published by the requester and the task results uploaded by the workers.

2.2 Workflow

As shown in Fig. 2, the process of TFCrowd contains the following steps:

- Initialization

  (1) The requester and the workers sign up, respectively. Each of them generates a pair of keys and sends the public key to the smart contract.
  (2) Smart contract checks the user's registration information and sends its public key to the authorized users. In the following steps, the requester and workers encrypt their private data with the public key generated by the smart contract and transfer the data to the smart contract.

- Task Management

  (3) The requester sends the task requirements to the IPFS and gets an address returned by the IPFS. Then, the requester sends the task address to the smart contract.

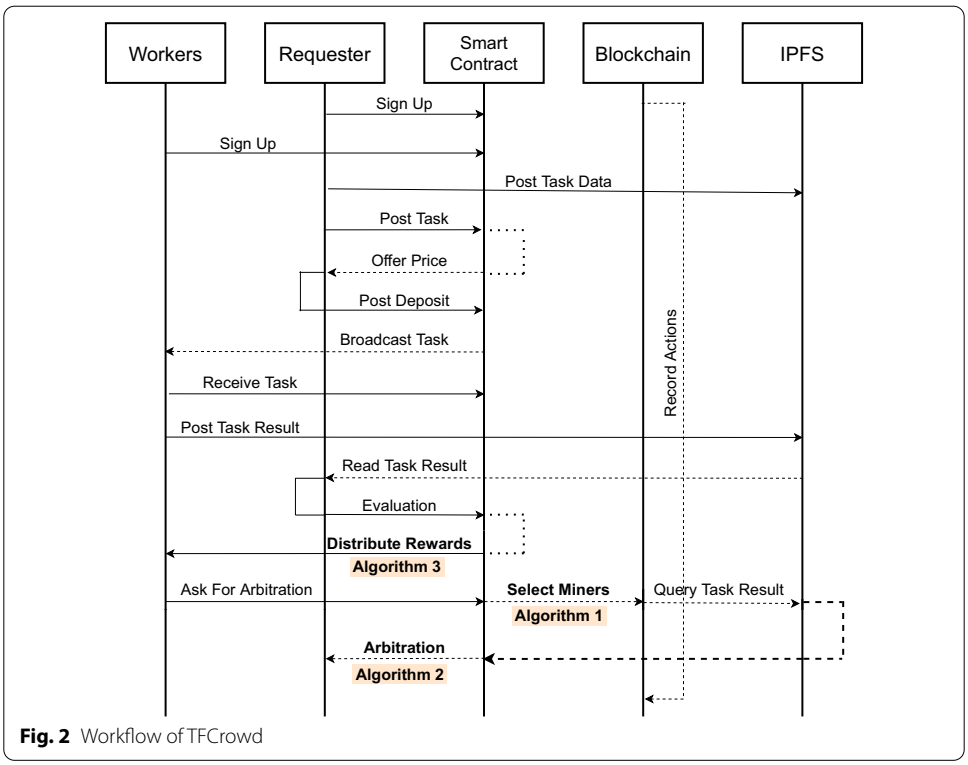


Fig. 2 Workflow of TFCrowd

(4) When the smart contract receives the request from the requester to publish the task, it checks the reputation of the requester and the workloads of the task. Then, smart contract calculates the corresponding quotation and sends the quotation to the requester.
(5) When the requester receives the quotation, he has to pay the deposit to the smart contract; otherwise, his task will not be published.
(6) When the smart contract receives the deposit, it will publish the task description and broadcast the task description to the workers.
(7) When the workers get the message from smart contract, they will check the task type, workloads, and the deadline. If the worker is available and interested in the task, he will take the task and notify the smart contract.
(8) For simplicity, we omit the description of the working process. When the workers finish the task, they will send the task results to the IPFS and get the data address returned by the IPFS. Then, the worker will notify the smart contract he has uploaded the data of the task results and the address saved on the IPFS.

- Task Evaluation

(9) When the deadline is reached or all the workers have submitted their solutions, the smart contract will notify the requester to check the task results. Then, the requester will download the task results and give a remark about the quality of the task result. The requester submits the evaluation to the smart contract.

(10) If the evaluation value is bigger than or equal to the given threshold, the smart contract will calculate how to distribute the deposit and transfer the corresponding rewards to the workers. If the evaluation value is less than the given threshold, the smart contract will refund the deposit to the requester.

- Arbitration

(11) If the workers do not agree with the evaluation, they will ask for arbitration. When the smart contract receives the request for arbitration from the workers, it will select some miners to form a committee to deal with the conflicts.

(12) When the smart contract completes the trial of the case, it will know whether the requester gives a bad remark maliciously or not. Meanwhile, the smart contract will update the reputation of the requester.

2.3 Design goal

Our proposed framework includes only one requester, who does not worry about competing with other requesters. If the requester gives an unreasonable comment on the task results uploaded by the workers, he can redeem his deposit after he receives the task results. This may lead to a behavior known as "false-reporting." We will design a reputation-based mechanism to penalize the requester who always gives unreasonable comments deviating from the truth and protect the interests of the workers. Besides, the contributions of the workers are unequal, so we design *Shapley value*-based method to distribute the revenue with fairness.

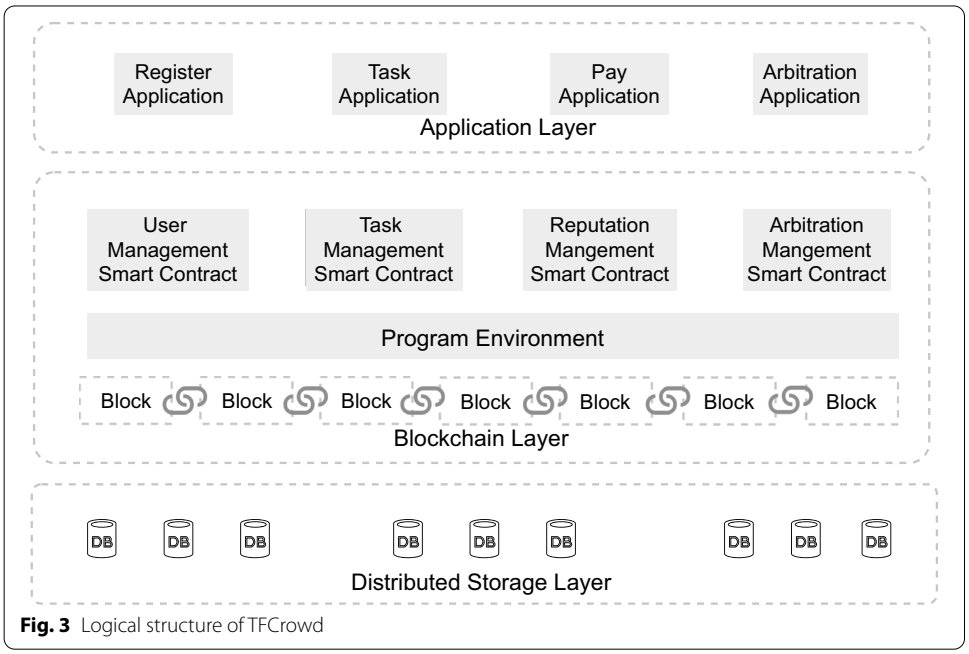


Fig. 3 Logical structure of TFCrowd

3 Methods

As shown in Fig. 3, the system consists of three layers. First, the distributed storage layer supplies a distributed database that saves task-related data. It stores the data from the requester and the workers, and it returns the data address to the data owners.

Second, the blockchain layer realizes the blockchain, which ensures a secure and trustworthy environment and smart contract. Smart contract consists of four predefined contracts: user management contract, task management contract, reputation management contract, arbitration management contract. Besides, this layer has a programming environment that consists of an Integrated Development Environment and program compiler. The Integrated Development Environment is a user graphic interface that allows users to type codes online. The compiler compiles the codes to smart contract.

Third, the application layer is the interface where the requester and the workers interface with the crowdsourcing platform. There are four units in this layer: register application unit, task application unit, pay application unit, and arbitration application unit.

3.1 Initialization

To protect users' privacy and security, only the permitted users can join in the TFCrowd. If the requester and the workers agree with the predefined contracts, they can register via the register application. When smart contract permits the requester's registration, it will return a tuple (*address*, *reputation*, *state*) to the requester. With address, smart contract can find the requester easily. Reputation value indicates the honesty of the requester. The requester has two states: normal state and untrust state. In normal state, requester can evaluate the solution's quality and redeem his deposit. In untrust state, requester has no rights to redeem his deposit and workers can receive the revenues every time.

3.2 Task management

When the requester is permitted to access the crowdsourcing platform, he will send the task demands which contains task description and deadline to the IPFS and get an address from IPFS. Then, the requester sends a request which contains the task address to smart contract for publishing the task. When receiving the request, smart contract will evaluate the workloads of the task and check the requester's reputation. Then, smart contract calculates the primal quotation C_0 of the task. In order to motivate the requester behaves well, we add a coefficient based on requester's reputation. Therefore, the final price is as below:

$$C = C_0 e^{\frac{1}{2}-R} \tag{1}$$

R is the reputation of the requester. We will introduce how to calculate R in the arbitration subsection. Next, the requester prepays a deposit of C to smart contract. After these actions, the task will be published in the blockchain and smart contract will broadcast the task to matched workers. Once the task is published, all workers can query and join in the task.

3.3 Evaluation

When a worker has finished his work of the task, he will submit his solution to the IPFS and gets an address from the IPFS. When he gets the solution's address, he notifies smart contract that he has uploaded his solution to IPFS and sends the solution's address. When all the recruited workers have uploaded their solutions or the deadline is reached, smart contract will notify the requester to check the task results saved on the IPFS. The requester rates the quality of the task results into 100 levels from 0.0 to 10.0. 0.0 represents none of the solutions is valid, and 10.0 represents all of the solutions are valid. We define a fixed threshold 6.0 as the requester's minimum requirement. Specifically, if the level is below 6.0, we define the solutions as not meeting the requester's minimum requirement. In this circumstances, the requester can redeem his deposit. If the level is over 6.0, we define the solutions as meeting the requester's requirement even if the task results are not perfect.

3.4 Arbitration

There is a risk in the system that the requester can redeem his deposit by giving a malicious evaluation even if the workers contribute high-quality solutions. In this condition, it is unfair to the workers because they work hard but cannot get rewards. So the workers will ask smart contract for arbitration, and the punishment is needed if the requester lies. The workers will send *task_address* to smart contract. With *task_address*, smart contract can find the workers who accomplish the task and their solutions. Meanwhile, smart contract can find the evaluation of the requester with *task_address*.

When smart contract receives the arbitration request, it will randomly select some miners to form a committee. Algorithm 1 can ensure the selection progress is random. The input is a tuple $(seeds_w, seed_r, m)$. The variable $seeds_w$ is a list of random integers, $seed_r$ is a random integer, and m is the number of miners to be selected. In the beginning, every worker and the requester are required to input an integer to the smart contract. Smart contract sorts M by miner's address. Smart contract selects one integer from $seeds_w$ once and sums it and $seed_r$, then smart contract calculates the hash value of the sum. With the result $rand_t$, smart contract calculates the remainder of $rand_t$ divided by n and gets the result t. n is the number of available miners. Smart contract selects t-th miner as the first committee member. When the first miner is selected, smart contract repeats the above steps until m miners are selected. When the committee is formed, they will check the quality of all the task results. Algorithm 2 is the algorithm of arbitration. According to the variable *task_address*, the committee can get the task description, the addresses of the workers who receive this task, the evaluation of the requester, and the task results saved on the IPFS. The m miners will be divided into two groups. Each group checks the result independently, and both of them have to give a score to evaluate the result. If the difference value is bigger than 1.0, smart contract will select $m/2$ miners to form the third group by Algorithm 1. The third group will check the result again. So there are three scores and smart contract will select two scores whose difference value is the least. Then, smart contract finds the average value of the two scores.

Algorithm 1 : Select miners to form committee

Input: Minsers set,M; Numbers of members of committee N; Random integer list $seeds1$, given by the workers; Random integer $seeds2$,given by the requester.
Output: Selected miners set, SM, to form a committee.

$n = len(M)$
$sort(M)$
for $seed_i$ in $seeds1$ **do**
  $rand_t = Hash(seed_i + seeds2)$
  $index = rand_t$ mod n
  $m = m - 1$
  $n = n - 1$
  $sort(M)$
  $SM.append(index)$
  if $meq0$ **then**
    stop
  end if
end for
return SM

We define the evaluation value of the requester as E_r and the evaluation value of the committee as E_c. We define the reputation of the requester as Eq. (2).

$$R_T = \begin{cases} r_0, & T = 1 \\ R_{T-1} + \frac{1}{1+e^{-\rho_0 d}} - \frac{1}{2}, & T > 1 \end{cases} \tag{2}$$

$$y = \frac{1}{1 + e^{-\rho x}} \tag{3}$$

r_0 is the initialized reputation of the requester which ranges in [0, 1]. ρ_0 is the coefficient denoting the degree of the R_T function. d is the value that E_r minus E_c. The equation of reputation is motivated by the sigmoid function because it has three features. (1) The sigmoid function is a point symmetry function. We can set the initial value of the requester at the symmetry point. (2) The slope of the function is large near the symmetry point. The reputation of the requester will fall or increase faster at the initial state. (3) This function has an upper bound and a lower bound. The price will fluctuate in a range. The sigmoid function is shown in Eq. (3). The figure of the sigmoid function is shown in Fig. 4.

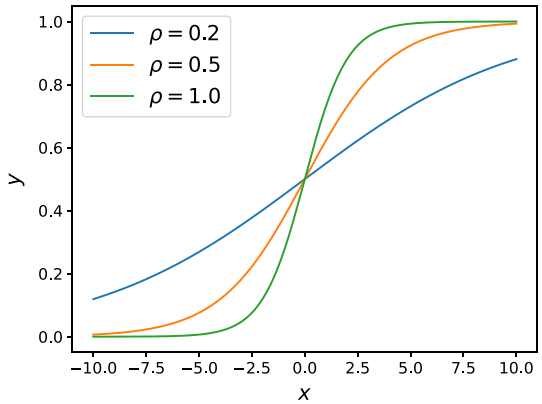


Fig. 4 Graph of sigmoid with different coefficient

3.5 Initialization

From Fig. 4, we can find $y \in (0, 1)$ and as ρ_0 increasing the degree becomes bigger too. From Eq. (2), we can deduce that R_T ranges from 0 to 1. If $E_r < E_c$, i.e., $x < 0$, we can find $y < 0.5$. So if the requester always tells a lie, his reputation will gradually decline. We define two threshold values R_l and R_h. If the requester's reputation falls below R_l, the requester's state becomes the untrust state from the normal state. In the untrust state, smart contract will distribute the requester's deposit to the workers whether the task result is good or not and the requester has no right to redeem his deposit.

If workers contribute low-level solutions in untrust state, the requester can ask smart contract for arbitration. Smart contract will select miners to check the solution again. If the quality of the solutions are lower than a default value, smart contract will improve the requester's reputation. Specifically, we set the default evaluation value of the requester 6.0 and calculate the reputation via Eq. (2). In untrust state, if the workers contribute high-quality solutions constantly, the reputation of the requester will remain the same. In this condition, the requester will have no chance to become an honest requester when he regrets his malicious comments. In order to avoid this issue, if the requester does not ask for arbitration, smart contract updates requester's reputation value by adding 0.1.

When the requester's reputation in untrust state exceeds R_h, his untrust state changes into the normal state. In the normal state, the requester can redeem his deposit if he evaluates the result as low quality.

Algorithm 2 : Arbitration

Input: Task address T_{addr}.
Output: miners' evaluation of worker's solution E_m; reputation value of the requester R_T; requester's state S_T.

$Task = FindTaskByAddress(T_{addr})$
$R_0 = FindReputaionOfReqestor()$
$S_0 = FindStateOfReqestor()$
$Solutions = FindSolutionByAddress(T_{addr})$
$evaluation = FindEvaluatioinByAddress(T_{addr})$ // the remark value gived by requester
$sum = 0$
$n = len(Solutions)$
for $Solution$ in $Solutions$ **do**
    $value = evaluate(Task, Solution)$ // returns 0 or 1;0 represents the solution is invalid and 1 represents the solution is effective.
    $sum = sum + value$
end for
$S_m = 10 * sum/n$ // map the score to [0,10]
$R_T = R_{T-1} + 1/(1 + math.pow(e, (evaluation - E_m) * \rho_0))) - 0.5$
if $S_0 isnormalstate$ **then**
    if $R_T \leq R_l$ **then**
        $S_T = untruest \quad state$
    else
        $S_T = normal \quad state$
    end if
else
    if $R_T \geq R_h$ **then**
        $S_T = normal \quad state$
    else
        $S_T = untruest \quad state$
    end if
end if
$S_m = sum/n$
return S_m, R_T, S_T

Algorithm 3 : Reward Distribution

Input: Task address T_{addr}.
Output: $[r_1, r_2, ..., r_k]$. // reward of every worker who participate the task

$Task = FindTaskByAddress(T_{addr})$
$W = FindWorkersOfTask(T_{addr})$
$Solutions = FindSolutionByAddress(T_{addr})$
$rev = FindDepositByAddress(T_{addr})$
$n_W = len(W)$
$Shapley = Array()$
$r = Array()$
for w in W **do**
    $n_w = evaluate(Task, Solution)$// the amount of m-th worker' solution
    $S_w = FindSetExclue(w)$
    $S = FindAllSubsetsOf(S_w)$
    $n_S = len(S)$
    $v_S = H(n_S)$
    $v_{S,w} = H(n_S + n_w) - H(n_S)$
    $tsh = 0$
    for S in S_w **do**
        $temp = (n_W - n_S - 1)! \times v_{S,w} \div n_W$
        $tsh = tsh + temp$
    end for
    $Shapley.append(tsh)$
end for
$Shapley_{sum} = sum(Shapley)$
for s in Shapley **do**
    $tempr = rev \times s \div Shapley_{sum}$
    $r.append(tempr)$
end for
return r

3.6 Reward distribution

It is important to reward every worker fairly especially when their contributions are unequal. Some methods have been developed to solve the rewards distribution. Among these methods, *Shapley value* is the most powerful one. *Shapley value* was proposed by Lloyd Shapley [14], which is a solution concept of fairly distributing revenues to several workers in a coalition. The *Shapley value* is used in situations when the contributions of each worker are unequal, but all the workers accomplish the whole task together. Essentially, the *Shapley value* is the average expected marginal contribution of one player after all possible combinations have been considered. It has been proven that using *Shapley value* is a fair approach to allocate value.

We denote the set of workers as W and $|W|$ is the number of workers. S is a non-empty subset of W and it represents a group of workers. The worth function of the group S is denoted as $v(S)$, which represents the contribution of the group. We use w_i denotes the i-th worker who is not in the group S. So the marginal contribution of w_i to the group S is defined as:

$$v(S, i) = v(S \cup \{i\}) - v(S) \tag{4}$$

The *Shapley value* is defined as

$$\phi(w_i) = \sum_{S \in S_i} \frac{(|W| - |S| - 1)!|S|!}{|W|!} v(S, i) \tag{5}$$

S_i is a collection of all subsets of w excluding w_i.

It is important to distribute the rewards fairly according to workers' contributions. In order to measure the contribution of each worker, we take the task of data collection or

image annotation as examples. In the task of data collection, the more data samples, the more accurate the final result of the task. In addition, the contribution of the latter collected samples is less than that of the former ones. So we use the function defined by Eq. (6) as the measurement of the contribution.

$$H(x) = 1 - e^{-\rho_1 x} \tag{6}$$

x is the amount of samples uploaded by a worker. For example, w_1 uploads 5 samples first and then, w_2 uploads 2 samples. To simplify the problem, we define ρ_0 as 1. The contribution of w_1 is $1 - e^{-5}$. The total contribution of w_1 and w_2 is $1 - e^{-7}$. So the contribution of w_2 is $1 - e^{-7} - (1 - e^{-5})$. If w_2 uploads his 2 samples first, the contribution of w_2 is $1 - e^{-2}$. And the contribution of w_1 who has 5 samples is $1 - e^{-7} - (1 - e^{-2})$. We can calculate every worker's contribution through Eq. (5). Algorithm 3 is designed for reward distribution.

4 Experiments

In this section, we examine the effectiveness of the TFCrowd. First, we examine the relationship between the price and the reputation, and then we examine the fairness of the reward distribution.

4.1 Experimental setting

We use a random float list ranging from 0 to 10 to simulate the quality of the task results, and it is also the value list of the committee's evaluation. This list follows a normal distribution. The mean value is 8, and the variance is 2. We use a python function *random.normalvariate*(8, 2) to generate 1000 random values. If the value is greater than 10, then it is set to be 10. If the value of the sample point is less than 0, then it is set to be 0. Figure 5 shows the first 100 values. It is obvious that most of the points fall in the area bigger than 6.0.

4.2 Reputation

We designed three groups of simulation experiments. In the first group of experiments, we examine the malicious requester's cost in normal state and Fig. 6 shows the results of 6 experiments. In the second group of experiments, we examine the malicious requester's cost in untrust state and Fig. 7 shows the result of the experiment.

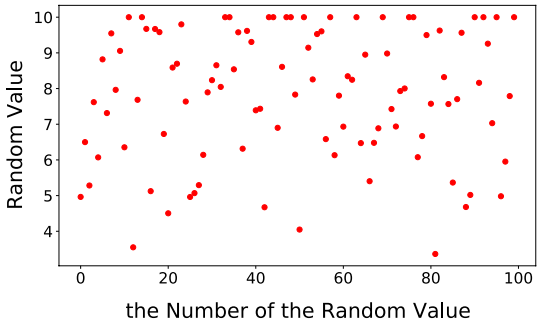


Fig. 5 The first 100 values of the quality of the task results

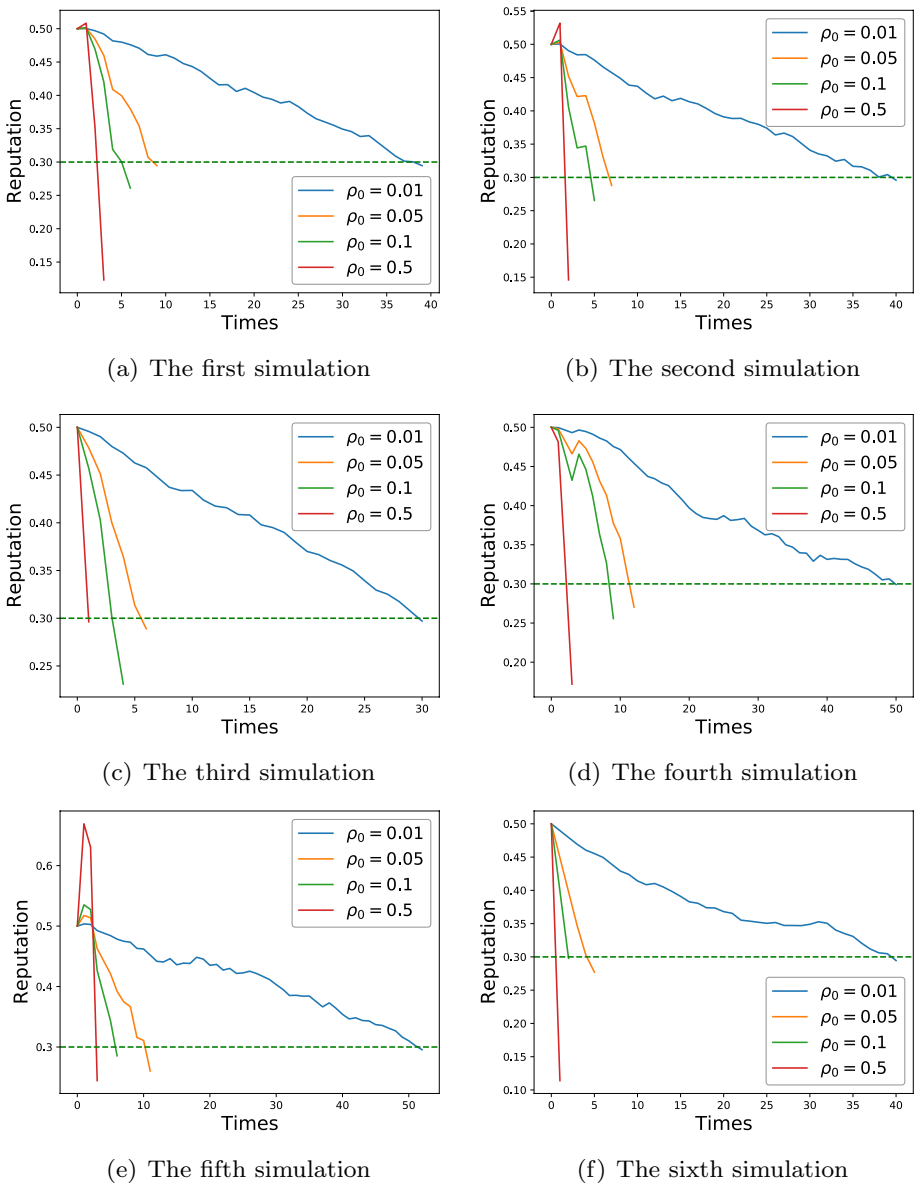
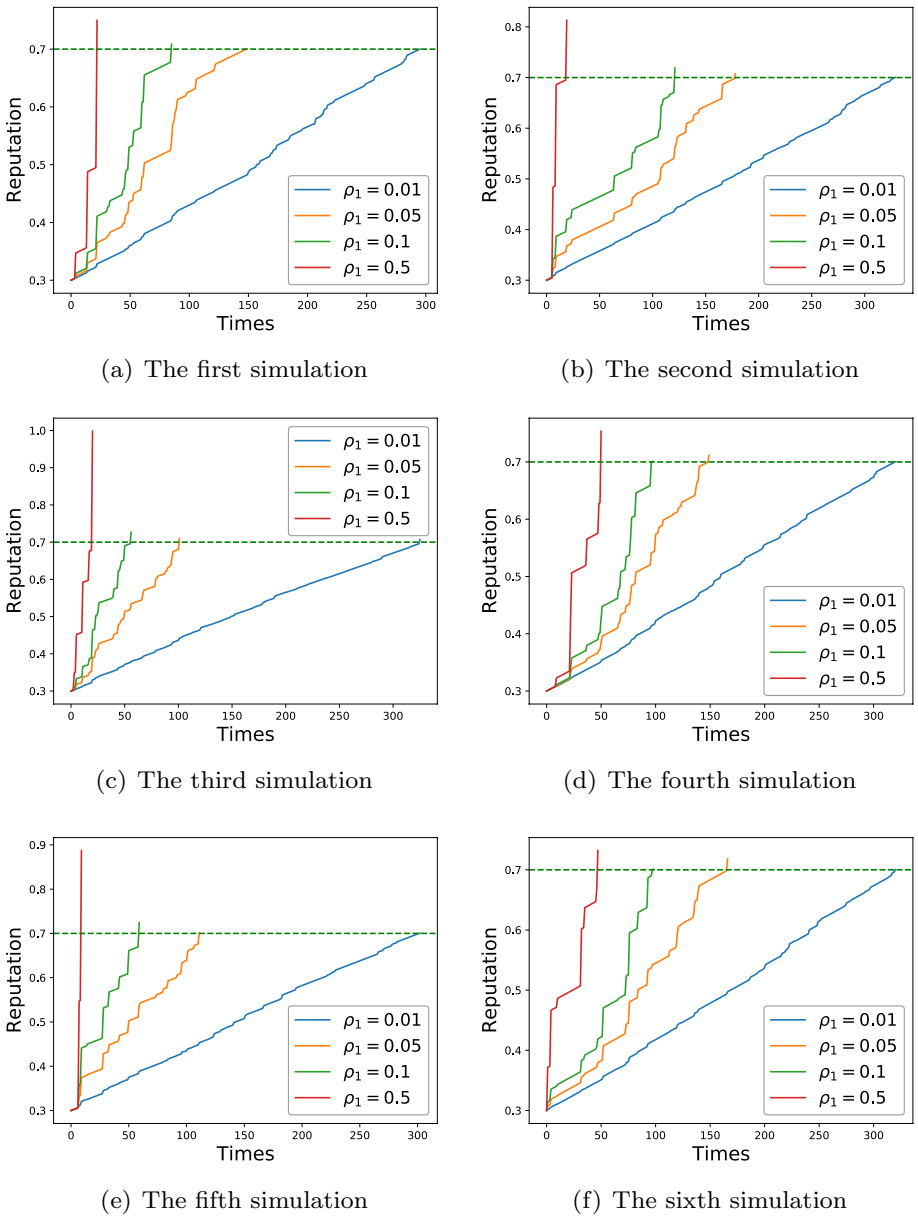


Fig. 6 The number of transactions for reputation values drop from 0.5 to 0.3

In normal state, if the malicious requester wants to get the results without any cost, it is a best choice for the requester to rate every high-quality result as 5.9. In this condition, the requester can redeem his deposit and downgrade his reputation slowly. In untrust state, the requester cannot redeem his deposit, so it is a best choice to keep silence when the workers contribute a high-quality solution and ask for arbitration when the workers contribute a low-quality solution. We initialize the reputation of the requester with 5.0. We set R_l to 0.3 and W_l to 0.7. In Fig. 6, it is obvious that as ρ_0 goes bigger, the reputation declines faster. However, the quality of the workers' solutions is different in every simulation, so the results differ from each other. In order to study the relationship between ρ_0 and the requester's cost, we conduct 1000 experiments and take the average value to compare the cost of the requester when lies or



(a) The first simulation (b) The second simulation

(c) The third simulation (d) The fourth simulation

(e) The fifth simulation (f) The sixth simulation

Fig. 7 The number of transactions for reputation values raise from 0.3 to 0.7

not and the results are shown in Tables 1 and 2. For simplicity, we assume the primal quotation of all the tasks are the same, which is denote as C_0.

When the requester's state becomes trust state from untrust state, his reputation value is 7.0 and he may give malicious evaluation again. Because in the initial state, the requester's reputation value is 0.5, we conduct the third group of experiments to examine the malicious requester's cost when his reputation value dropped from 0.7 to 0.3. The experimental results are shown in Table 3.

From Table 1, we can know when $\rho_0 = 0.5$, the requester's reputation will fall below 0.3 after 2 times. As the ρ_0 goes down, the times will increase. When $\rho_0 = 0.01$, the requester evaluate the results malicious at least 30 times before he becomes untrustworthy. If the

Table 1 The results of the first group of experiments

ρ_0	min(n)	max(n)	avg(n)	Cost	Average normal cost	Average save
0.01	30	61	43.11	0	$43.11C_0$	$43.11C_0$
0.05	5	23	9.1	0	$9.1C_0$	$9.1C_0$
0.1	2	10	4.83	0	$4.83C_0$	$4.83C_0$
0.5	1	5	1.65	0	$1.65C_0$	$1.65C_0$

Table 2 The results of the second group of experiments

ρ_1	min(n)	max(n)	avg(n)	Cost	Average normal cost	Average save
0.05	91	187	137.79	$169.37C_0$	$137.79C_0$	$-31.58C_0$
0.1	30	154	86.48	$106.82C_0$	$86.48C_0$	$-20.34C_0$
0.5	3	68	26.5	$33.59C_0$	$26.5C_0$	$-7.09C_0$

Table 3 The results of the third group of experiments

ρ_1	min(n)	max(n)	avg(n)	Cost	Average normal cost	Average save
0.05	10	32	17.6	0	$18.6C_0$	$18.6C_0$
0.1	5	19	8.83	0	$9.83C_0$	$9.83C_0$
0.5	2	7	2.62	0	$3.62C_0$	$3.62C_0$

requester maliciously evaluates the results more than 30 times, we can define him as a malicious requestor forever. And the workers will not get reward more than 30 times. So, the value of ρ_0 should not be set too small. And, in the second and third experiments, we omit the case of $\rho = 0.01$.

From Tables 2 and 3, we can compare the cost of the requester. We can see when $\rho_1 = 0.5$, the requester saves $3.62C_0$ but spend $7.09C_0$ more. If the requester evaluate the results maliciously, he will spend more than he saves.

4.3 Shapley value

The evaluation function of our experiment is Eq. (6). We take a set of three workers w_1, w_2, w_3 as an example. We assume that w_1 completes the workload of 1 samples and w_2 completes the workload of 3 samples and w_3 completes the workload of 5 samples. So the evaluation value of the total results is $1 - e^{-9\rho_1}$. We use (w_1, w_2, w_3) as the completion sequence. So the contribution value of w_1, w_2, w_3 is as Table 4 shows. The average contribution of w_1 is as shown in Eq. (7).

$$\frac{1 - e^{-1\rho_1}}{3} + \frac{e^{-3\rho_1} - e^{-4\rho_1}}{6} + \frac{e^{-5\rho_1} - e^{-6\rho_1}}{6} + \frac{e^{-8\rho_1} - e^{-9\rho_1}}{3}. \tag{7}$$

And we can also calculate the *Shapley value* according to Eq. (5).

Because

$$S_1 = \phi, \{w_2\}, \{w_3\}, \{w_2, w_3\},$$

and

$$v(\phi, w_1) = v(\phi \cup w_1) - v(\phi) = v(w_1),$$

we have

$$\begin{aligned} v(\{w_2\}, w_1) &= v(\{w_2\} \cup w_1) - v(w_2) = v(\{w_1, w_2\}) - v(w_2), \\ v(\{w_3\}, w_1) &= v(\{w_3\} \cup w_1) - v(w_3), \\ v(\{w_2, w_3\}, w_1) &= v(\{w_2, w_3\} \cup w_1) - v(\{w_2, w_3\}), \end{aligned}$$

and

$$\begin{aligned} \phi(w_1) &= \sum_{S \in S_1} \frac{(|w| - |S| - 1)!|S|!}{|w|!} v(S, i) \\ &= \frac{1 - e^{-1\rho_1}}{3} + \frac{e^{-3\rho_1} - e^{-4\rho_1}}{6} + \frac{e^{-5\rho_1} - e^{-6\rho_1}}{6} + \frac{e^{-8\rho_1} - e^{-9\rho_1}}{3} \end{aligned}$$

We can see the result of $\phi_1(w)$ is equal to Eq. (7). Figure 8 shows the distribution of *Shapley value* with different ρ_1. We can conclude that as ρ_1 increases, the difference of *Shapley value* decreases. All in all, from the simulations, it is obvious that if the requester always gives malicious evaluations instead of true evaluations, he will spend more money to motivate workers to complete his tasks in the long run. Meanwhile, the H function used as the contribution function is reasonable. We can adjust the distribution of rewards by changing the ρ_1 coefficient.

5 Related work

5.1 Blockchain

With the rapid growth of blockchain, the smart contract has become a hot research topic. More and more decentralized applications have implemented smart contracts as a trusted authority to enhance their services robustness. Zhou et al. [15] implemented a smart contract with a witness model to tackle service violations. Specifically, the authors proposed to use smart contract to select a group of miners randomly as the committee to solve the service violations. In [16], the authors built a stored-value card platform with the smart contract on Ethereum. With this platform, the agreement of stored-value

Table 4 Contributions distribution

	w_1	w_2	w_3
(w_1, w_2, w_3)	$1 - e^{-1\rho_1}$	$e^{-1\rho_1} - e^{-4\rho_1}$	$e^{-4\rho_1} - e^{-9\rho_1}$
(w_1, w_3, w_2)	$1 - e^{-1\rho_1}$	$e^{-6\rho_1} - e^{-9\rho_1}$	$e^{-1\rho_1} - e^{-6\rho_1}$
(w_2, w_1, w_3)	$e^{-3\rho_1} - e^{-4\rho_1}$	$1 - e^{-3\rho_1}$	$e^{-4\rho_1} - e^{-9\rho_1}$
(w_2, w_3, w_1)	$e^{-8\rho_1} - e^{-9\rho_1}$	$1 - e^{-3\rho_1}$	$e^{-3\rho_1} - e^{-8\rho_1}$
(w_3, w_1, w_2)	$e^{-5\rho_1} - e^{-6\rho_1}$	$e^{-6\rho_1} - e^{-9\rho_1}$	$1 - e^{-5\rho_1}$
(w_3, w_2, w_1)	$e^{-8\rho_1} - e^{-9\rho_1}$	$e^{-5\rho_1} - e^{-8\rho_1}$	$1 - e^{-5\rho_1}$

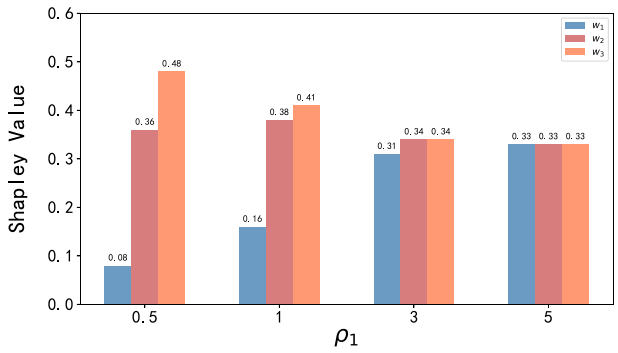


Fig. 8 Workers' shapley values

cards is more clear, the customers can refund more easily and all customers can realize the transfer of stored-value.

5.2 Crowdsourcing

With the rapid development of the internet and the sharing economy, cooperation among people has become more and more frequent. Crowdsourcing as a bridge between the requester and the workers has attracted a lot of attention. More and more researchers and companies study and implement crowdsourcing to complete some complex tasks. In [17, 18], the authors pointed out that traditional crowdsourcing platforms based on a centralized architecture are vulnerable to be attacked. Also, the unfairness of the third party was discussed in [17, 19, 20]. Moreover, in [17], blockchain-based mobile crowdsourcing was designed, satisfying a decentralized and distributed management in mobile crowdsourcing to collect data more efficiently. Moreover, the authors of [18] proposed a BC-FGA-DCrowd scheme, which ensured the correctness and fairness of requirements for a data trading scheme. Furthermore, in [19, 21, 22], the authors introduced the development and the strength of blockchain and described how to implement a safe and trustworthy crowdsourcing with privacy protection. Zhu, Kane et al. [23] proposed a data-driven crowdsourcing quality control model for tasks distributed in parallel.

5.3 Malicious evaluation

In a transaction, there are at least two roles: buyers and sellers and sometimes a third party. Sellers provide goods, services, and solutions. Buyers buy goods, services, and solutions from sellers and pay rewards to sellers. However, buyers may undervalue the goods, solutions maliciously, and refuse to pay rewards to the workers. In this condition, buyers can get the services or solutions without rewarding sellers. This dishonest behavior is harmful to the transaction and is not sustainable. In order to protect the sellers, malicious evaluations are needed to be eliminated. There are two main methods to discourage the buyers to evaluate the solutions maliciously. The first method is evaluating the solutions by a trusted third party instead of buyers. The other method is designing a reputation mechanism to penalize the buyer who always gives bad comments. In [15], Zhou et al. implemented a smart contract with a witness model to tackle service violations. In this paper, customers bought services from the ISP directly and evaluate the services. Smart contract selected a group of miners randomly as the trusted third

party to solve the service violations. In [24], Zhang et al. designed a mechanism EFF which eliminates dishonest behaviors with the help of a trusted third party for arbitration. Reputation-based mechanisms are typical incentive mechanisms, which introduce reputation or reputation-like metric to evaluate the credits of the users and give the corresponding reward or punishment [25–28]. Specifically, in [25], Sun proposed a framework to analyze reputation-based incentive mechanisms for P2P services. In [20, 26–31], the authors built reputation systems or grading systems to avoid dishonest behaviors. In [27, 28], reputation-based mechanisms were used to address "false-reporting" and "free-riding" behaviors. [32] implemented reputation-based trading system to encourage the participants to adopt a long-term solution in emission reduction.

6 Results and discussion

The most widespread evaluation mechanism of crowdsourcing relies on a trusted third-party of authority to evaluate the work reliably and reasonably. The trustworthiness of the third-party determines the security and stability of the whole system. If the authority colludes with other participants or is attacked by adversaries, it will bring irreparable losses. Moreover, the evaluation results directly affect the reward distribution for workers. A malicious or compromised third-party of authority undervalues the workers' contributions, resulting in a "free lunch" to the requester or more rewards to unworthy workers. Therefore, it will bring unfairness and lead to decreased activity of workers who cannot receive reasonable rewards. Overall, the contribution of this work can be summarized as follows:

- **Decentralized evaluation for trustworthiness** : Our blockchain-based evaluation mechanism draws support from the blockchain with the intrinsic advantages of decentralization, transparency, traceability, and immutability, which is more difficult to be attacked comparing with centralized systems. Moreover, the trustworthy smart contracts of blockchain can transparently maintain a reputation value to prevent the "false-reporting" of the requester. Our proposed reputation-based evaluation mechanism punishes the malicious requester who misreports workers' contribution as low quality.
- **Fair distribution for activating workers** : Only when workers can reap fair and worthy rewards, they will have enough motivation to continue to participate in crowdsourcing. We model crowdsourcing as a cooperative game, one of the best solution of which is *Shapley value* for fair allocation of rewards. Our proposed *Shapley value*-based distribution method can evaluate each worker's contribution via the smart contract and allocate the rewards fairly.

In this work, we ignore the privacy-preserving requirement of workers' data, which is not suitable to be openly accessed by all participants (e.g., miners, other workers) in a blockchain-based crowdsourcing platform. Cryptography technologies, such as asymmetric encryption and homomorphic encryption schemes, will be utilized to solve this challenge in future work. In this paper, we only consider the case of one requester, because when more requesters join in, they will compete with each other and the

influencing factors will increase. Therefore, our mechanism is not suitable for the case of multiple requesters participating. We will consider the scenario of more requesters in future work.

7 Conclusion

In this paper, we propose a new blockchain-based crowdsourcing framework with trustworthiness and fairness name TFCrowd. In order to prevent requester from making malicious comments, we design a reputation-based mechanism to penalize the requester. Besides, we propose a reward distribution method based on *Shapley value* to distribute rewards with fairness. We simulate the transactions with different coefficients to prove the reputation-based mechanism is effective, which motivates the requester to behave well. Meanwhile, our experimental results prove that the *Shapley value*-based method can distribute rewards fairly.

Abbreviations

IPFS: InterPlanetary File System; n: the number of experiments; min (*n*): the minimum number of experiments; max (*n*): the maximum number of experiments; avg (*n*): the average number of experiments.

Acknowledgements

This work was supported by the Fundamental Research Funds for the Central Universities under Grants 2020NTST32 (Grant No. 310421108).

Authors' contributions

C.L. initiated this project and designed the framework. X.Q. and Y.G. have been involved in drafting the manuscript. All authors read and approved the final manuscript.

Declarations

Competing interests

The authors declare that they have no competing interests.

Received: 20 January 2021 Accepted: 2 August 2021
Published online: 19 August 2021

References

1. Upwork: Upwork project (2015). Online at https://www.upwork.com/
2. CrowdFlower: Crowdflower project (2015). Online at https://www.crowdflower.com/
3. A.M. Turk, Amazon mechanical turk project (2015). Online at https://www.mturk.com/
4. Elance and odesk hit by ddos (2014). Online at https://gigaom.com/2014/03/18/elance-hit-by-major-ddosattack-downing-service-for-many-freelancers/
5. Uber china statement on service outage (2015). Online at http://shanghaiist.com/
6. Y. Guo, H. Xie, Y. Miao, C. Wang, X. Jia, Fedcrowd: A federated and privacy-preserving crowdsourcing platform on blockchain. IEEE Trans. Serv. Comput. **16**, 3068–3081 (2020)
7. M. Li, J. Weng, A. Yang, W. Lu, Y. Zhang, L. Hou, J.-N. Liu, Y. Xiang, R.H. Deng, Crowdbc: a blockchain-based decentralized framework for crowdsourcing. IEEE Trans. Parallel Distrib. Syst. **30**(6), 1251–1266 (2018)
8. C. Zhang, Y. Guo, X. Jia, C. Wang, H. Du, Enabling proxy-free privacy-preserving and federated crowdsourcing by using blockchain. IEEE IoT-J. **8**, 6624–6636 (2020)
9. Z. Chen, Y. Guo, H. Du, X. Jia, Pfcrowd: privacy-preserving and federated crowdsourcing framework by using blockchain, in *Proceedings of IEEE IWQoS* (2020)
10. E. Toch, Crowdsourcing privacy preferences in context-aware applications. Pers. Ubiquitous Comput. **18**(1), 129–141 (2014)
11. S. Zhang, J. Wu, S. Lu, Minimum makespan workload dissemination in dtns: making full utilization of computational surplus around, in *Proceedings of the Fourteenth ACM International Symposium on Mobile Ad Hoc Networking and Computing*, pp. 293–296 (2013)
12. J.-S. Weng, J. Weng, Y. Zhang, W. Luo, W. Lan, Benbi: scalable and dynamic access control on the northbound interface of sdn-based vanet. IEEE Trans. Veh. Technol. **68**(1), 822–831 (2019)
13. J. Benet, Ipfs-content addressed, versioned, p2p file system (2014). arXiv preprint arXiv:1407.3561
14. A.E. Roth, *The Shapley Value: Essays in Honor of Lloyd S. Shapley* (Cambridge University Press, Cambridge, 1988)

15. H. Zhou, X. Ouyang, Z. Ren, J. Su, C. de Laat, Z. Zhao, A blockchain based witness model for trustworthy cloud service level agreement enforcement, in *IEEE INFOCOM 2019-IEEE Conference on Computer Communications* (IEEE, 2019), pp. 1567–1575
16. X. Ma, J. Zhou, H. Guo, J. Wang, Design of a stored-value card platform based on smart contract, in *2019 3rd International Conference on Circuits, System and Simulation (ICCSS)* (IEEE, 2019), pp. 178–182
17. W. Feng, Z. Yan, MCS-Chain: decentralized and trustworthy mobile crowdsourcing based on blockchain. Future Gen. Comput. Syst. **95**, 649–666 (2019)
18. H. Ma, E.X. Huang, K.Y. Lam, Blockchain-based mechanism for fine-grained authorization in data crowdsourcing. Future Gen. Comput. Syst. **106**, 121–134 (2020). https://doi.org/10.1016/j.future.2019.12.037
19. Y. Ma, Y. Sun, Y. Lei, N. Qin, J. Lu, A survey of blockchain technology on security, privacy, and trust in crowdsourcing services. World Wide Web **23**(1), 393–419 (2020). https://doi.org/10.1007/s11280-019-00735-4
20. N. Delhi, T. Committee, WorkerRep : Building Trust on Crowdsourcing Platform Using Blockchain Student Name: Gurpriya Kaur Bhatia 49 (2018)
21. N. More, D. Motwani, A blockchain-based decentralized framework for crowdsourcing, in *Advances in Intelligent Systems and Computing 1200 AISC(6)* (2021), pp. 448–460
22. X. Xu, Q. Liu, X. Zhang, J. Zhang, L. Qi, W. Dou, A blockchain-powered crowdsourcing method with privacy preservation in mobile environment. IEEE Trans. Comput. Soc. Syst. **6**(6), 1407–1419 (2019). https://doi.org/10.1109/TCSS.2019.2909137
23. S. Zhu, S. Kane, J. Feng, A. Sears, A crowdsourcing quality control model for tasks distributed in parallel, p. 2501 (2012). https://doi.org/10.1145/2212776.2223826
24. X. Zhang, G. Xue, R. Yu, D. Yang, J. Tang, Keep your promise: mechanism design against free-riding and false-reporting in crowdsourcing. IEEE IoT J. **2**(6), 562–572 (2017)
25. W. Sun, S. Wang, General analysis of incentive mechanisms for peer-to-peer transmissions: a quantum game perspective, pp. 517–526 (2017)
26. G. Wang, Repshard: reputation-based sharding scheme achieves linearly scaling efficiency and security simultaneously, in *2020 IEEE International Conference on Blockchain (Blockchain)* (IEEE, 2020)
27. Y. Zhang, M. Van der Schaar, Reputation-based incentive protocols in crowdsourcing applications, in *2012 Proceedings IEEE INFOCOM* (IEEE, 2012), pp. 2140–2148
28. D. Peng, F. Wu, G. Chen, Pay as how well you do: a quality based incentive mechanism for crowdsensing, in *Proceedings of the 16th ACM International Symposium on Mobile Ad Hoc Networking and Computing* (2015), pp. 177–186
29. M. Feldman, C. Papadimitriou, J. Chuang, I. Stoica, Free-riding and whitewashing in peer-to-peer systems. IEEE J. Sel. Areas Commun. **24**(5), 1010–1019 (2006)
30. D. Carboni, Feedback based reputation on top of the bitcoin blockchain (2015). arXiv preprint arXiv:1502.01504
31. R. Dennis, G. Owen, Rep on the block: a next generation reputation system based on the blockchain, in *2015 10th International Conference for Internet Technology and Secured Transactions (ICITST)*(IEEE, 2015), pp. 131–138
32. K.N. Khaqqi, J.J. Sikorski, K. Hadinoto, M. Kraft, Incorporating seller/buyer reputation-based system in blockchain-enabled emission trading application. Appl. Energy **209**, 8–19 (2018)

Publisher's Note