

Article

# A Blockchain Approach of Model Architecture for Crowdsourcing Design Services under the Context of Social Manufacturing

Dianting Liu <sup>1,2</sup> and Dong Liang <sup>1,\*</sup>

<sup>1</sup> College of Mechanical and Control Engineering, Guilin University of Technology, Guilin 541004, China

<sup>2</sup> College of Information Science and Engineering, Guilin University of Technology, Guilin 541004, China

\* Correspondence: 1020211081@glut.edu.cn; Tel.: +86-19330430487

**Abstract:** Crowdsourcing design is generally monitored by the platform. However, the traditional crowdsourcing platforms face problems such as centralization, lack of credibility and vulnerability to single point of failure. Under the context of social manufacturing, how to address these potential issues has both research and substantial value. In this paper, we introduce decentralized blockchain technology for crowdsourcing service systems and propose a method to manage and control the process of crowdsourcing design services. We depict complex crowdsourcing logic by smart contract. The process of crowdsourcing design is not dependent on any third party. It is decentralized, tamper-proof, traceable and protects the privacy of users to a certain extent. We implement this crowdsourcing design system on a specific blockchain test network to experiment and test its functionalities. Experiment results show the feasibility and usability of our crowdsourcing design system. In the future, we will further improve the algorithmic logic of smart contracts so that they can run stably and securely in a complex node network environment.

**Keywords:** social manufacturing; crowdsourcing design; blockchain technology; smart contract



**Citation:** Liu, D.; Liang, D. A Blockchain Approach of Model Architecture for Crowdsourcing Design Services under the Context of Social Manufacturing. *Machines* **2023**, *11*, 69. <https://doi.org/10.3390/machines11010069>

Academic Editors: Pingyu Jiang, Ying Liu and Maolin Yang

Received: 8 December 2022

Revised: 1 January 2023

Accepted: 3 January 2023

Published: 5 January 2023



**Copyright:** © 2023 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

## 1. Introduction

As crowdsourcing theory has matured, it has triggered a crowdsourcing design [1] model for product development. Crowdsourcing design is based on traditional centralized crowdsourcing platforms. However, under the context of social manufacturing, crowdsourcing design requires a decentralized platform and traditional crowdsourcing platforms can hardly meet this demand.

Social manufacturing is a networking manufacturing model based on distributed, self-organized collaboration and sharing of social resources [2]. Its decentralization is an important feature that distinguishes it from other manufacturing models [3]. The decentralized and traceable features of blockchain technology exactly fit the needs of crowdsourcing design service systems under the context of social manufacturing.

Blockchain technology originated from Nakamoto's proposal of Bitcoin [4], which is a decentralized digital currency. Initially, this technology was mainly used in digital cryptocurrency systems. With blockchain platforms such as Ethereum [5] supporting smart contract [6] development and deployment, blockchain technology has shown more possibilities, such as supply-chain traceability, data sharing and crowdsourcing [7].

The idea of combining crowdsourcing platforms with blockchain is to deploy the crowdsourcing platform to the blockchain and replace the centralized platform of the traditional crowdsourcing system with the blockchain platform. The introduction of blockchain technology can bring the following advantages to crowdsourcing services [8]: solving the problem of having a single point of failure, a certain degree of privacy protection and eliminating subjective factors affecting the fairness of the decision.

At present, many scholars have applied the blockchain and crowdsourcing model to their own research fields. Ding et al. combined blockchain technology with crowdsourcing to design credit mechanisms and arbitration mechanisms to discipline the entire crowdsourcing process to ensure that users were serious about completing crowdsourcing operations [9]. In response to the problems of security and reliability in the field of agricultural products logistics, Wu et al. built a seven-level crowdsourcing platform architecture to realize a crowdsourcing-based agricultural product logistics platform based on blockchain technology [10]. In the context of crowdsourcing services, Yang et al. designed a trusted management framework for crowdsourcing intellectual property that contains four levels and three modules. Based on the blockchain on-chain and off-chain decomposition and fusion technology, it realized the credible management of crowdsourced testing intellectual property and solved the long-standing problems of irregular management and lack of credibility of crowdsourced testing services [11]. Ghaffaripour et al. proposed a zk-SNARK cryptographic primitive privacy-preserving method for crowdsourcing medical research to address the fair reciprocal exchange reward problem seen in medical relationships [12]. Gao et al. presented a blockchain crowdsourcing solution for spatial crowdsourcing tasks based on a specific rule, the Task-Select-Worker Mechanism, for sorting and assigning spatial crowdsourcing tasks [13]. Although most of the abovementioned studies have given corresponding blockchain-based application solutions or methods for their specific fields with better results, crowdsourcing design differs from the abovementioned fields in terms of business logic, task nature and worker characteristics. To accommodate these characteristics, and in conjunction with the social manufacturing environment, specialized research is needed on how to introduce blockchain technology into the crowdsourcing design model.

Therefore, in this paper, according to the characteristics of the crowdsourcing design model, we use blockchain technology, distributed memory, and the keccak256 algorithm to implement the workflow logic of crowdsourcing design services through smart contracts. A blockchain-based control method of crowdsourcing design service process in the context of social manufacturing is proposed to realize query, tracking and traceability of the crowdsourcing design service. Crowdsourcing design services are guaranteed to be decentralized, fairly and equitably executed. The main contributions of this paper are summarized as follows:

- We build a new crowdsourcing design model based on blockchain technology. Smart contracts on the blockchain are used to enable interaction between users and to ensure that the crowdsourcing design system fits into a community-based manufacturing context.
- Two types of smart contracts are designed to achieve information management and task control of crowdsourcing design users.
- We design a task state machine model in the smart contract to describe the task state of the crowdsourcing design service process, which is convenient for subsequent users to query the traceability of the crowdsourcing design tasks.
- Experiments were conducted to verify that the “blockchain and crowdsourcing” model proposed in this paper is applicable to the crowdsourcing design service process in the context of social manufacturing.

The sections of this paper are organized as follows. Section 2 presents the background knowledge of the study. Section 3 demonstrates the framework proposed in this paper. Section 4 specifies the smart contract protocols in this framework. In Section 5, we conduct an experimental analysis of the crowdsourcing design process simulation. Section 6 gives the conclusions.

## 2. Background

### 2.1. Crowdsourcing Workflow

Traditional crowdsourcing systems consist of three roles: requester, worker and a centralized crowdsourcing platform. As shown in Figure 1, the requester and the worker interact through a crowdsourcing platform. The requester uses the crowdsourcing platform

to publish task requirements. The worker receives rewards for completing the tasks on the crowdsourcing platform.

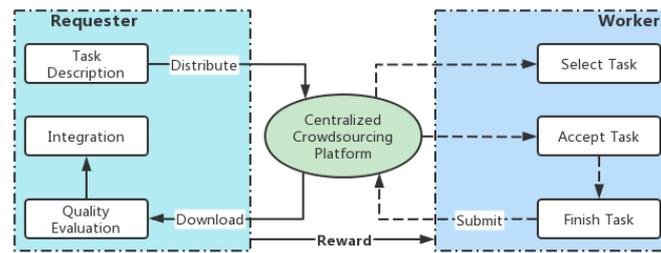


Figure 1. The workflow of traditional crowdsourcing.

Crowdsourcing platforms have a key role in the overall workflow; thus, the security of the platform is related to the whole crowdsourcing process. The traditional centralized crowdsourcing platform is vulnerable to the following problems: single point of failure problem, user-privacy-protection problem and platform-fairness problem. The above problems can discourage crowdsourcing workers and negatively affect the development of crowdsourcing platform services.

### 2.2. Blockchain

A blockchain consists of blocks linked to blocks, which is essentially a decentralized, tamper-evident, traceable and distributed database. It is maintained by each node according to the consensus mechanism; each node records the complete ledger information. Newly occurring transactions will only be allowed to be recorded on the blockchain if they have the consensus of a majority of nodes. Transactions will not be modified once they are confirmed on the chain [4]. As the data structure of the blockchain shown in Figure 2, the block header of each block contains the hash value of the previous block. If a user tampers with the transaction data of a block, it will affect all subsequent blocks and will not be able to reach consensus with the ledger maintained by other nodes, which is an important guarantee of blockchain security.

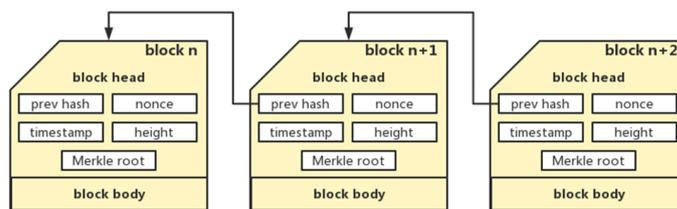
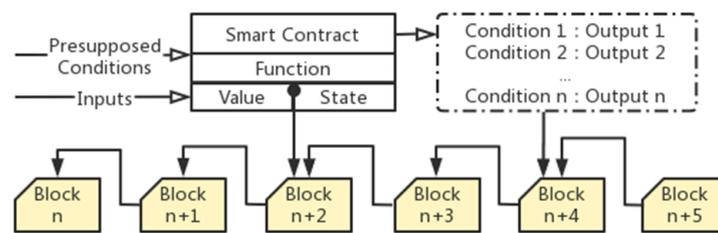


Figure 2. Data structure of the blockchain.

### 2.3. Smart Contract

In 1995, Szabo introduced the concept of smart contracts, a set of commitments defined in digital form [6]. Smart contracts have been further developed with the introduction of blockchain technology. An executable program facilitates consensus between two or more nodes in the blockchain network. Once the contract is validated and deployed on the blockchain, it cannot be tampered with. Subsequent on-chain operations that meet the trigger conditions will be executed automatically, completing the predefined transactions in the contract.

The operation mechanism of smart contracts is shown in Figure 3. In Ethereum, smart contracts encapsulate predefined variables, code execution logic and its trigger conditions, which are filled in as executable code in a transaction on the blockchain that is distributed to each network node through the P2P network. Contracts are subsequently triggered by external accounts by sending transactions. The corresponding functions in the contract are called and executed based on the relevant information of the transaction. The generated results and variable status are updated and written into the blockchain.



**Figure 3.** The operation mechanism of smart contracts.

Smart contracts on the blockchain have the following advantages [8]:

- The contract code is recorded and verified by the blockchain with tamper-evident characteristics;
- Smart contracts are executed between distrustful nodes, without centralized control and without third-party coordination, ensuring fair execution of contract content;
- Traceability of information on the chain can be achieved through function calls of smart contracts.

The above advantages satisfy the control of the crowdsourcing service process and ensure the fairness and safety of the process.

### 3. Blockchain-Based Model Architecture of Crowdsourcing Design

#### 3.1. Overview

Blockchain-based model architecture of crowdsourcing design services refers to the application of blockchain technology to the management and control of the process of crowdsourcing design services. Using a blockchain-based platform instead of a traditional centralized database to store transaction data, the whole process of crowdsourcing work is open, transparent, tamper-proof and traceable, the current status of the crowdsourcing design task can be viewed at any time. In this paper, combining the advantages of blockchain to build a decentralized crowdsourcing design service system, we include all crowdsourcing functions such as user registration, posting tasks and receiving tasks. Crowdsourcing task requesters and workers can interact with the blockchain and reach agreements about crowdsourcing design tasks on the chain.

Inspired by [14], the blockchain-based crowdsourcing design service framework is divided into three layers: the application layer, the blockchain layer and the storage layer. As shown in Figure 4, users complete the crowdsourcing process of registering, posting tasks, receiving tasks at the application layer and reaching task agreements at the blockchain layer with task-status changes. Due to the limited storage capacity of the blockchain, files with large memory can seriously affect the operational efficiency of the blockchain. This framework stores the metadata of the files (such as author, hash value, file pointer, etc.) to the blockchain, while the detailed source files are stored to the storage layer to improve the efficiency of the crowdsourcing service.

In order to improve the control of crowdsourcing design services, this paper constructs a task-state machine that describes all the states of a crowdsourcing design task. There exist six states: *Pending*, *Claimed*, *Unclaimed*, *Evaluating*, *Completed*, *Uncompleted*. The task state will shift according to the user's input in the application layer or the smart contract's determination of the current task state, as shown in Figure 5, which illustrates the process and reasons for the task-state change. Whenever a task publisher posts a new task in the application layer and it is recorded on the blockchain, the system will automatically generate a task-state machine that tracks the status of the task and can be used by all users to query the current status of the task at any time.

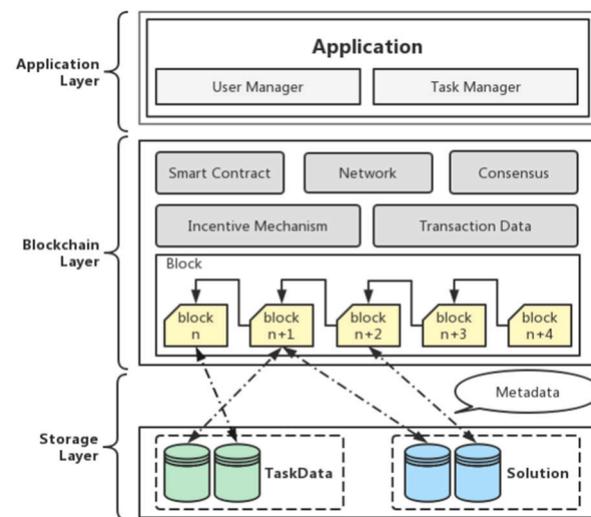


Figure 4. Blockchain-based Crowdsourcing Service Framework.

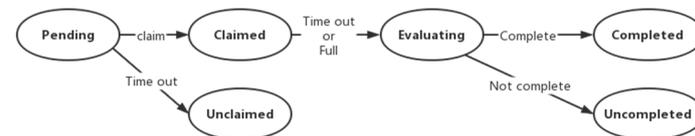


Figure 5. Task-state machine model.

### 3.2. Structure

#### 3.2.1. Application Layer

The application layer serves as a port for users to participate and complete crowdsourcing design tasks. Its operation is completely independent of any trusted third-party centralized servers, so that even the presence of some bad nodes will not affect the operation of the whole system.

In the application layer, there are two main modules: user management (UM) and task management (TM). Among them, the UM contains user registration and user information management functions. All users can only use subsequent crowdsourcing services after completing registration in the application layer. For each new user registration, the system will send the corresponding transaction information to the blockchain; after the registered transaction information is confirmed on the chain, the system will generate and store an abbreviated information structure of the user for the subsequent management of crowdsourcing services. Blockchain-based crowdsourcing design services can be seen as a program deployed on the blockchain to post, receive, submit and finally distribute rewards for design tasks.

#### 3.2.2. Blockchain Layer

The blockchain layer is located in the middle layer of the framework; its main purpose is to receive valid input from users in the application layer, store the corresponding file metadata to the blockchain and store the larger attachments to the storage layer. All transaction operations of users in the application layer need to be completed on this layer and reach consensus. The process is roughly as follows: the transaction is sent to the blockchain layer; triggering the corresponding smart contract to reach an agreement; the program is compiled and finally confirmed by the miner to be recorded on the blockchain; the miner receives the corresponding incentive. Another purpose of the blockchain layer is to run the task-state machine, where valid user’s inputs at the application layer change the task state of the blockchain layer synchronously.

Generally, blocks in blockchain layer cannot hold large file data, which will consume a lot of local disk space and affect the operational efficiency of the blockchain. To reduce the

amount of data stored in the blockchain, the source-file information is split into metadata and source data in this system framework. Source data refers to those attachments with large amounts of data that are kept in the storage layer—the distributed database under the blockchain. Following this, a unique file query string is generated. While metadata only retains basic information about the file, such as owner name, timestamp, file hash pointer, query string, etc., this information, with small amount of data, can be stored on the blockchain. Users can find the source file through the query pointer on the blockchain, they can also confirm whether the data stored in the storage layer has been tampered with using the file hash value.

### 3.2.3. Storage Layer

The storage layer, as the last layer of the framework, mainly uses distributed databases such as IPFS [15] to keep the original large files of tasks or solutions. The data uploaded by a user should be signed with his/her private key; other users can use his/her public key to verify the authenticity and integrity of the file at the blockchain layer. In particular, before workers submit their solutions to the storage layer, they need to be encrypted with the requester's public key to prevent data leakage. After the requester finds the corresponding solution file through the data pointer, he/she should first decrypt it with his/her private key, then verify the digital signature with the worker's public key to ensure the reliability and authenticity of the file. With this method, arbitrary crowdsourcing design services can be achieved, ensuring the security and confidentiality of the entire process.

### 3.3. Process of Crowdsourcing Design

**Step 1:** The user completes registration. The system records the valid information entered by the user as a transaction to the blockchain.

**Step 2:** Transaction information is confirmed by miners and permanently recorded on the blockchain

**Step 3:** The task requester posts tasks. The requester can set the task conditions; only workers who meet the requirements can receive this task. Moreover, the requester needs to pay a certain task deposit in advance as a deposit for this crowdsourcing design act.

**Step 4:** Workers receive crowdsourcing design tasks that meet their requirements. Similar to requesters, workers are required to pay a deposit for each task they receive to ensure the quality of their completed task.

**Step 5:** Worker completes the crowdsourcing design task and submits it before the task deadline. They first sign a digital signature using their private key, then encrypt it using the requester's public key, then upload the encrypted file to the storage layer, a distributed database. Finally, they store the file's hash and retrieval pointer to the blockchain.

**Step 6:** The requester should complete the task evaluation before the task evaluation deadline. The system will automatically issue rewards and reputation value to the worker according to the task evaluation level. Completing high-quality crowdsourcing design tasks will result in more rewards and increased reputation value.

## 4. Smart Contracts and Algorithm Protocols

### 4.1. Notations

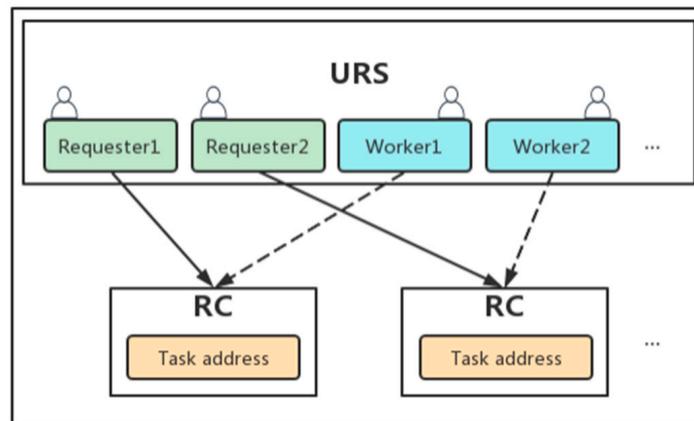
Before detailing the smart contracts in the framework proposed in this paper, the notations used in smart contracts are shown and explained in Table 1. When the corresponding symbols are used subsequently, they are further explained.

### 4.2. Smart Contract

Two types of smart contracts are implemented to complete the above crowdsourcing design service process and realize the control of the service process. They are: user register and summary contracts (URSs) and relationship contracts (RCs). The general structure of the contract is shown in Figure 6.

**Table 1.** The notations of explanation.

Natation	Explanation
$\beta_W$	The reputation value of workers
$addr_{User}, addr_{Contract}$	The user’s address and the address of smart contract
$\theta$	The reward of the crowdsourcing design task
$\delta$	The deposit of posting and receiving task
$\lambda$	The number of workers required for the task
$dl$	The task deadline, which refers to the future block height
$pointer$	The hash pointer of the attachment of task or solution
$coins(v)$	The virtual coin of value $v$



**Figure 6.** The structure of smart contracts.

After the user completes the registration, URS will record all the information of the user, such as the user’s address, name and resume information. The task information of the crowdsourcing design will be updated as the user completes the task in the system, such as the reputation value of the worker and the address of the task received. RC mainly records the interaction information between users, such as the details of the posted task and the completion status of the worker after receiving the task. The worker and the publisher reach a crowdsourcing design service agreement in RC.

The contract contains an important protocol: the reputation value update protocol *updateReputation()*, which is used to manage the user’s reputation. It automatically increases or decreases the reputation value of the corresponding worker based on the publisher’s evaluation of the task.

While most traditional crowdsourcing systems focus on finding deception or other malicious behaviors after workers complete their tasks, this crowdsourcing design system curb malicious behaviors of workers in advance and improve the quality of crowdsourcing design task completion by setting requirements such as the required minimum reputation value of task workers when requesters post their tasks.

#### 4.2.1. User Register and Summary Contract (URS)

URS contract is mainly used for user registration and managing user information. Users can complete registration without using their real identity information, providing them with a certain level of privacy protection. The main structure of the contract is shown in Figure 7, where a total set of users, *UserPool*, and a series of data structures recording the user’s details are provided. Each user’s data structure records *Username*, *Address*, *profile*, *RegisterTime*, *Reputation*, the number of ongoing crowdsourcing design tasks (*ProcessTaskNum*), the number of finished crowdsourcing design tasks (*FinishTaskNum*) and a list of addresses of completed tasks (*RC Address*). The *reputation* is an important parameter; the system will issue the default reputation value after the user completes the registration, which will be updated automatically with the user’s work performance in the

future. The incentive mechanism of reputation value is based on the literature [16], where the level of reputation value directly can reflect the user's level of crowdsourcing design. *ProcessTaskNum* reflects the user's current level of busyness. *FinishTaskNum* combined with *Reputation* reflect the user's ability to work. *RC Address* reflects the main areas of work of the user. None of the above data can be easily altered by third parties and can only be updated automatically after a crowdsourcing design task. Therefore, using the above parameters as criteria for receiving tasks ensures that the worker has sufficient time and capacity to complete the task.

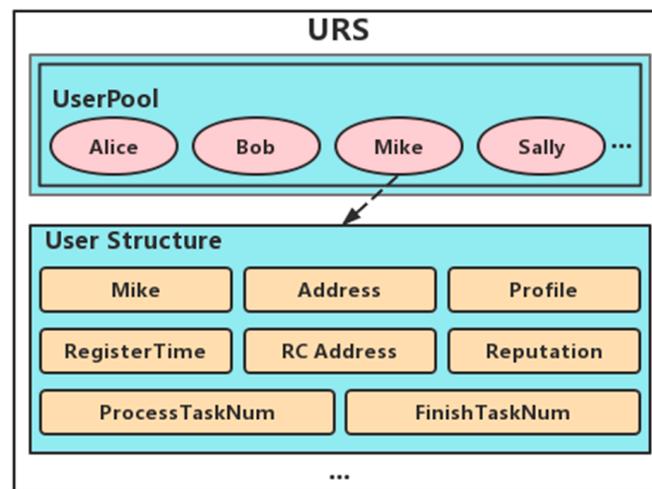


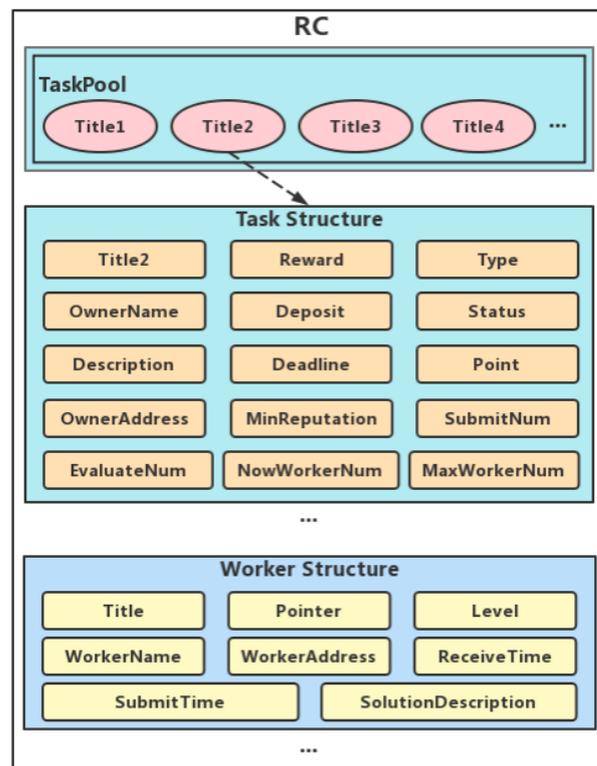
Figure 7. The structure of URS Contract.

In particular, the contract is set up in such a way that the same *Username* cannot be registered repeatedly, making each username unique. Setting the user's name as a mapping of the user's detailed data structure ensures accurate invocation of the user's information in subsequent smart contracts for crowdsourcing design services. With each new user registration, the system detects the *Username* in *UserPool* and automatically updates the mapping of the user data structure.

#### 4.2.2. Relationship Contract (RC)

RC contract records the interaction protocols between publishers and workers, covering the entire process of posting, picking up, submitting and acquiring solutions, evaluating and issuing rewards for tasks. The RC contract architecture is shown in Figure 8. The contract architecture contains a total task set, *TaskPool*; a data structure, *Task Structure*, that records task details; and a data structure, *Worker Structure*, that records worker task processes.

Through a smart contract, the task publisher posts a crowdsourcing design task and sets the corresponding requirements for the worker. The system will automatically generate the corresponding task structure and store it on the blockchain. Before a worker receives a task, the function *ReceiveTask()*, defined in the RC contract, evaluates the worker's ability to work according to the requirements set by the requester. If the task requirements are satisfied, the worker can successfully receive the task. At the same time, the system will generate a corresponding worker structure; the publisher can track the progress of the worker's crowdsourcing design task at any time. The task-status machine changes the task status from *Pending* to *Claimed*. When the number of workers receiving the task reaches the required number of workers set by the publisher, the task will not be available for receiving again. Workers who have received an assignment will need to submit a solution before the deadline.



**Figure 8.** The structure of RC contract.

If the posted task includes some essential large files or folders, requesters should also save the corresponding attachment folders to the storage layer as mentioned above, then store the corresponding generated search pointer to the blockchain. The worker uses the pointer to find the corresponding task document to access the details and background of this crowdsourcing design task. In addition, a simple task deposit agreement is set up in the smart contract. Users are required to pay a certain amount of task deposit when posting or receiving tasks to ensure a fair performance in the crowdsourcing service agreement reached by both parties.

#### 4.3. Reputation Algorithm

Each successfully registered user is assigned a default reputation  $\beta_W, \beta_W \in Z, (0, 1, \dots, \beta_W^{Max})$ , which reflects the user's ability to work and reputation level. The reputation value is updated depending on the publisher's evaluation of the worker's completion of the crowdsourcing design task, with  $a = H$  indicating a positive review and  $a = L$  indicating a negative review. The formula for calculating the reputation can be described as:

$$\beta_W = \begin{cases} \min(\beta_W^{Max}, \beta_W + 1), & \text{if } a = H \text{ and } \beta_W \geq h_k \\ \beta_W + 1, & \text{if } a = L \text{ and } \beta_W < h_k + 1 \\ \beta_W - 1, & \text{if } a = H \text{ and } \beta_W \geq h_k + 1 \\ 0, & \text{if } a = L \text{ and } \beta_W = h_k \end{cases} \quad (1)$$

where  $h_k$  is the social strategy threshold, a method of using social norms to control worker behavior [16]. If the user's reputation value drops to a threshold and the crowdsourcing design task does not receive a positive review, the user's reputation value will simply be cleared. The user will not be able to receive tasks normally and will need to complete some simple tasks with no specific requirements for reputation to accumulate positive feedback until the reputation value rises to the threshold, when the user's task receiving function will return to normal.

#### 4.4. Protocol Algorithm

To completely describe the process of crowdsourcing design services, the following algorithms are designed in the smart contract: *Register*, *PostTask*, *ReceiveTask*, *SubmitSolution*, *GetSolution*, *EvaluateSolution* and other auxiliary algorithms. The user interacts with the smart contract deployed on the blockchain through the crowdsourcing application to complete the crowdsourcing design. The flow of smart contract and protocol interaction for crowdsourcing design services is shown in Figure 9.

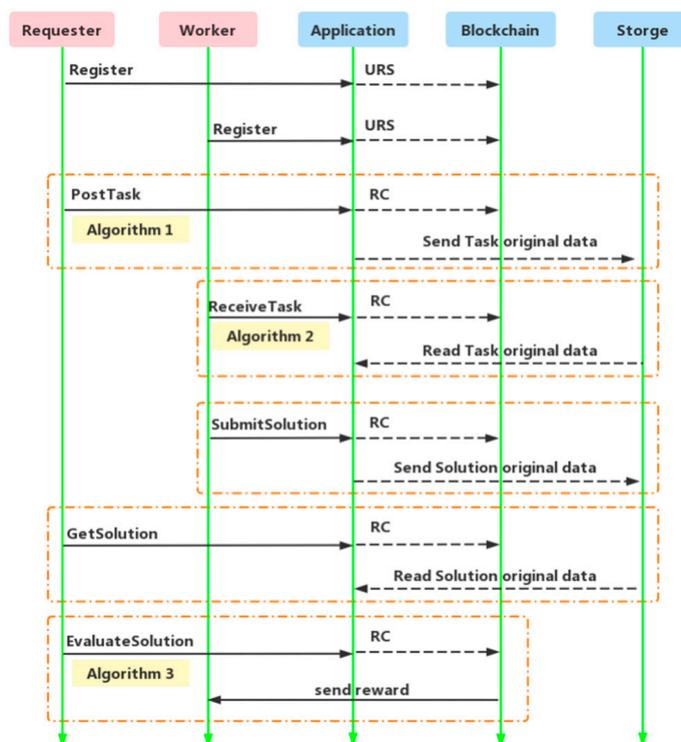


Figure 9. The flow of smart contract and protocol interaction for crowdsourcing design services.

##### 4.4.1. Register

Users need to complete identity registration before participating in a crowdsourcing design task. The *Register* protocol first confirms that the currently registered username is not in use, then publishes the user’s detailed registration information  $User = \{addr_{User}, Username, Profile, Time_{Reg}, Num_{PT}, Num_{FT}, \beta_W\}$  to the blockchain, where  $Time_{Reg}$  denotes the user registration time.  $Num_{PT}$  and  $Num_{FT}$  denote the number of tasks being processed and the number of finished tasks by workers, respectively, both with initial values of zero.  $\beta_W$  is the initial reputation value issued by the system, which is related to the average reputation value of all workers in the current crowdsourcing system.

##### 4.4.2. Posting Task

Once a task publisher has completed registering, they can post crowdsourcing design tasks  $Task = \{addr_{URS}, Requester, Title, Description, \theta, dl_S, dl_E, \lambda, \beta_W^{Min}, Type, Pointer_T\}$  in the crowdsourcing application, where  $dl_S$  and  $dl_E$  denote the deadline for task workers to submit tasks and the deadline for publishers to complete evaluations, respectively;  $Type$  is the type of the current crowdsourcing design task; and  $Pointer_T$  denotes a pointer to the necessary attachments for the current crowdsourcing design task. In order to avoid malicious behavior from requesters, such as refusing to pay commission, it is set that requesters need to pay the corresponding task deposit before posting the task. Algorithm 1 shows the concrete implementation of posting task.

**Algorithm 1:** *PostTask*


---

**Inputs:** the address of URS  $addr_{URS}$ , *Requester*, *Title*, *Description*, task reward and deposit  $coins(\theta + \delta)$ , the deadline of worker submit task  $dl_S$ , the deadline of requester evaluate task  $dl_E$ , maximum workers number  $\lambda$ , the limited condition of worker  $\beta_W^{Min}$ , *Type*, pointer of task attachment  $Pointer_T$

**Outputs:** RC contract  $RC_{Task}$ , update  $URS_{requester}$

```

01 if Requester is unregistered then
02   Requester has not been registered;
03   go final;
04 if Title is not unique then
05   Title has been taken;
06   go final;
07 if  $coins(\theta + \delta) < \theta + \delta$  then
08   Requester deposits the reward on blockchain failed;
09   go final;
10  $postTaskList() \leftarrow Title$ ;
11  $TaskPool(Title) \leftarrow Task\{addr_{URS}, Requester, Title, Description, \theta, dl_S, dl_E, \lambda, \beta_W^{Min}, Type, Pointer_T\}$ 
12  $nowWorkerNum_{task} \leftarrow 0$ ;
13  $updateURSContract(RC_{Task}, URS_{requester})$ ;
14 final;
15 returns  $RC_{Task}$ 

```

---

## 4.4.3. Receiving Task

The worker calls the *getPendingTask()* protocol in the RC contract to query the crowdsourcing design tasks available for collection and receives the task that matches his/her work capability via Algorithm 2. Similar to task requesters, workers are required to pay a certain amount of task deposit before receiving the task in order to prevent them from slacking off and working negatively.

**Algorithm 2:** *ReceiveTask*


---

**Inputs:** RC contract  $RC_{Task}$ , the address of URS  $addr_{URS}$ , *Worker*, *Title*, task reward and deposit  $coins(\delta)$ , worker  $URS_{worker}$

**Outputs:** update RC contract  $RC_{Task}$  and URS contract  $URS_{worker}$

```

01 if Worker is unregistered then
02   Worker has not been registered;
03   go final;
04 if  $\beta_W < \beta_W^{Min}$  then
05   Worker does not satisfy the condition;
06   go final;
07 if  $checkTaskStatus(Title)$  is not Pending or Claimed then
08    $TaskPool(Title)$  can be accepted anymore;
09   go final;
10 if  $coins(\delta) < \delta$  then
11   Requester deposits the reward on blockchain failed;
12   go final;
13  $workerList() \leftarrow worker(addr_{worker}, W_{username}, Title, Timestamp)$ 
14  $Num_{PT} ++$ ;
15  $receivedTaskList() \leftarrow Title$ ;
16  $nowWorkerNum_{task} ++$ ;
17  $updateTaskStatus(Title)$ ;
18  $updateURSContract(RC_{Task}, URS_{worker})$ ;
19 final;
20 return  $RC_{Task}$ 

```

---

#### 4.4.4. Submitting and Acquiring Task Solution

A worker completes a task and submits the task solution via the *submitSolution()* protocol, which is set to be invoked by workers who have received the task to complete the solution submission only before the task's submission deadline. First, the worker signs with the private key and the public key of the task requester to encrypt the content of the solution and store it in the storage layer—the distributed database. Following this, the corresponding file hash value and the file retrieval pointer are submitted and stored in the blockchain. Finally, the requester gets the file pointer through the *getSolution()* protocol to find the source file and decrypts it with his/her private key to access the solution content.

#### 4.4.5. Evaluating Task Solution and Sending Reward

After a worker submits a task, the requester needs to start the process of evaluating the task and paying the task reward before the evaluation deadline. The task reward is determined by the quality of the worker's completed task via Algorithm 3; based on the evaluation result, the URS contract automatically updates the worker's reputation value synchronously.

---

#### Algorithm 3: Evaluate Solution

---

**Inputs:** RC contract  $RC_{Task}$ , *Requester*, *Title*, *Worker*, task evaluate *level*, social strategy  $h_k$   
**Outputs:** update RC contract  $RC_{Task}$ , update URS contract  $URS_{requester}$  and  $URS_{worker}$ , send reward to the worker  $W$

```

01 if checkTaskOwner(Requester) is failed then
02      $R_{username}$  is not the owner of this task;
03     go final;
04 if checkTaskLevel is success then
05     task has been evaluated;
06     go final;
07 if checkTaskStatus(Title) is Uncompleted then
08      $\beta_W \leftarrow 0$ ;
09     sendReward(coins(0));
10 else if  $\beta_W \geq h_k$  & level  $\equiv H$  then
11      $\beta_W \leftarrow \min(\beta_W^{Max}, \beta_W + 1)$ ;
12     sendReward(coins( $\theta + \delta$ ));
13 else if  $\beta_W \geq h_k$  & level  $\equiv L$  then
14      $\beta_W \leftarrow \beta_W - 1$ ;
15     sendReward(coins( $\delta$ ));
16 else if  $\beta_W \equiv h_k$  & level  $\equiv L$  then
17      $\beta_W \leftarrow 0$ ;
18     sendReward(coins( $\delta$ ));
19 else if  $\beta_W < h_k$  & level  $\equiv H$  then
20      $\beta_W \leftarrow \beta_W + 1$ ;
21     sendReward(coins( $\delta$ ));
22 updateReputation( $URS_{worker}$ ,  $\beta_W$ );
23 updateURSContract( $RC_{Task}$ ,  $URS_{worker}$ );
24 updateURSContract( $RC_{Task}$ ,  $URS_{Requester}$ );
25 updateAvgReputation();
26 final;
27 returns  $RC_{Task}$ ,  $URS_{requester}$ ,  $URS_{worker}$ 

```

---

## 5. Experiment

### 5.1. Experiment Environment

This paper is designed to realize blockchain-based process management and control of crowdsourcing design services based on the above framework. The smart contract framework is written and tested using Solidity v0.8.7 programming language and Remix web editor. The experimental environment is Win10 x64 bit operating system, using the current popular blockchain development framework Truffle v5.6.4, then combined with

JavaScript and Ganache v2.6.0-beta.3 client to complete the smart contract compilation, deployment and testing.

## 5.2. Experiment Process and Analysis

To evaluate and validate the feasibility of the framework presented in this paper, we have deployed this crowdsourcing design platform on our regional blockchain, validating the functionality of the smart contract in crowdsourcing design service steps, including user registration, the requester posting a crowdsourcing design task, the worker receiving the task, the worker submitting the task and the user traceability of the crowdsourcing design task.

### 5.2.1. Smart Contracts Compilation and Deployment

The compilation and deployment of the smart contract is completed using commands in the operating system terminal, as shown in Figure 10. The ABI of the smart contract will be generated after compilation, which is used to call the functions in the smart contract and implement the data storage. This is successfully deployed to the Ganache client, as shown in Figure 11.

```
> Artifacts written to D:\BC\build\contracts
> Compiled successfully using:
  - solc: 0.8.17+commit.8df45f5f.Emscripten.clang

Starting migrations...
=====
> Network name:   'ganache'
> Network id:    1337
> Block gas limit: 6721975 (0x6691b7)
```

Figure 10. Successfully compile and deploy smart contracts.

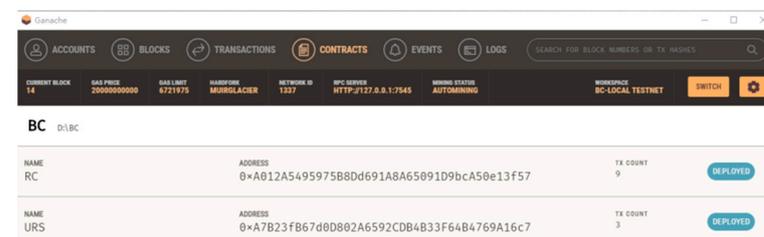


Figure 11. Deploying contracts to Ganache client.

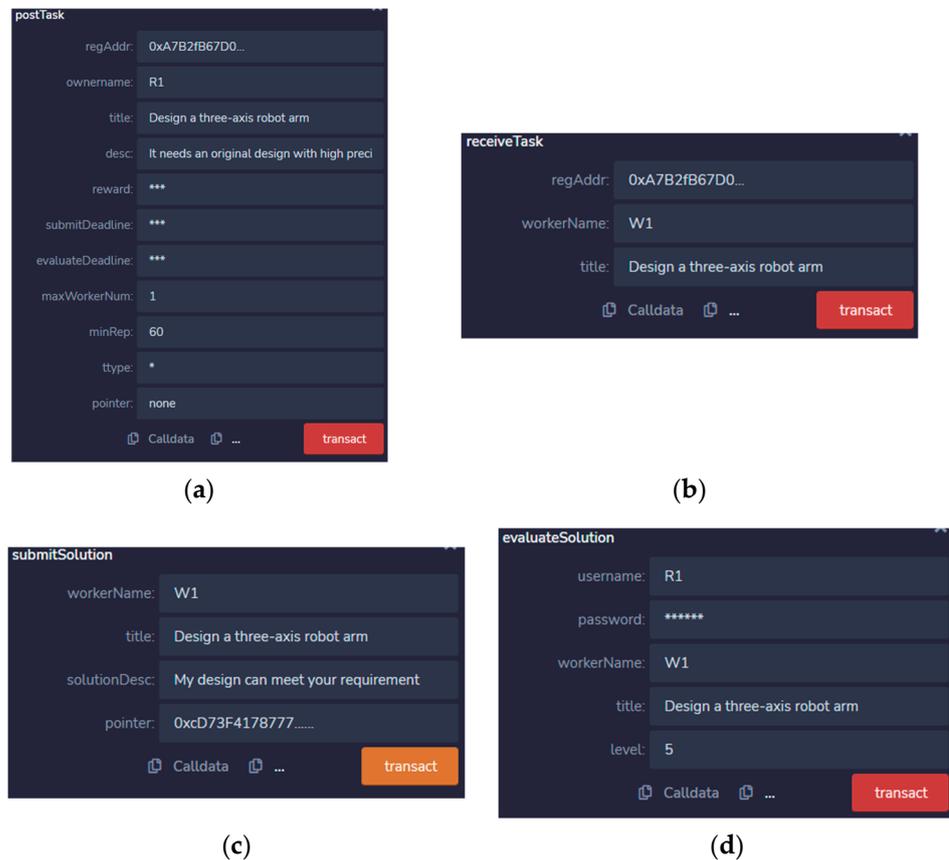
### 5.2.2. System Functionality Testing

The following is a test to verify some basic functions of this crowdsourcing design service process control. Figure 12 shows the user registration data recorded on the blockchain and *addrList[]* stores the address information of the users; currently three users have completed registration.

```
STORAGE
{
  0 items
  owner : address "0x49b7fcD87ee14457628E752735Ce418F01F24902"
  addrList : [ 3 items
    0 : string "0xF9241ad32C5E44ea9f..."
    1 : string "0x976b9Ea76a21ce8c31..."
    2 : string "0x5a964485eff6cc2322..."
  ]
  maxRegistrants : uint 64
  numRegistrants : uint 3
  sumRep : uint 115
  userPool : {} mapping 0 items
}
```

Figure 12. User registration data on the blockchain.

As shown in Figure 13, the requester *R1* posts a task titled “Design a three-axis robot arm” on this crowdsourcing design platform, and the worker *W1* receives and submits the task. One must simply follow the prompts to enter the corresponding crowdsourcing design task information and then initiate a transaction to complete the interaction with the blockchain and accomplish the crowdsourcing design service.



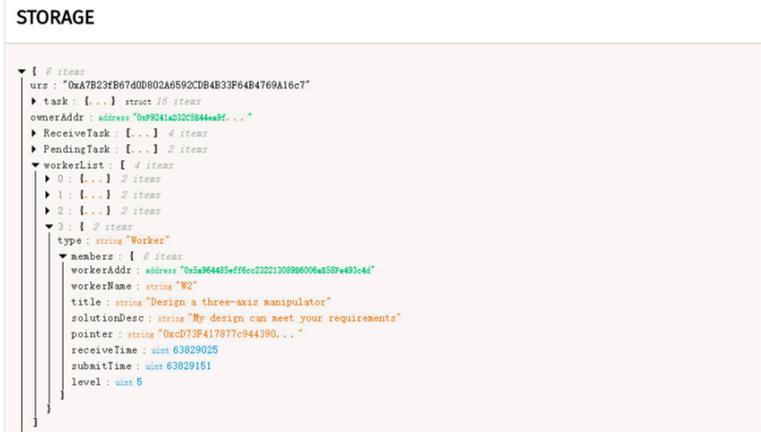
**Figure 13.** Users complete the process of crowdsourcing design services. (a) Requester *R1* posts the task; (b) Worker *W1* receives the task; (c) Worker *W1* submits the solution; (d) Requester *R1* evaluates the solution.

After the above simulation of the crowdsourcing design task, the details of this crowdsourcing design task are recorded on the blockchain, as shown in Figure 14: “Requester *R1* has posted a crowdsourcing design task, ‘Design a three-axis robot arm’. The design requires ‘It needs an original design with high precision, low cost, simple transmission, simple manufacturing process’. Currently one worker received this task and completed the submission; the status of the task is completed”.



**Figure 14.** Crowdsourcing design task information.

Figure 15 shows the traceability information of worker *W1*'s completion of the crowdsourcing design task recorded on the chain. The traceability information details the address information of the worker, the time of receiving the task and the time of submitting the task, a brief description of the task and the attached retrieval pointer.



```

STORAGE
{
  @ items
  url : "0xA7B234B67d0D802A6592CDB4B3F64B4769A16c7"
  task : [...] struct 18 items
  ownerAddr : address "0x9941a032c5644a9f..."
  ReceiveTask : [...] 4 items
  PendingTask : [...] 2 items
  workerList : [ 4 items
  0 : [...] 2 items
  1 : [...] 2 items
  2 : [...] 2 items
  3 : [ 2 items
  type : string "Worker"
  members : [ 2 items
  workerAddr : address "0x5a96485eff6c222130896006a58f493c4d"
  workerName : string "W2"
  title : string "Design a three-axis manipulator"
  solutionDesc : string "My design can meet your requirements"
  pointer : string "0xc073f417877c944390..."
  receiveTime : uint 63829025
  submitTime : uint 63829151
  level : uint 5
  ]
  ]
}

```

Figure 15. The traceability information of crowdsourcing design task.

In summary, the test results show that the blockchain-based control of crowdsourcing design services and visual management of crowdsourcing design tasks are achieved. From the posting of a crowdsourcing design task to the final payment of the task reward, the complete details of the user's operation are recorded on the blockchain and updated as the user interacts with the smart contract. In particular, the data of crowdsourcing design tasks cannot be tampered with or deleted at will. Crowdsourcing design tasks performed under this framework can always track the current status of the task; even after the crowdsourcing design task is finished, the task data will be permanently saved and can be traced back to any task at any time. The entire system consensus is jointly maintained by the participating nodes, which has higher credibility than relying on third parties for data management. The blockchain system and IPFS are both distributed architecture, which does not have the problem of single point of failure and has higher availability.

## 6. Conclusions

In this paper, we analyze some problems of the traditional crowdsourcing design service platform in the context of social manufacturing, design a blockchain-based approach of model architecture for crowdsourcing design services using the decentralization, tamper-proof and traceability features of blockchain. Combined with IPFS and other technologies, the whole crowdsourcing logic process is implemented with smart contracts. The paper details the composition of the application layer, blockchain layer and storage layer in this method; depicts the data-interaction process between three layers; and solves the problem of blockchain-based crowdsourcing design services involving large files that affect the operation of the blockchain. Finally, we tested the deployment of the entire framework on the blockchain. The results showed that the entire crowdsourcing design workflow can be fully implemented and the control of the crowdsourcing design service process can be achieved.

In this paper, a series of algorithmic protocols are designed based on smart contracts. A task-state machine is designed to accurately describe all the states of the tasks in the crowdsourced design workflow and enhance the flexibility of the crowdsourced design service, combining blockchain technology with a crowdsourcing design model under the context of social manufacturing.

Although the model and method proposed in this paper are feasible and practical, there are still some shortcomings. The experimental validation in Section 5 simply verifies the usability of the model functions with a small scale of experimental data. The presented

model performs well when the number of network nodes is low, but as the number of network nodes increases, the uncertainty brought to the model remains to be studied in depth. In the future, we will further improve the algorithmic logic of smart contracts so that they can run stably and securely in a complex node network environment before deploying them to public test networks for testing.

**Author Contributions:** Conceptualization, methodology, supervision, project administration and funding acquisition, D.L. (Dianting Liu); Software, formal analysis, writing—original draft preparation, writing—review and editing, and visualization, D.L. (Dong Liang). All authors have read and agreed to the published version of the manuscript.

**Funding:** This research was funded by the National Natural Science Foundation of China, grant number 71961005, and Natural Science Foundation of Guangxi Zhuang Autonomous Region, grant number 2020GXNSFAA297024.

**Data Availability Statement:** The smart contract used during the current study are available from the corresponding author Dong Liang (1020211081@glut.edu.cn) upon reasonable request.

**Conflicts of Interest:** The authors declare no conflict of interest. The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript; or in the decision to publish the results.

## References

1. Guo, W.; Wang, Z.; Shao, H.Y.; Wang, L.; Gong, L.; Yu, S.S.; Feng, Y.X.; Wan, X.M.; Liu, J.Q. Crowdsourcing design theory and key technology development. *Comput. Integr. Manuf. Syst.* **2022**, *28*, 2650–2665.
2. Jiang, P.Y.; Yang, M.L.; Li, W.D.; Liu, J.J.; Guo, W.; Li, P.L. CI Literature Review and Its Application Exploration in Social Manufacturing. *China Mech. Eng.* **2020**, *31*, 1852–1865.
3. Jiang, P.Y.; Leng, J.W.; Ding, K. Analyzing and delimiting overlapping concept boundaries of social manufacturing. *Comput. Integr. Manuf. Syst.* **2018**, *24*, 829–837.
4. Nakamoto, S. Bitcoin: A Peer-to-Peer Electronic Cash System. Available online: <http://www.bitcoin.org/bitcoin.pdf> (accessed on 18 September 2022).
5. Gavin, W. Ethereum: A Secure Decentralised Generalised Transaction Ledger. *Ethereum Proj. Yellow Pap.* **2014**, *151*, 1–32.
6. Szabo, N. Smart Contracts: Building Blocks for Digital Markets. Available online: <https://www.docslib.org/doc/246577/smart-contract-building-blocks-for-digital-markets> (accessed on 10 November 2022).
7. Howe, J. The rise of crowdsourcing. *Wired Mag.* **2006**, *14*, 1–4.
8. Li, Y.; Duan, H.Y.; Yin, Y.Y.; Gao, H.H. Survey of Crowdsourcing Applications in Blockchain Systems. *Comput. Sci.* **2021**, *48*, 12–27.
9. Ding, Y.; Chen, Z.; Lin, F.; Tang, C. Blockchain-Based Credit and Arbitration Mechanisms in Crowdsourcing. In Proceedings of the 2019 3rd International Symposium on Autonomous Systems (ISAS), Shanghai, China, 29–31 May 2019; pp. 490–495. [CrossRef]
10. Wu, C.; Zhao, M.N.; Hua, X.H.; Qi, H.W. Design and Implementation of Crowdsourcing Agricultural Products Logistics Platform Based on Block Chain Technology. *J. Geomat.* **2022**, *2*, 139–143.
11. Yang, Z.; Huang, S.; Zheng, C.Y.; Wang, T.Y. A Trusted Management Framework for Crowdsourced Testing Intellectual Property Based on Blockchain. *Comput. Appl. Softw.* **2021**, *10*, 27–32+99.
12. Ghaffaripour, S.; Miri, A. A Decentralized, Privacy-Preserving and Crowdsourcing-Based Approach to Medical Research. In Proceedings of the 2020 IEEE International Conference on Systems, Man, and Cybernetics (SMC), Toronto, ON, Canada, 11–14 October 2020; pp. 4510–4515.
13. Gao, L.P.; Cheng, T.; Gao, L. TSWCrowd: A Decentralized Task-Select-Worker Framework on Blockchain for Spatial Crowdsourcing. *IEEE Access* **2020**, *8*, 220682–220691. [CrossRef]
14. Li, M.; Weng, J.; Yang, A.J.; Lu, W.; Zhang, Y.; Hou, L.; Liu, J.N.; Xiang, Y.; Deng, R.H. CrowdBC: A Blockchain-based Decentralized Framework for Crowdsourcing. *IEEE Trans. Parallel Distrib. Syst.* **2019**, *20*, 1251–1266. [CrossRef]
15. Benet, J. IpfS-content addressed, versioned, p2p file system. *arXiv* **2014**, arXiv:1407.3561.
16. Zhang, Y.; van der Schaar, M. Reputation-Based Incentive Protocols in Crowdsourcing Applications. In Proceedings of the 2012 Proceedings IEEE INFOCOM, Orlando, FL, USA, 25–30 March 2012; pp. 2140–2148.

**Disclaimer/Publisher’s Note:** The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.