Secure and Distributed Crowd-Sourcing Task Coordination Using the Blockchain Mechanism

Safina Showkat Ara(✉), Subhasis Thakur, and John G. Breslin

Insight Centre for Data Analytics, NUI Galway, Galway, Ireland
{safina.ara,subhasis.thakur,john.breslin}@insight-centre.org

Abstract. A complex crowd-sourcing problem such as open source software development has multiple sub-tasks, dependencies among the sub-tasks and requires multiple workers working on these sub-tasks to coordinate their work. Current solutions of this problem employ a centralized coordinator. Such a coordinator decides on the sub-task execution sequence and as a centralized coordinator faces problems related to cost, fairness and security. In this paper, we present a futuristic model of crowd-sourcing for complex tasks to mitigate the above problems. We replace the centralized coordinator by a blockchain and automate the decision-making process of the coordinator. We show that the proposed solution is secure, efficient and the computational overhead due to employing a blockchain is low.

Keywords: Crowd-sourcing · Task coordination · Blockchain · Trust

1 Introduction

In this paper, we study complex Crowd-Sourcing (CS) task which has multiple sub-tasks, constraints among these sub-tasks. It will require multiple workers to coordinate their effort to solve such a complex task. Coordination among the workers is recognized as an important aspect for future CS platforms [6,16]. For example in open source software development [11], a developer works on a specific part of the software and it must coordinate with other developers working on other parts of the software. Another example of such CS task may be a multi-player version of the protein folding game [3]. The challenges with executing such complex CS tasks are as follows:

Cost of Coordination: We may assign the CS sub-tasks to the workers and the workers must coordinate their efforts to comply with the constraints among the sub-tasks. Coordinated execution of these sub-tasks is a sequence of sub-task executions where a worker must consider previous solutions while solving its task. The most efficient coordinator would minimize the length of such a sub-task execution sequence. Apart from deciding on the sub-task execution sequence, the coordinator must also evaluate compatibility among solutions of the sub-tasks.

R. Montella et al. (Eds.): IDCS 2019, LNCS 11874, pp. 402–413, 2019.
https://doi.org/10.1007/978-3-030-34914-1_38

It may hire additional workers to perform such evaluations and hence the cost to execute the task will increase.

Fairness: Workers may need to execute the sub-tasks multiple times to comply with constraints among the sub-tasks. We need a method to ensure fairness in this process, i.e., the coordinator must follow certain rules to ensure fairness as it asks a certain worker to solve its sub-task again.

Trust on the Coordinator: Either the CS platform or the workers may act as the coordinator. The problem with the first option is that it is a centralized solution and the workers must trust the centralized coordinator for correct evaluations of coordination decisions. The problem with the second option is that a worker may be malicious and may manipulate the coordination decisions. Note that a worker may need to execute its task multiple times to comply with constraints among the sub-tasks. If the solutions of two workers are in conflict then one of them should backtrack and solve its task again. If the coordinator is malicious then it may favor certain workers and they may not execute their task again. Such activities will reduce the cost of task execution of malicious workers but it will reduce the quality of the overall solution.

The existing solutions [12,14,16] for CS task coordination do not mitigate these challenges. In this paper, we propose a BlockChain (BC) based distributed coordinator for CS where any worker may act as the coordinator and the BC mechanism ensures that such a worker remains honest while acting as the coordinator. Hence the proposed solution erases the requirement that the workers must evaluate their trust on the coordinator. Using the BC mechanism, we may securely store transaction records among peers of a peer to peer network. We use the BC mechanism to develop the CS platform as follows:

- Every worker acts as a miner and each worker keeps the entire record of the solutions produced by the workers.
- We propose a distributed coordination mechanism where each worker may act as a coordinator. The BC mechanism ensures that a worker behaves honestly while acting as a coordinator. Hence, this BC-based distributed coordination mechanism erases the requirement that a worker must evaluate its trust on the coordinator.
- We keep solutions in a BC, it becomes impossible to overwrite the records of the solutions.

In this paper we present the following results:

Security: We present a BC maintained CS platform where CS data (such as task, worker selection, solutions) is securely stored and it is infeasible to overwrite it. Also, blockchain ensures that the workers behave honestly while acting as the coordinator.

Fairness: We develop rules to ensure fairness in sub-task re-execution. These rules are part of the blockchain's data structure and consensus protocol to ensure enforcement of such rules.

Convergence: We show that the distributed coordination mechanism converges quickly.

Computational Overload: We show that the proposed coordination mechanism has negligible computational overload due to participating in a blockchain.

Efficiency: We show that the proposed distributed coordination mechanism is efficient. An efficient coordinator minimizes the number of times each worker executes its subtask.

The paper is organized as follows: In Sect. 2 we describe the task coordination problem. In Sect. 3 we present a brief description of the BC mechanism. In Sect. 4 we present the BC-based distributed coordination mechanism. In Sect. 5 we present an experimental evaluation of the proposed coordination mechanism. In Sect. 6 we mention relevant literature and we conclude the paper in Sect. 7.

2 Problem Statement

In this section, we describe a CS task which requires coordination among multiple workers. There are n workers $W = (w_1, \ldots, w_n)$. A task $T = (t_1, \ldots, t_n)$ has n subtasks. Each subtask is assigned to one worker. $W(t_i) \in W$ indicates the worker of the subtask t_i. subtasks require coordination among workers to satisfy constraints among them. There are k constraints $\theta = (\theta_1, \ldots, \theta_k)$. A constraint θ_i requires coordination of workers corresponding to tasks $\theta_i(T) \subset T$. In centralized coordination, workers will report solutions of their respective subtasks to the coordinator who decides the execution of next set of subtasks, i.e., based on the present solutions who should again execute their tasks to satisfy constraints among the subtasks.

We represent the task, subtask allocation to workers and constraints among subtasks using a task graph. A task graph is an undirected graph $G = (W, E)$ whose vertices are the workers and there is an edge $(w_i, w_j) \in E$ if there is a constraint θ_x such that $t_a, t_b \in \theta_x(T)$ and $W(t_a) = w_i$ and $W(t_b) = w_j$. For example consider a sudoku puzzle, cells with label 0 are empty cells and we have to assign a value between 1 to 9 to the empty cells. A subtask is to assign a value to an empty cell. A worker w_i (with subtask t_i) is neighbour of another worker w_j(with subtask t_j) if t_i and t_j are on the same row (or column) of the sudoku puzzle.

In this paper, we will use sudoku puzzle as a complex task to be solved by multiple workers in a CS platform. The challenges for a distributed coordinator are as follows:

- **Secure records:** As the workers act as coordinator they can access the data on task execution by the workers. They can modify this data. Hence we need a security mechanism for safe storage and access to these data.
- **Task generation:** A worker executes its task with the information about solutions to subtasks solved by its neighbours to produce a solution which does not violate any constraints or maximally complies with the constraints. It incurs a certain cost every time it executes its subtask. We need a mechanism

that can correctly identify the worker who should adjust its solution, i.e., execute its subtask again based on the solutions produced by other workers. It may happen that a malicious worker would deny to execute its task again and ask its neighbours to adjust their respective solutions.

3 The Blockchain Mechanism (BC)

BC allows peers of a peer to peer network to transfer tokens among them using transactions. We will provide a detailed description of the transaction data structure. If a peer P_1 wants to send x tokens to P_2 then it creates the transaction T_1 and announces it to its neighbours in the BC peer to peer network. Once such a neighbour P_3 receives the transaction T_1, P_3 attempts to verify it. If it can verify T_1 as a valid transaction it forwards T_1 to its neighbours. BC mechanism stores consistent replicas of transaction history among the peers of a peer to peer network on multiple peers. Valid transactions are grouped into a block and blocks are stored as BC where each block has only one parent block. A new block can be added to the BC as the child of the most recent block. Any peer can verify transactions and add a new block to the BC provided it satisfies the conditions of the distributed consensus protocol. Distributed consensus protocol ensures that all peers have the same replica of the BC, i.e., they have the same history of transactions.

4 BC Based Task Coordination

We use blockchain as a coordinator for complex CS task is as follows:

- Workers form the blockchain peer to peer network. Two workers are neighbours in this network if their corresponding sub-tasks have at least one constraint.
- Each worker solves its sub-task and the solution is converted as a blockchain transaction by attaching the solution file (i.e., a textfile, a media file, etc.) to a blockchain transaction.
- After solving the sub-task, the worker announces its solution by creating the above transaction whose recipient is one of its neighbours in the blockchain peer to peer network. For example, as shown in Fig. 1, worker w_1 and w_2 send the solutions of their sub-tasks t_1 and t_2 to the worker w_3 as transactions τ_1 and τ_2.
- A worker (a peer) regularly compares the solutions it has received from its neighbours. The worker follows a set of rules to evaluate the solutions.
- The solutions a worker has to compare and evaluate is represented by its unspent transactions. For example in Fig. 1, w_3 has to compare solutions in τ_1 and τ_2. For each such unspent transaction, it first checks if there is any other unspent transaction such that there are constraints among the corresponding sub-tasks. Thus w_3 checks if the sub-tasks whose solutions are mentioned in τ_1 and τ_2 have any constraints. In this example, we assume that there are such constraints $(\theta_1, \theta_2, \theta_3)$.

- After the evaluation of solutions mentioned in its unspent transactions, each worker performs the following steps:
  - Let w_3 evaluated that solution mentioned in τ_1 is valid compared with the solution mentioned in τ_2.
  - w_3 will create two new transactions. In the transaction τ_1' it will copy the content of τ_1 and send it to a neighbour w_4. In the transaction τ_2' it will copy the content (the solution to the sub-task) of τ_2 and send it to w_2 who is the creator of the content of τ_2.
- If a worker receives a transaction such that it had created the content then it must solve its sub-task again.
- The above-mentioned procedure continues until a time limit as solutions to each sub-tasks are evaluated against other sub-tasks and invalid sub-tasks are solved again.

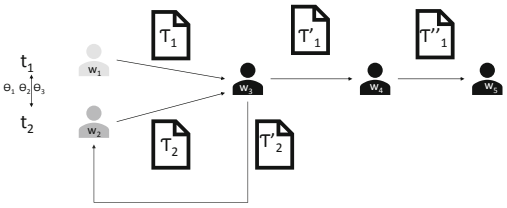


Fig. 1. Sub-tasks t_1 and t_2 with a set of constraints among them are assigned to w_1 and w_2. They solve t_1 and t_2 and create τ_1 and τ_2 with w_3 as the recipient. w_3 evaluate these solutions and decides that the solution of t_1 is valid. It sends the solution to t_1 to another neighbour for further evaluation and sends the solution of t_2 back to its creator w_2. w_2 solves its sub-task again.

In the above approach we observe the following:

- A solution of a sub-task is evaluated against solutions to other sub-tasks (with which it has certain constraints). Thus the number of times a solution to a sub-task is compared with solutions for related sub-tasks is an indicator of its validity compared with valid solutions for all sub-tasks.
- We need additional data fields in the transaction data structure to indicate who and when the solution to a sub-task was found.

Now we present a detailed description of the above mentioned distributed coordination mechanism.

4.1 Peer to Peer Network

We construct a BC peer to peer network from a task $T = (t_1, \ldots, t_n)$, $\theta = (\theta_1, \ldots, \theta_k)$ constraints over the subtasks and a set of workers $W = (w_1, \ldots, w_n)$. We assign task t_i to worker w_i. The peer to peer network is an undirected graph $G = (W, E)$ where E is the set of edges. Two workers w_i and w_j are neighbours if there is a constraint θ_x such that $t_i, t_j \in \theta_x$.

4.2 Transactions

We augment the transaction data structure as follows:

Content: A transaction contains a file describing the solution to a sub-task. This file may be a text file or a media file but must be in a predefined file format. It replaces the 'amount' information of a BC transaction.

Checksum: It will contain checksum value of the file to ensure the integrity of the file and hence the solution.

Origin: It will contain the identity of the peer who created the content file. It will store such peer's public key.

Origin Time: It will record the time when the content file was created.

Trail Number: It will record the number of times the solution corresponding to a transaction is compared with solutions corresponding other transactions and this solution was verified as a valid solution (does not violate any constraint) at every instance of such comparison.

UTXO: Unspent transaction output (UTXO) ensures that only unspent transactions are used as input to new transactions. We use the following procedure to enforce the UTXO requirement.

- Transaction for solution generation: A worker w_i should create a new solution for its subtask if either of the following holds:
  (1) It has received a transaction τ_x whose origin is w_i and trail number is 0. τ_x indicates that the previous solution of w_i for its subtask is rejected by other workers and hence it must execute its subtask again. w_i will execute its task again and such a solution will be included in a new transaction τ_y whose Content is the new solution, 'checksum' is the checksum of the new solution and it uses τ_x as the input to τ_y.
  (2) Each worker is endowed with an empty transaction τ^0 and it is used as the input to the first transaction that the worker creates whose content is its solution to its subtask.
- Transaction forwarding: A worker w_i forwards its unspent transactions τ whose *Origin* is not w_i by following this procedure:
  (1) For each transaction $\tau_x \in \tau$, if there is no other transaction in τ with which τ_x shares at least one constraint then w_i forwards τ_x to a neighbouring worker by creating a new transaction $\tau_{x'}$ whose input is τ_x. Content, Origin, Origin time and trail number of $\tau_{x'}$ remains same as τ_x. Note that trail number indicates the number times a transaction (hence a solution to a subtask) is evaluated as a valid transaction against other relevant transactions (corresponding subtasks share constraints). As there are no other relevant transactions we do not increase the trail number.
  (2) For each transaction $\tau_x \in \tau$, if there are other transaction in $\tau' \subseteq \tau$ with which τ_x shares at least one constraint then (a) if trail number of τ_x is more than any other transaction in τ' and Origin time of τ_x is less than

the same of any other transaction $\tau_y \in \tau'$ whose trail number of τ_y is same as τ_x then w_i forwards τ_x to a neighbouring worker by creating a new transaction $\tau_{x'}$ whose input is τ_x. Content, Origin and Origin time of $\tau_{x'}$ remain the same as τ_x. But we increase trail number of $\tau_{x'}$ as 1 more than the same for τ_x. (b) If τ_x does not satisfy the previous criteria then w_i forwards τ_x to $Origin(\tau_x)$ by creating a new transaction $\tau_{x'}$ whose input is τ_x and content, origin remains same as τ_x. But $\tau_{x'}$'s trail number becomes 0. Hence if a new solution is in conflict with an old solution then the new solution is regarded as a valid solution if its trail number is same as the trail number of the old solution.

Figure 2 shows an example of the above transaction data structure. w_1 and w_2 solved their respective sub-tasks (with a constraint among them) and sent the solution as transactions τ_1 and τ_2 to w_3. *Origin* field of τ_1 is marked with the public key of w_1 (we just use w_1) and its trail number is 1. w_2 evaluates that solution in τ_1 is valid compared with the solution in τ_2. Hence it forwards the solution in τ_1 to w_4 for further evaluation by creating a new transaction τ'_1 with *origin* as w_1 and trail number as 1 more than the trail number of τ_1. It also rejects the solution mentioned in τ_2 as it creates a new transaction τ'_2 with trail number 0 and *Origin* as w_2. Hence the solution is τ_1 is further evaluated with solutions from other sub-tasks and w_2 generates a new solution for its sub-task.

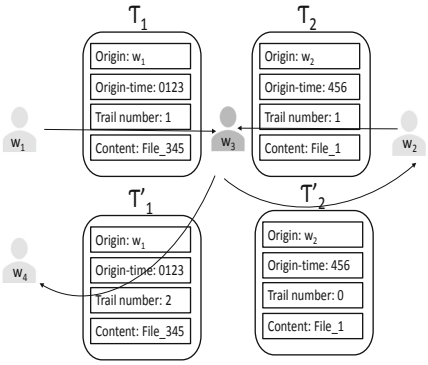


Fig. 2. It shows how data fields of transaction are changed. For a valid transaction trail number will be increased by 1 and for an invalid transaction it will be zero.

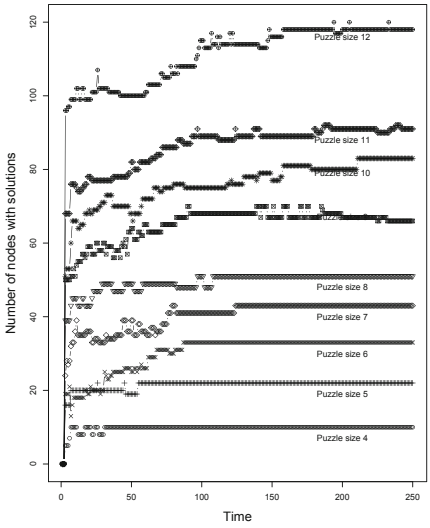


Fig. 3. Convergence time for distributed coordination mechanism.

We will use proof of work as the distributed consensus protocol. We summarize the fairness rules for sub-task re-execution such as Any worker must re-execute its sub-task if it is not maximally compatible with other related sub-tasks. If two related sub-tasks are in conflict then the sub-task which is least tested for compatibility with solutions for related sub-tasks will be executed again. If two related sub-tasks are in conflict and the number of times these sub-tasks are tested for compatibility with solutions for related sub-tasks are equal then, the sub-task which was solved earlier will be executed again.

5 Experimental Evaluation

In this section present an experimental evaluation of the proposed BC based distributed coordination mechanism. We simulate a BC using agent based modelling and asynchronous event simulation in Python. Peers (workers for a CS task) are modelled as autonomous agents. There are two types of workers (a) good quality workers and (b) bad quality workers. A good quality worker quickly finishes its subtask and a bad quality worker takes more time. In the following experiments, a bad quality worker, takes 5 times more time to finish its subtask compared with any good quality worker. We implement each worker's workflow as two processes. The first process is concerned with solving its subtask and the second process is concerned with forwarding transactions according to the transaction forwarding rules described in Sect. 4. Execution time for the first process depends on the worker's quality. We simulate these processes using SIMPY package of Python which simulates asynchronous events. For a bad quality worker we suspend its subtask solving process for next 5 time instances and the same is suspended for 1 time instant for good quality workers. q_i will denote the 'Timeout' (the time duration for which this process will sleep) amount for this process. We assume that verifying whether a solution for a subtask is correct or not is an easy problem and hence we do not impose any 'Timeout' for transaction forwarding process. There is no standard benchmark dataset for task coordination in CS [12,14]. We propose to use sudoku puzzles as benchmark dataset. Each empty cell of a sudoku puzzle is a subtask and one subtask is assigned to one worker. We use puzzles of size 4 to 14. By increasing the puzzle size we can increase the number of subtasks, the complexity of constraints and number of constraints.

Algorithm 1 describes the workflow for each worker. It executes two processes. The first process (line 5 to line 9) solves the subtask and the second process (line 10 to 27) forwards transactions. The first process checks if the worker w_i received a transaction with origin w_i and trail number 0. Such a transaction indicates the solution proposed by w_i is discarded by other workers as it is in conflict with solutions for other subtasks. If there is such a transaction then w_i finds a new solution for its subtask and creates a new transaction. The second process checks if it has received any transaction whose origin is not w_i. For each of such transaction τ_x, it checks if trail number and origin time of τ_x is more and at most the same of any other transaction $\tau_{x'}$. If τ_x satisfies these conditions then it forwards τ_x to a neighbour by increasing the trail number. Additionally, it

Algorithm 1. Simulation

Data: A task $T = (t_1, \ldots, t_n)$, Task graph $G = (W, E)$ and set of constraints $\theta = \{\theta_i \in 2^T\}$
Result: Solution to T

```
1  begin
2      for Every iteration do
3          for Every worker w_i do
4              τ ← unspent transactions of w_i
5              if τ_x ∈ τ : Origin(τ_x) is w_i and Trail(τ_x) = 0 then
6                  Solve subtask t_i while complying with {θ_x} and other solutions
                   with trail number 0 (subtask is the CS subtask for example
                   developing part of a software)
7                  Create new transaction τ' with the new solution
8                  Send the new transaction to a random neighbour
9                  Sleep(q_i)
10             τ ← unspent transactions of W_i whose Origin is not w_i
11             if τ ≠ ∅ then
12                 for Each transaction τ_x ∈ τ do
13                     τ' = τ\τ_x
14                     τ'' ⊆ τ' such that for any τ_y ∈ τ'' there is a constraint θ_i
                       where τ_y, τ_x ∈ θ_i
15                     if τ'' = ∅ then
16                         Create transaction τ_x' with input τ_x
17                         set Trail(τ_x') = Trail(τ_x)
18                         Forward τ_x' to a neighbour w_k
19                     else
20                         if Trail(τ_x) > Trail(τ_x') and T^0(τ_x) ≤ T^0(τ_x') then
21                             Create transaction τ_x' with input τ_x
22                             set Trail(τ_x') = Trail(τ_x) + 1
23                             Forward τ_x' to a neighbour w_k
24                         else
25                             Create transaction τ_x' with input τ_x
26                             set Trail(τ_x') = 0
27                             Forward τ_x' to a peer w_k whose public key is
                               Origin(τ_x)
28             β ← create new block
29             Solve BC puzzle and augment BC and announce β
30             β' ← received new block
31             If β' is valid then Augment BC & announce β'
```

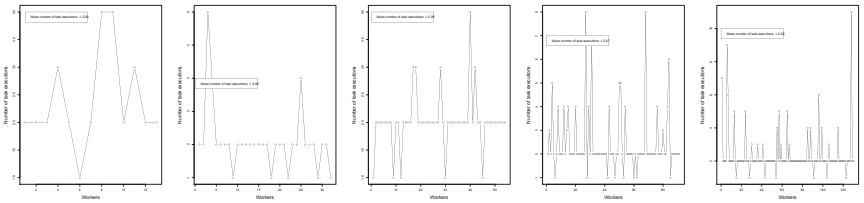
executes proof of work protocol with two more processes. The first process (line 28–29) creates and new block, solves the puzzle and announces puzzle solution and new block to the network. The second process (line 30–31) receives a new block from its neighbour. If the peer can verify that the block and associated

puzzle solution is correct then it augments its BC according to the rules described in Sect. 3 and it forwards the block to its neighbours. The second process may interrupt the block creation process if a new received block contains transactions which are included in the new block under construction of the former process. If the block creation process is interrupted then the peer restarts it.

First, we analyze convergence time of the distributed coordination mechanism. We measure convergence time as the minimum number of iteration required by the above simulation to find a valid solution for each subtasks. Note that 'time' is measured as the number of iteration. We may calculate the minimum convergence time for a centralized coordinator as follows: It will allow subtasks with no constraints among them to run in parallel and other tasks will be executed serially. There are at most x^2 subtasks for a puzzle with puzzle size x. As a subtask has common constraint with at most $2x$ other subtasks, in every iteration $x^2/2x = x/2$ subtasks can be executed in parallel. Hence the minimum convergence time is $x^2/(x/2) = 2x$. We use 9 datasets as sudoku puzzles with size from 4 to 12. The number of subtasks (and workers) for each dataset is at most x^2, each worker has 2 constraints and number of variables in each constraint is at most x where x is the puzzle size. Figure 3 shows the convergence time for these datasets. It shows the number of workers with valid solutions for their respective subtasks w.r.t time. We observe that convergence time increases as we increase the puzzle size. We conclude that distributed coordination mechanism has finite and short convergence time.

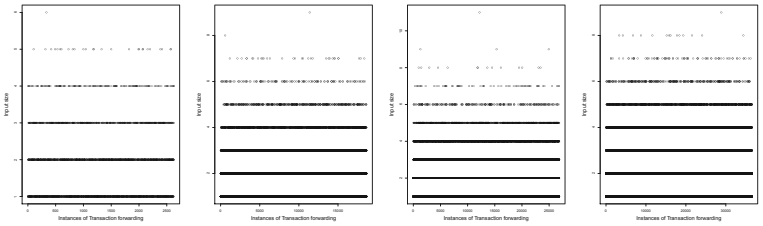
Next, we measure the efficiency of the distributed coordination mechanism in terms of the number of times each worker solves its subtask. The most efficient coordination mechanism requires each worker to execute its subtask only once. Although this problem can be classified as distributed constraint satisfaction problem and the complexity of such problems is [15] NP-complete. Hence it is unlikely that there exists the most efficient algorithm. We evaluate the efficiency of the proposed distributed coordination mechanism with datasets consisting of sudoku puzzles with puzzle size 4, 6, 8, 10, 12 and 14. Figure 4 shows the efficiency results as we plot the number of times each worker execute its subtask. We found that on average each worker executes its subtask twice.

Finally, we measure the computational overhead of the proposed distributed coordination mechanism. We measure it as the number of times each worker needs to execute their subtasks and input size of each transaction forwarding instances. In Fig. 5 we plot the average number of transactions per transaction forwarding instances. We found that the input size remains approximately 2 while we increase puzzle size from 4 to 14. Also as shown in Fig. 4 workers need to execute subtasks approximately twice while we increase puzzle size from 4 to 14. Hence it shows that both these parameters do not increase as the puzzle size is increased. Hence we claim that computational overhead of the proposed distributed coordination mechanism is negligible.



(a) Puzzle size 4 (b) Puzzle size 6 (c) Puzzle size 8 (d) Puzzle size 10 (e) Puzzle size 12

Fig. 4. Efficiency of the distributed coordination mechanism



(a) Puzzle size 4 (b) Puzzle size 10 (c) Puzzle size 12 (d) Puzzle size 14

Fig. 5. Computational overhead of the distributed coordination mechanism

6 Related Literature

It is difficult to verify the solution produced by the workers. As mention by [5] low quality workers and spammers are big threats to CS. Workers may also collude in data labelling tasks [7]. [1] develop an algorithm to search expertise in decentralised social network and provide incentives for them. Various mechanisms are developed to address the problem of low quality work in CS. In [13] trust and reputation are used to identify honest workers. In another approach, mechanism design is used to encourage the workers to remain honest. These mechanisms [2,8,9] developed rules for paying the workers in such a way that honest workers receive better payment than dishonest workers. It should be noted that these CS tools can be used in CS tasks which require coordination among workers. Algorithms developed for distributed constraint satisfaction [15] may be used to coordinate the workers. [12] proposed task coordination for CS but these algorithms do not guarantee the security of CS platform and workers may collude to overwrite transactions which recorded their work history. The first BC mechanism [10] uses proof of work as the distributed consensus protocol. Peercoin (https://peercoin.net/) introduced the proof of stake protocol which uses stake as the voting power instead of computing resource [4].

7 Conclusion

In this paper, we have proposed a BC-based solution for CS complex task which requires coordination among the workers. The BC provides a secure CS environment which does not need a trusted coordinator.

Acknowledgement. This publication has emanated from research supported in part by a researchgrant from Science Foundation Ireland (SFI) under Grant Number $SFI/12/RC/$ 2289-P2(Insight) and by a research grant from SFI and the Department of Agriculture, Food and the Marine on behalf of the Government of Ireland under-Grant Number SFI/12/RC/3835 (VistaMilk), co-funded by the European Regional Development Fund.

References

1. Ara, S.S., Thakur, S., Breslin, J.G.: Expertise discovery in decentralised online social networks. In: ASONAM 2017 (2017)
2. Kamar, E.: Incentives for truthful reporting in crowdsourcing. In: AAMAS 2012 (2012)
3. Khatib, F., Cooper, S., Tyka, M.D., Xu, K.: Algorithm discovery by protein folding game players. Proc. Natl. Acad. Sci. **108**, 18949–18953 (2011)
4. King, S., Nadal, S.: PPCoin: peer-to-peer crypto-currency with proof-ofstake (2012). http://www.peercoin.net/assets/paper/peercoin-paper.pdf
5. Kittur, A., Chi, E.H., Suh, B.: Crowdsourcing user studies with mechanical turk. In: CHI 2008 (2008)
6. Kittur, A., Nickerson, J.V., Bernstein, M., Gerber, E., Shaw, A.: The future of crowd work. In: CSCW 2013 (2013)
7. Lee, K., Tamilarasan, P., Caverlee, J.: Crowdturfers, campaigns, and social media: tracking and revealing crowdsourced manipulation of social media. In: ICWSM 2013 (2013)
8. Liu, S., Miao, C., Liu, Y., Yu, H., Zhang, J., Leung, C.: An incentive mechanism to elicit truthful opinions for crowdsourced multiple choice consensus tasks. In: WI-IAT (2015)
9. Miller, N., Resnick, P., Zeckhauser, R.: Eliciting informative feedback: the peer-prediction method (2009)
10. Nakamoto, S.: Bitcoin: a peer-to-peer electronic cash system (2009). http://www.bitcoin.org/bitcoin.pdf
11. Olson, D.L., Rosacker, K.: Crowdsourcing and open source software participation. Serv. Bus. **7**, 499–511 (2013)
12. Rahman, H., Roy, S.B., Thirumuruganathan, S., Amer-Yahia, S., Das, G.: "The whole is greater than the sum of its parts": optimization in collaborative crowdsourcing. CoRR (2015)
13. Ren, J., Zhang, Y., Zhang, K., Shen, X.: SACRM: social aware crowdsourcing with reputation management in mobile sensing. CoRR (2014)
14. Tavakoli, A., Nalbandian, H., Ayanian, N.: Crowdsourced coordination through online games. In: HRI 2016 (2016)
15. Yokoo, M., Durfee, E.H., Ishida, T., Kuwabara, K.: The distributed constraint satisfaction problem: formalization and algorithms. IEEE Trans. Knowl. Data Eng. **10**, 673–685 (1998)
16. Zhang, H., Law, E., Miller, R., Gajos, K.: Human computation tasks with global constraints. In: CHI 2012 (2012)