COCP: Blockchain Proof-of-Useful-Work Leveraging Real-Life Applications

Tatjana Davidović*, Milan Todorović*, Dušan Ramljak+, Tatjana Jakšić Krüger*, Luka Matijević*, Djordje Jovanović*, Dragan Urošević*
*Mathematical Institute, SASA
Belgrade, Serbia
E-mail: tanjad@mi.sanu.ac.rs, mtodorovic@mi.sanu.ac.rs, tatjana@mi.sanu.ac.rs, luka@mi.sanu.ac.rs, giorgaki.jovanovic@gmail.com, draganu@mi.sanu.ac.rs
+Penn State Great Valley, The Pennsylvania State University
Malvern, PA, USA
E-mail: dusan@psu.edu

***Abstract*—We propose Combinatorial Optimization based Consensus Protocol (COCP) that considers efficient use of energy in Blockchain (BC) consensus protocol based on *Proof-of-Useful-Work* (PoUW) concept. Instead of classical cryptographic puzle, it involves dealing with hard real-life combinatorial optimization (CO) problems submitted by BC participants called customers.**

Two sources of rewards are provided for miners, one related to adding a new block and the other for solving an instance of CO problem. The main issues that arise when combining BC and CO, are security and consistency of maintaining the whole system. We discuss how to resolve them and what are the benefits of the proposed COCP for all BC participants. Our proposed doubly-rewarding scheme and efficiency in energy exploration are illustrated on a small example from the Ethereum BC network.

***Index Terms*—Distributed databases, autonomous systems, efficient consensus protocols, combinatorial optimization, heuristics**

I. Introduction and Literature Review

Blockchain (BC) is a highly energy intensive system, due to the large computing overhead used for its autonomous maintenance, i.e., for the execution of Proof-of-Work (PoW)-based consensus protocol [1], [2], [14]. We are addressing different ways to utilize available resources by solving instances of hard real-life Combinatorial Optimization (CO) problems during the execution of consensus protocol.

In the literature, the corresponding consensus protocols are known as *Proof-of-Useful-Work* (PoUW) [1], [3]–[9], [12], [13] and they provide incentives and advantages for all BC participants.

Various PoUW approaches are proposed, and majority of them involve solving hard CO problems [1], [4], [5], [12] or Artificial Neural Network (ANN) training [3], [7]. These ideas are promising as they ensure employing the resources for some useful computations and provide two sources of reward for miners: the basic one related to the insertion of new block into BC and the other one coming from the customers whose problem instances are solved within PoUW. The main problem is that most of the papers only present the ideas, discuss their advantages, and describe some evaluations, mostly on a conceptual level. Only a few of them [1], [4], [12] present detailed description of PoUW based BC systems providing some information on implementation and/or experimental evaluation. In [1] PoUW was based on solving CO problems represented by polynomials and treated as Orthogonal Vectors. Consensus protocol in [12] requires users to submit the algorithms for solving their problems, which is unrealistic. The users that have such algorithms would solve the problems themselves, probably outside BC network. DPLS generic algorithm is suggested to solve CO problems in [4], however, generic solvers might work inefficient for some problems.

To overcome these limitations we propose our Combinatorial Optimization based Consensus Protocol (COCP) approach that involves possibility to submit arbitrary CO problem instance for which solution methodology is provided within BC network.

In order to correctly implement PoUW-based BC system, several issues should be resolved:

1) The format for stating problem instances
2) Correspondence between problem instance and the composed block
3) Large enough number of instances to make this correspondence meaningful
4) Controlling the hardness of problem instances
5) The efficient exploration of dedicated hardware already owned by miners
6) Security issues in managing data

Serbian Ministry of Education, Science and Technological Development, Agreement No. 451-03-9/2021-14/200029; Science Fund of Republic of Serbia, under the project "Advanced Artificial Intelligence Techniques for Analysis and Design of System Components Based on Trustworthy BlockChain Technology (AI4TrustBC)"; Penn State Great Valley Big Data Lab.

978-1-6654-9958-3/22/$31.00 ©2022 IEEE

When developing our COCP, we consider listed issues and explain them in more details in this paper using an illustrative example. Our example is based on the Ethereum BC network (https://ethereum.org/en/). Ethereum is a decentralized, open-source blockchain with smart contract functionality, where anyone can join, participate in its functioning, and leave at any time. Ethereum implementation is transparent and open source, and thus represents a great potential to explain our implementation.

In the remainder of this paper, we explain how to deal with the above mentioned issues section (II), show more details through our illustrative example (section III) and provide concluding remarks (section IV).

II. Methodology

The COCP implementation requires modification of the usual BC environment. First, along with basic users, miners and verifiers, we introduce a new category of participants named *customers*. They are usually companies or individuals dealing with hard instances of real-life CO problems and willing to pay for obtaining high quality solutions. Customers could play any other role, especially the role of basic users, as their payments could could be performed via regular transactions in the proposed BC system. Next, an *instance pool* containing CO problem instances should be introduced and a format for submitting instances has to be defined. Contrary to other papers from the literature, we allow customers to submit instances of different CO problems and provide proper user interface as well as different solution methods. BC can be considered as a live system to which we can easily append new modules related to defining and solving different types or variants of CO problems. Submitted CO problem instances that are stored in the instance pool are considered active and should be selected by miners who compose blocks to be added to BC network. When an instance is solved and solution provided to the customer, it is removed from the instance pool and, together with the solution, stored in *instance archive*.

In the classical PoW consensus protocol, each block has to store the hash value of its predecessor to keep the consistency. To prove that they invest some work before adding new blocks, miners need to solve the cryptographic puzzles, i.e., to find the appropriate *nonce* value that, combined with block header, results in block hash value smaller the a given threshold. COCP aims at performing some useful work instead of solving cryptographic puzzles, and therefore, we need to re-define the proof of the invested work. To be allowed to add a block to BC, a miner has to solve an instance of real-life CO problem submitted by one of the customers. We need to establish the correspondence between the composed block and an CO problem instance so that verifiers can easily check not only if the selected transaction are valid but also if the proper instance is solved by miner. As the content of instance pool in each time point is common knowledge of all BC participants, we defined the connection between block and instance as the results of modulo operation applied to block hash value and number of instances in the pool.

COCP has to be supplied with large enough number of CO instances such that different blocks always correspond to different CO instance. However, if at some point there are no CO problem instances in the pool, miners could be provided classical cryptographic puzzles that should mimic instances of CO problems.

To keep the frequency block of insertion, it is necessary to control the hardness of CO problem instances. Some of the instances may solved very efficiently, however, for customers it is important to obtain their solutions. On the other hand, some CO problem instances may be very hard to be solved efficiently and miners that are dealing with them may have difficulty to add new blocks. However, in COCP it is not necessary to always obtain optimal solution for the considered instance. In the input data the customers provide solution threshold, i.e., the bound on the desired objective function value. As soon as a solution with the objective function that meets the given threshold is found, the miner can publish the corresponding block. If the time required to obtain the desired quality solution is too long, it is possible to ask the customer to adjust the threshold value or to enable that solution process continues from the current search state next time the same instance is selected.

Dealing with the selected CO problem may require special resources, both hardware and software. Obtaining adequate software should be easier for a miner because there exist efficient codes for a lot of optimization methods. On the other hand, some of these software packages require hardware resources that are not usually owned by a typical miner. In the literature usually it is suggested to use hybrid approach: miners can choose between the classical PoW and PoUW consensus protocol depending on their preference and the available hardware resources. However, by doubly rewarding scheme, miners are motivated to select and solve CO problem instances from the pool. We also consider the development of optimization algorithms that would engage the hardware already owned by miners.

Regarding BC data security issues, we considered preventing of various malicious behavior related to tampering with input data and solution data of CO problem instance. Both data types could be very large, and thus, are not included in the BC network directly, as parts of blocks. Instead, they are kept outside BC, in some public locations accessible to all participants. The location address of input data is provided when submitting instance, via special type of transactions or smart contracts. Data related to valid solution, miner stores in the private location, predefined and known to all participants. A pointer to the transaction or smart contract that corresponds to the solved CO problem instance, miners includes in the block's header making it easy for all participants to access the input data. Storing a hash value of the problem data in the block's header can prevent malicious users who may try to change the input data of CO problem instance, and thus introduce the confusion between instance and its solution. In such a way any change can be detected by comparing this hash value with the one calculated from the file defined by

the location address.

The second COCP security issue that we consider is preventing miners to perform selfish mining and other types of known frauds [10], [11]. For example, a miner, pretending to be a customer, submits the already solved CO problem instance and tries to compose a block of transaction corresponding to that instance. To discourage this, we ensure that large enough number of CO problem instances is always provided. With increasing the number of instances in the pool, the effort needed to compose the block that corresponds to any particular instance is as hard as the effort required for actual solution process.

Another type of fraud that we are preventing is miner's attempt to announce a new block before obtaining a valid solution and then continue solutions process with the hope to find the solution before the verifiers approve the block. Before publishing the composed block, miner has to store both the hash value of the instance input data and of the solution in block's header.

In the process of block verification, the verifiers access the instance data via the Corresponding location from block's header and the solution from the miner's private location. Verification consists of calculating hash values of input data and of the solution and comparing them with the corresponding hash values stored in the block's header. If any pair of hashes does not contain the same values, we can detect malicious behaviour and discard the composed block because the BC security is broken.

III. Experimental Results and Discussion

To illustrate our proposed COCP, we performed simulation of adding new blocks on a small example containing 4 blocks ($B1, B2, B3, B4$) from the publicly available Ethereum's test network Ropsten. Here, we skip many details about the implementation of the corresponding BC system, and focus only on the parts related to COCP.

Let the instance pool contain 10 CO problem instances: $i0, i1, \ldots, i9$. As we already mentioned, we defined the connection between block and instance as the results of modulo operation applied to block hash value and number of instances in the pool. It is a bit-wise xor operation on problem instance hash and solution hash values. To simplify, in this example we put the hash value of the instance input data file name in the **nonce** field. We needed to specify both these values in order to calculate the hash value of the complete block, as it represents the connection between blocks in BC network and is stored in the header of the next added block.

Mining steps are performed by 5 miners on Ubuntu 20.04, Intel Core i7-10750H CPU @ 2.60 GHz, 32 GB RAM and NVIDIA GeForce RTX 2060 with 6 GB GPU memory. Mining software is based on the latest version of standard Ethereum Go client[1] and uses the Ethereum's PoW algorithm hash function *keccak256*.

[1]Go Ethereum: Official Go implementation of the Ethereum protocol (`https://geth.ethereum.org/`

Although at some steps we assumed that different miners are creating different blocks, our goal was to propagate the current block from the example to illustrate direct, as well as side effect benefits of using our COCP PoUW-based consensus protocol. Moreover, in this simulation the time stamp and miner identification fields were not changed because they do not have significant impact on the simulation results.

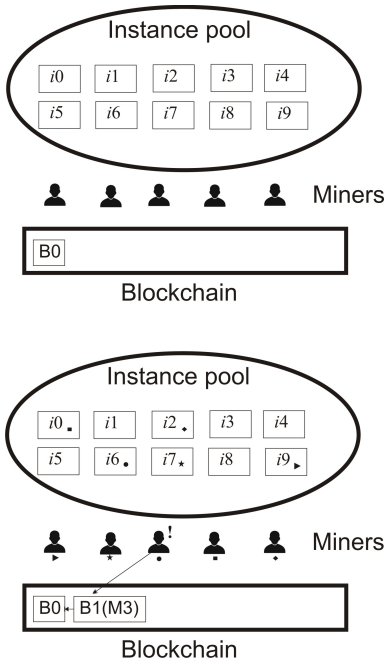

Fig. 1. Adding $B1$, successful miner is marked with exclamation point

The first block insertion in the considered BC system is illustrated in Fig. 1. We need the generic block $B0$ to provide the previous block hash value for $B1$. Assuming the simplest scenario, all miners compose their blocks and select the corresponding instances for solving (as it is indicated by the various geometric shapes in Fig. 1). Miner $M3$ was the fastest in providing the valid solution (the one that satisfies the given threshold) for instance $i6$ and is allowed to add $B1$ to BC and to gain the total reward (for inserting block and solving the instance).

For the second block insertion, again all miners composed their blocks and selected the corresponding instances. However, now scenario is that miners $M2$ and $M5$ found valid solutions almost at the same time. As only one block can be added to BC, let verifiers decide on $M2$ who composed block $B2$ and solved instance $i2$. This miner is doubly rewarded: for adding a block and solving an instance. Miner $M5$ is rewarded only for providing solution for instance $i9$.

Before considering $B3$, there are 7 instances in the pool, and we assume that miners composed their blocks and, consequently, selected five different instances ($i0, i1, i3, i5, i7$) for solving (see Fig. 2). Different geometric shapes denote correspondence between miners and instances. Miners M2, M4, and M5 solved their instances almost at the same time,

and all three of them are trying to add their blocks and announce the corresponding solutions to customers. This will generate a fork in BC presented at the upper part of Fig. 2. Although (after the synchronization step) only $B3$ (composed by miner M4) will remain in BC (lower part of Fig. 2), three problem instances will be removed from the pool and their solutions provided to the customers. Miner $M4$ is rewarded for both tasks (adding a block and solving instance), while $M2$ and $M5$ get reward only for finding the solutions of the corresponding instances.

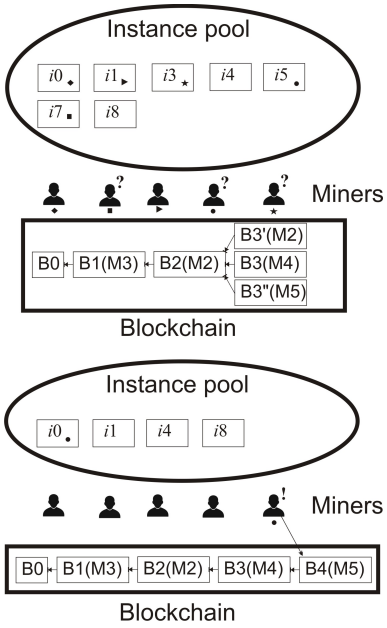

Fig. 2. Adding $B3$ and $B4$, question marks indicate need for synchronization

Having removed these 3 instances from the pool, 4 instances remain before $B4$ is to be added. We assumed that $M5$ added it upon solving instance $i0$.

From the performed simulation, one can clearly see the benefits of our approach: in addition to inserting 4 blocks into the BC network, 7 instances of various CO problems have been solved. This means that the outcomes of the utilised resources (consumed energy) are twofold, not only the new blocks are added, but also some useful work is performed.

IV. Conclusions

We presented the details of our Combinatorial Optimization based Consensus Protocol (COCP). The main advantages of the envisioned consensus protocol are efficient utilization of computing resources and various sources of rewards for participants. The new type of BC users, customers are introduced to supply COCP with CO problem instances. Solving real-life instances of Combinatorial Optimization (CO) problems while attempting to include block into BC helps both customers and miners. In COCP miners are rewarded for adding block and/or solving instances, while customers are rewarded by obtaining solutions of their instances. Inclusion of each block is performed according to one of the possible scenarios that can occur during the block mining process. Using a small example consisting of 4 blocks from the publicly available Ethereum's Ropsten network we illustrated the benefits of our protocol.

COCP relies on the implementations of the underlying concept of consensus protocol like Proof-of-Useful work for majority of the details. Possible directions for future work may include detailed implementation of all components of COCP along with identifying and resolving issues that may arise. It is also important to identify and resolve all security challenges that come with solving real-life CO problem instances, introducing new type of participants, and providing various incentives for them. Limitations of our work lie in increasing computational efficiency and solving the latency issue. That is another important research avenue that could be followed through introduction of distributed computing as a tool. It leads us to the *pool of miners* concept: several miners join resources and divide the required computations/corresponding reward.

References

[1] M. Ball, A. Rosen, M. Sabin, and P. N. Vasudevan, "Proofs of useful work," IACR Cryptology ePrint Archive (https://eprint.iacr.org/2017/203.pdf), 2017, last update 2021.

[2] S. M. H. Bamakan, A. Motavali, and A. B. Bondarti, "A survey of blockchain consensus algorithms performance evaluation criteria," *Expert Systems with Applications*, pp. 113 385:1–21, 2020.

[3] C. Chenli, B. Li, Y. Shi, and T. Jung, "Energy-recycling blockchain with proof-of-deep-learning," in *2019 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*. IEEE, 2019, pp. 19–23.

[4] M. Fitzi, A. Kiayias, G. Panagiotakos, and A. Russell, "Ofelimos: Combinatorial optimization via proof-of-useful-work\a provably secure blockchain protocol," IACR Cryptology ePrint Archive (https://eprint.iacr.org/2021/1379.pdf), 2021.

[5] M. Haouari, M. Mhiri, M. El-Masri, and K. Al-Yafi, "A novel proof of useful work for a blockchain storing transportation transactions," *Information Processing & Management*, vol. 59, no. 1, p. 102749, 2022.

[6] B. Li, C. Chenli, X. Xu, T. Jung, and Y. Shi, "Exploiting computation power of blockchain for biomedical image segmentation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, 2019, pp. 2802–2811.

[7] B. Li, C. Chenli, X. Xu, Y. Shi, and T. Jung, "Dlbc: A deep learning-based consensus in blockchains for deep learning services," *arXiv preprint arXiv:1904.07349v2*, 2020.

[8] A. Lihu, J. Du, I. Barjaktarevic, P. Gerzanics, and M. Harvilla, "A proof of useful work for artificial intelligence on the blockchain," *arXiv preprint arXiv:2001.09244*, 2020.

[9] C. Qiu, X. Wang, H. Yao, J. Du, F. R. Yu, and S. Guo, "Networking integrated cloud-edge-end in iot: A blockchain-assisted collective q-learning approach," *IEEE Internet of Things Journal*, 2020.

[10] M. Saad, J. Spaulding, L. Njilla, C. A. Kamhoua, D. Nyang, and A. Mohaisen, "Overview of attack surfaces in blockchain," *Blockchain for distributed systems security*, pp. 51–66, 2019.

[11] S. Shalini and H. Santhi, "A survey on various attacks in bitcoin and cryptocurrency," in *2019 International Conference on Communication and Signal Processing (ICCSP)*. IEEE, 2019, pp. 0220–0224.

[12] N. Shibata, "Proof-of-search: combining blockchain consensus formation with solving optimization problems," *IEEE Access*, vol. 7, pp. 172 994–173 006, 2019.

[13] W. A. Syafruddin, S. Dadkhah, and M. Köppen, "Blockchain scheme based on evolutionary proof of work," in *2019 IEEE Congress on Evolutionary Computation (CEC)*. IEEE, 2019, pp. 771–776.

[14] Z. Zheng, S. Xie, H. Dai, X. Chen, and H. Wang, "An overview of blockchain technology: Architecture, consensus, and future trends," in *IEEE international congress on big data (BigData congress)*. IEEE, 2017, pp. 557–564.