Fall 9-17-2021

A Novel Approach for Smart Contracts Using Blockchain

Dr khaled nagaty
khaled.nagaty@bue.edu.eg

Manar AbdElhamid
The British University in Egypt, Manar.Abdelhamid@bue.edu.eg

A Novel Approach for Smart Contracts Using Blockchain

Manar Abdelhamid and Khaled Nagaty

Abstract Despite all the advantages delivered by smart contracts deployed on top of the blockchain, several challenges are hitting the industry. Blockchain has many security and performance issues that need attention. It facilitates the interaction between two parties who can interact without a third party to accept this transaction. This leads to the creation of smart contracts that help the blockchain to execute more efficiently. A smart contract is an executable code that runs on top of the blockchain. Any modification after execution is not allowed. This causes a problem that is not solved by many scientists. This paper proposes a solution to this problem by creating a modifying blockchain that will hold all the modified data to be added to the smart contract using Ethereum. The proposed solution allows the modification to happen in a separate blockchain associated with the main blockchain containing the smart contract. This will result in faster execution, easier modifying and less used storage. Ethereum used to deploy the blockchain while deploying the smart contract requires a programing language that supports all the necessary requirements for deployment. Solidity programing language is used as it provides all the functions and operations for creation.

Keywords Smart contract · Blockchain · Efficiency · Modifying · Ethereum

1 Introduction

Transactions that occur in any traditional system usually happened in a centralized way which requires the involvement of a trusted third party [1]. Blockchain is a decentralized system that exists between all parties, which will lead to no need to pay intermediaries to help in the process. In other words, blockchains allow untrusted

M. Abdelhamid (✉) · K. Nagaty
The British University in Egypt, Cairo, Egypt
e-mail: Manar.Abdelhamid@bue.edu.eg

K. Nagaty
e-mail: Khaled.Nagaty@bue.edu.eg

X.-S. Yang et al. (eds.), *Proceedings of Sixth International Congress on Information and Communication Technology*, Lecture Notes in Networks and Systems 236,
https://doi.org/10.1007/978-981-16-2380-6_72

entities to interact with each other in a trusted manner without the involvement of a trusted third party. Blockchain technology has been invented and modified to tackle the single point of failure and high transaction fees that occurs in traditional systems [1]. Simply speaking, a blockchain is a distributed ledger that records all the transactions that occurred in a network. Each network node maintains the distributed ledger, so each node has a transaction that occurred over the network. The transaction of data will be executed from one node to another. All inserted information in a blockchain will be made public and cannot be modified or erased. Applying blockchains help in saving time and conflict between the systems. They are faster, cheaper and more secure than the traditional systems, so it is important to convert all traditional systems to decentralized systems and depend on smart contracts to reduce the paperwork in order to be more professional and secure. Also, bitcoin, which is a peer-to-peer digital payment system, is originally introduced by blockchains [2]. Bitcoin is used for a large decentralized range of applications.

Blockchains replicate and share data between peer-to-peer networks. Blockchains were initially introduced by Satoshi Nakamto, who developed bitcoin to transfer digital currencies directly without the need of third parties [3]. In the blockchain, each block refers to the next block using its address. The blocks are linked together and executed one after the other without any modification or change. The first block in the blockchain is called the genesis block [4], which contains no address. As mentioned, the blockchain has many applications to do transactions; one of them is smart contracts.

A smart contract is a new concept that has to be deployed on top of the blockchain. It is an executable code that runs automatically on the blockchain with rules to facilitate and enforce the terms of agreements between untrusted parties to work together [5]. The contract is stored on the blockchain like any transaction and runs its terms without the need of trusted intermediaries [4]. Compared with a traditional contract, a smart contract does not rely on a third party to trust the operation, and this results in fast transactions. In the traditional contract, you need a third party and a lot of paperwork to reach the required result; this will require much effort to reach the data needed and much time to spend. Blockchains can be applied on different platforms that help in developing smart contacts. The most common platform to deploy smart contracts that runs on the blockchain is Ethereum [2]. Ethereum language supports a complete feature and easy use of smart contracts that allow more creative and advanced customized contracts [1].

Traditional contracts and paperwork have been a serious problem in wasting resources in the last decade. Creating smart contracts can help the world and save the wasted resource and facilitates the process. Smart contract is a program that holds a set of rules. The smart contract executes its set of agreements. This execution takes place through Ethereum virtual machine (EVM). Smart contracts are deployed to the blockchain by submitting a contract creation transaction [4]. Once this transaction occurred and got accepted, the Ethereum accounts can invoke this contract and its functions. This invocation happened when the code is totally executed and the transaction is sent to the address of the next node [3]. Smart contracts can help in many fields such as supply chains and in various applications such as voting management

or health care management systems [5]. The smart contracts work when a set of code deployed on the blockchain start trigging by the identifier address. This will lead to total execution of the smart contract by all miners.

The problem that the developers might face during writing smart contracts is the loss of data and the inability to modify the block with data and much time spent to fetch the needed information. The goal of this work is to help convert all the traditional methods and paperwork to smart contract technology. These smart contracts will provide multiple advantages over the traditional arrangements that can be classified into three categories:

1. Provide the systems with more accuracy and transparency: this is one of the most important requirements provided by smart contracts, which help in fasting the process of extracting any needed information. If the accuracy is high, this leads to low transaction errors [6]. At the same time, the terms and conditions of these contracts are fully visible and accessible to all relevant parties, which helps in a high level of trust.
2. Increase the efficiency: higher efficiencies in the results of the transactions processed per unit of time. Moreover, higher speed and accurate output will be provided when using smart contracts [7].
3. Paper-free, easy storage and backup. These contracts record essential details in each transaction. Therefore, anytime your details are used in a contract, they are permanently stored for future records, which will lead to a lower amount of paper used, which removes the need for vast reams of paper. Also, backup all the information so that it will help in preventing data loss.

A smart contract is an executable code that could be deployed on top of a block in a blockchain. These contracts have a set of rules that are written using solidity language. Solidity provides a contract with functions and rules that help a transaction to meet the needed purpose. The smart contract has many problems that limit its usage. Identifying the gaps as the inability to modify the contract after the execution was one of the main problems. The solution presented in the state-of-the-art was to create another block and copy all the data of the contract into it and finally add the modified part. This solution leads to inefficient use of storage. Another solution was to create a template that will be deployed on the contract so that changes will be available anytime. This solution will need to categorize all the contracts, as one template will not fit all types of contracts. Therefore, much time and more space will be consumed.

This paper proposes a different perspective of blockchains and the deployment of the smart contracts onto them. It is organized as follows: Section 2 discusses all the previous work problems, how they solved them and the gaps. Section 3 discusses the proposed solution for modifying smart contracts. Section 4 includes the experimental results and its environment. Section 5 presents the discussion. Finally, Sect. 6 concludes this paper and the future work.

2 Related Works

Different research works proposed methods for solving the smart contract deployed on blockchain issues and its modification. Delmonline et al. [1] discussed the difficulty of writing a smart contract and its correctness and classified them under the codifying issue. The reason to have the right smart contract is that saving the actions made for any transaction is very important in order not to lose any data or important information. If the smart contract is not executed in the right manner, some of its currency in the transaction will disappear, which leads to the loss of data. Delmonline et al. [1] tackled this issue by creating a semi-automated smart contract to ease the process of writing it. In addition to this, this contract will handle the human-readable contract and turn them into rules for the smart contract. The problem with the semi-automatized smart contract is that any modification is not allowed because any smart contract after being deployed over the blockchain will not have the access to modify the data into it. This will make the smart contract not handy and if any change is needed will require attaching another copy to the blockchain that will take much memory. This solution to modify a deployed contract will require duplicating the space of the block because of adding an extra block to the blockchain to perform the modification.

Another approach introduced by Marino and Juels [8] is the ability to modify any smart contract using a pre-defined set of rules or templates. This pre-defined set of rules will be changed according to the legal law. This standard that is pre-defined will be then applied to the Ethereum-based smart contract to start compiling with the contract. The proposed solution by Marino et al. is too vague, as these templates could not be applied to all categories of contracts as law changes from one category to another. In other words, if the deployed contract was for health care, all rules and templates are also applied to the supply chain category. In order to solve this, it is required to categorize the rules and templates and deploy them all on the blockchain, which will also use a large amount of memory. So this solution is also inefficient regarding the space complexity as that of Delmonline et al. [1].

More approaches are introduced to tackle the smart contract transaction-ordering dependency issue. Natoli and Gramoli [9] mentioned that if having two dependent transactions that is invoked at the same time to the same contract that is included in one block, the order of execution of the contract will depend on the miner. This means if the miner executes a transaction or accepts one before the other, it will cause a loss of data. So Natoli and Gramoli [9] suggested using the Ethereum-based function. This function will be responsible for sending and receiving responses to enforce the order of transactions. This solution will waste time waiting for a response. The contract's response depends on the number of functions executed and the time for execution. So, if the contract is waiting for a response from different functions, this results in a long execution time, which makes the processing of the contract slow.

Fernandaz et al. [10] concluded that there is an extensive processing time in the execution of the smart contract as processing requires updating on few centralized systems with extensive time for processing. This updating will depend on multiple

parties to execute. Moreover, several functions will need to execute that will lead to huge costs for data retrieval. Fernandaz's solution stores the data in each block in the blockchain in order to retrieve the data from all parties. Therefore, all the data will be duplicated on each block without any need for it, which consumes a large amount of memory and much time to retrieve any needed data that will take a long time to be loaded.

Vukolić [11] introduced another problem which is the smart contracts sequential execution of the contract. Smart contracts are executed on the blockchain one at a time. This means that by the time smart contracts increase they will slow the blockchain performance, as it should wait for each smart contract to execute. This means that the execution of the smart contracts per second will be very limited on the blockchain. Vukolić proposed execution of the smart contracts in parallel, as long as they are independent of each other.

3 Contribution

This paper proposes a modified blockchain associated with the main blockchain to solve the problems of modification and execution time of smart contracts. As smart contract after execution could not be modified and the alternative to modify it will result in more storage usage and extra time for execution, a solution based on modifying blockchain associated with the main blockchain containing the smart contract is introduced. The block in the modifying blockchain will point to the block containing the smart contract in the main blockchain. The main blockchain executes normally and when it comes to the smart contract, it looks for a modifying blockchain associated with the smart contract. If a modifying blockchain is found, control goes to the modifying blockchain to search for the latest modification of this smart contract. Finally, the smart contract and its latest modification are merged together and the control returns back to the main blockchain. This will lead to a very easy modification, much lesser time and much lesser space.

In Fig. 1, one main blockchain has multiple blocks. Each block in the chain will have different data. In block 2, for example, there is a smart contract that requires modification. As mentioned above, smart contracts could not be changed or modified, so a modifying blockchain is created for the smart contract if do not exist, and a new block that holds the modified data is added to this chain. This data will be ready to execute whenever the contract needs modification. The main blockchain executes if a smart contract is found. It looks for a modifying blockchain for the smart contract. If a modifying blockchain exists, the latest modification is retrieved and connected to the smart contract in the main block. By using the modifying blockchain, we can modify a smart contract after execution without duplicating the memory space and with very low execution time as follows: The main block that contains the smart contract did not have any changes, thus preserving the main characteristic of the smart contract which do not allow modification after execution. The proposed modifying blockchain

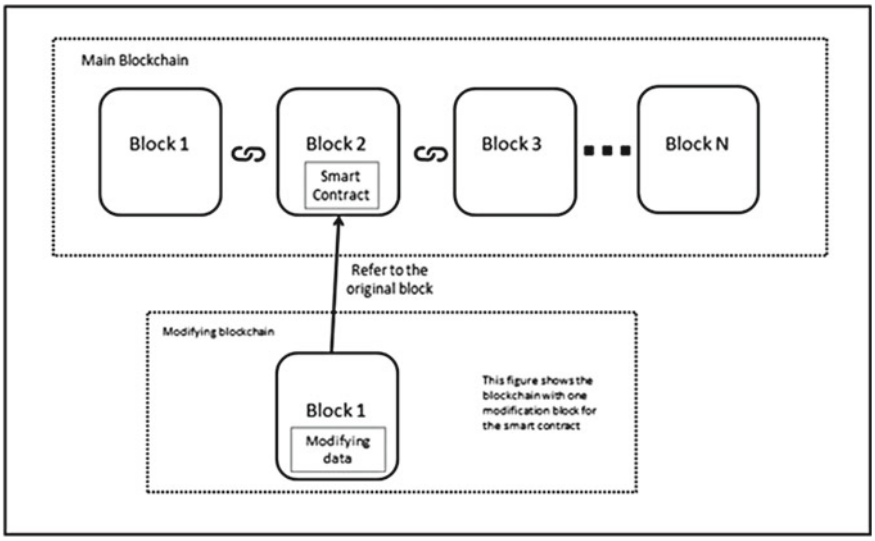

Fig. 1 Modified blockchain with smart contract deployed on it

contains modifying blocks that allow modifications to the main blockchain with low memory space and less execution time.

Figure 2 shows the case if more than one modification took place to the same contract. The first modification takes place in the first block of the modifying

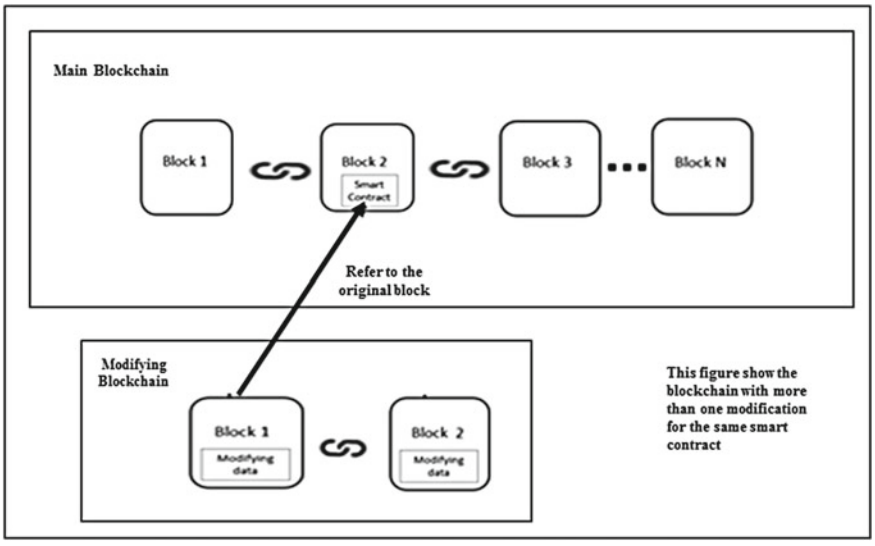

Fig. 2 Multiple modifying blocks associated with a smart contract in the main blockchain

blockchain associated with the smart contract. After execution, any modification could be added by adding another modifying block linked to the previous modifying block in the chain. This means that many modifications to the smart contract in the main blockchain can take place without duplicating the main block of the smart contract in each modification.

Each smart contract should have its own modifying blockchain. This will reduce the execution time significantly. A smart contract with no modifications has no modifying blockchain. If a smart contract requires modification, a modifying blockchain is created and associated with the main block containing the smart contract if this modifying blockchain does not exist. A look-up table contains the address of each main block containing a smart contract in the main blockchain and the address of the modifying blockchain associated with it. Note that the address of the first modifying block is the address of the modifying blockchain. When a smart contract executes, the main blockchain searches for the look-up table to see if a modifying blockchain for this smart contract exists. If it exists, then the control is transferred to the modifying blockchain to retrieve the latest modifications. The average time complexity using hash is O(1). To speed up the search process, a hash table could be used where the address of the main block containing the smart contract is input to a hash function and the output is the location in the hash table containing the address of the modifying blockchain associated with the main block. If there is an address corresponding to the address of the main block containing the smart contract, then this main block has a modifying chain, and if it is null, then this main block has no modifying blockchain. Using hash tables will make the execution significantly faster (Fig. 3).

In Fig. 4, the chart shows the flow of the blockchain deployed on top of the smart contract. The execution will be done when a smart contract is found in a block. The main block that contains the smart contract will search for the look-up table for the modified data in the modifying blockchain. If a modifying blockchain for this smart contract exists, then the modifying data is added to the smart contract. If there is no modifying blockchain associated with that smart contract, the blockchain will continue with the next main block in the main blockchain.

However, in the traditional blockchain, the execution happened respectively. This means that each block will execute and the address of it will be sent to the next block until all the blocks are executed. This means that any changes could not happen. Modifications will be done by adding new blocks to the blockchain, which will need the whole blockchain to execute all over again that results in high transactions. In Ethereum there are three types of execution: financial transfers, message calls and contract creations. Each one of these types contains basic elements *from, to, gas, value* and *data* [12]. The *from* and *to* functions represent the sender and the receiver. When the number of functions increases the execution time will increase as well as the *gas* (unit to measure the execution in Ethereum) will be consumed. The *value* is used to save the amount transferred in any transaction in the blockchain. The *data* field is used to hold any data in the blockchain. If any of these fields increase their usage the execution time will increase. The fee for a transaction with attached data covers the cost of storing the data permanently in the blockchain and is proportional to the size of the data [12]. In the modified blockchain, when a smart contract is

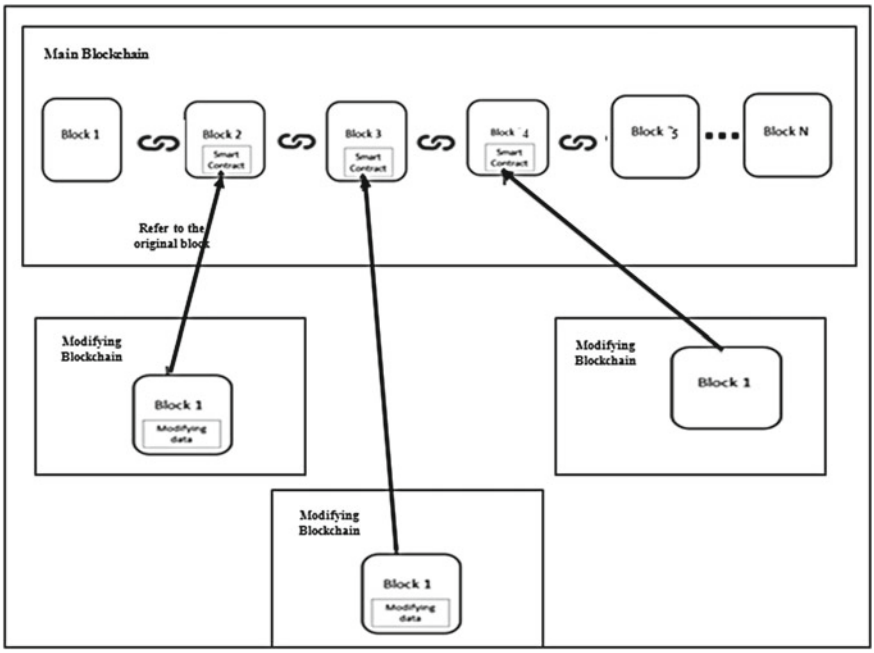

Fig. 3 Multiple modifications to different smart contracts in the main blockchain

created, a particular Ethereum address is subsequently used to interact with that contract. Then this address will check if any modification is associated with this block and fetch it. This will lower the transaction waiting time as well as much less memory will be used [12].

4 Experimental Results

To develop the proposed solution, a set of technologies was chosen. Ethereum is the most commonly used platform to create and execute smart contracts on the blockchain. This platform deploys the smart contract and executes its transactions very easily. To deploy the smart contract on top of the blockchain, solidity is the language that is used to create it. Solidity is a programing language that provides the contract with all its functions and operations for transactions.

Ethereum has a blockchain that contains many blocks with different data. These blocks have data that may contain smart contracts deployed onto them. In Ethereum, each node has a virtual machine called EVM which could be used to process a representation of a smart contract. Each block refers to the next one, but once it is executed, any modification could not be done. In Fig. 5, the structure of the Ethereum contains many layers. The first layer is the compiling of the smart contract that is

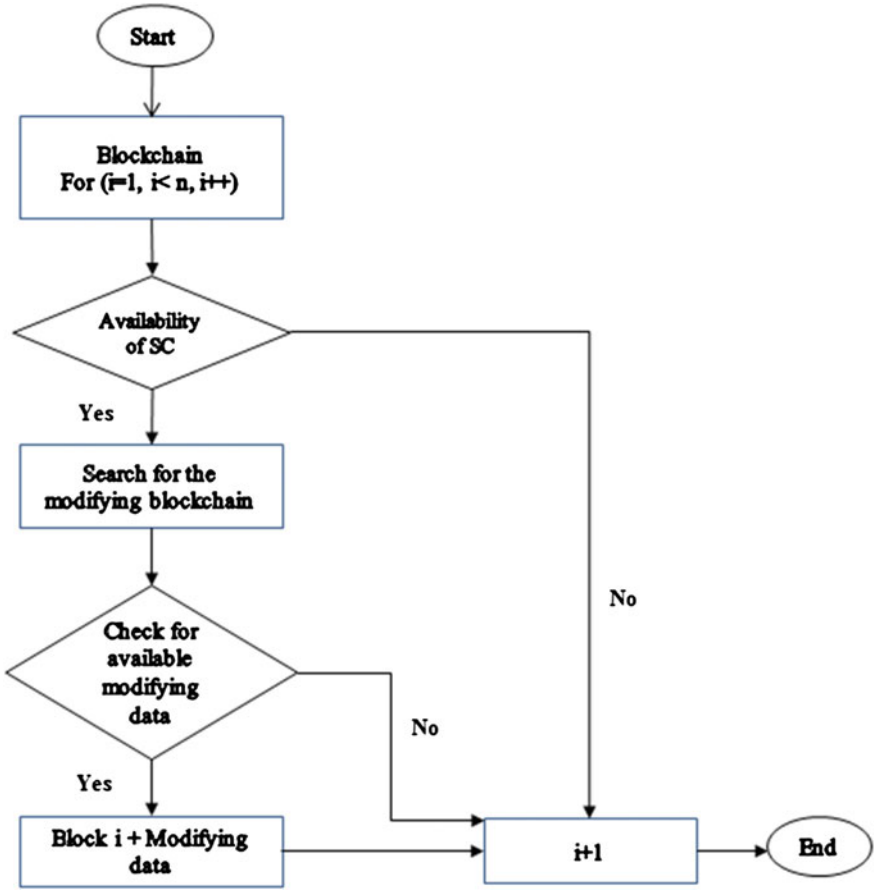

Fig. 4 Flowchart for modifying smart contract in blockchain

developed using solidity language. The second layer will contain the deployment method. This deployment will be on top of the main blockchain. These main blocks will contain many transaction functions, which is data. One of these main blocks will contain the smart contract that requires modification. These modifications will be performed through a modifying blockchain that is associated with a pointer to the main block which contains the smart contract that requires modification in the main blockchain.

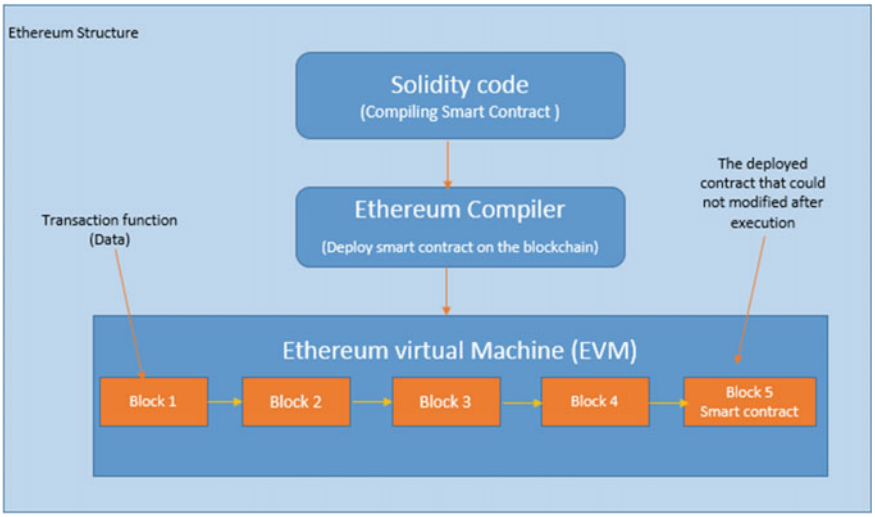

Fig. 5 Ethereum structure

5 Discussion

Comparing the modifying blockchain approach with the approaches mentioned by the authors Delmonline et al. [1], Marino and Juels [8] and Natoli and Gramoli [9], we find that the modifying blockchain consumes less memory and the execution time will be shorter due to the small number of functions to be executed. Moreover, the loss of data will be controlled as the modification will not take place in the original data. All modifications will be done in the modifying block. With this, the modifying block approach will be less time-consuming and memory-saving than the state-of-the-art.

6 Conclusion

Smart contracts have issues during implementation. One of these issues is the inability to modify or change the smart contract after execution. In this paper, we proposed modifying blockchain to modify smart contracts without losing data, consuming less memory and less execution time. To illustrate our approach more, modifying blockchain is conducted to allow the modifying in another block that is associated with the main block in the main blockchain that has the smart contract as you can't do any modification in the smart contract after execution. The modifying blockchain will be pointing to the main blockchain that has the original data, so whenever the contract needs these changes point out to it and get the updated version. As a part of future work, we will study how to create smart contracts on the blockchain with

different ways of modifications. We will use the hash table to speed up the execution time of smart contract processing. Also, we are planning to create a new method for modifying smart contracts using linked lists which provides us with more storage saving and much less execution time as it will provide a memory space that will hold the modified data in it.

References

1. Delmolino K, Arnett M, Kosba A, Miller A, Shi E (2016) Step by step towards creating a safe smart contract: lessons and insights from a cryptocurrency lab. In: Lecture notes in computer science (including subseries Lecture notes in artificial intelligence and Lecture notes in bioinformatics), LNCS, vol 9604, pp 79–94. https://doi.org/10.1007/978-3-662-53357-4_6.
2. Bhargavan K et al (2016) Formal verification of smart contracts: short paper. PLAS 2016—Proceedings of 2016 ACM work programming language analysis for security, co-located with CCS 2016, pp 91–96. https://doi.org/10.1145/2993600.2993611
3. Rouhani S, Deters R (2019) Security, performance, and applications of smart contracts: a systematic survey. IEEE Access 7:50759–50779. https://doi.org/10.1109/ACCESS.2019.2911031
4. Lunardi RC, Nunes HC, da Silva Branco V, Lippert BH, Neu CV, Zorzo AF (2019) Performance and cost evaluation of smart contracts in collaborative health care environments (January 2020)
5. Aldweesh A, Alharby M, Solaiman E, Van Moorsel A (2019) Performance benchmarking of smart contracts to assess miner incentives in Ethereum. In: Proceedings of 2018 14th European dependable computing conference, EDCC 2018, June 2019, pp 144–149. https://doi.org/10.1109/EDCC.2018.00034
6. Kosba A, Miller A, Shi E, Wen Z, Papamanthou C (2016) Hawk: the Blockchain model of cryptography and privacy-preserving smart contracts. In: Proceedings of 2016 IEEE symposium security privacy, SP 2016, pp 839–858. https://doi.org/10.1109/SP.2016.55
7. Watanabe H, Fujimura S, Nakadaira A, Miyazaki Y, Akutsu A, Kishigami JJ (2016) Blockchain contract: a complete consensus using blockchain. In: 2015 IEEE 4th global conference on consumer electronics, GCCE 2015, pp 577–578. https://doi.org/10.1109/GCCE.2015.7398721
8. Marino B, Juels A (2016) Setting standards for altering and undoing smart contracts. In: Lecture notes in computer science (including subseries Lecture notes in artificial intelligence. Lecture notes bioinformatics), vol 9718, pp 151–166. https://doi.org/10.1007/978-3-319-42019-6_10
9. Natoli C, Gramoli V (2016) The Blockchain anomaly. In: 2016 IEEE 15th international symposium on network computing and applications (NCA), October 2016, pp 310–317. https://doi.org/10.1109/NCA.2016.7778635
10. Fernandez C, Hickmott S, Norta A (2020) Tokenizing commercial property with smart contracts
11. Vukolić M (2017) Rethinking permissioned blockchains [Extended Abstract]. IBM Res 3–7. https://doi.org/10.1145/3055518.3055526
12. Rimba P, Tran AB, Weber I, Staples M, Ponomarev A, Xu X (2017) Comparing blockchain and cloud services for business process execution. In: 2017 IEEE international conference on software architecture (ICSA), April 2017, Section III, pp 257–260. https://doi.org/10.1109/ICSA.2017.44
13. Tian Y, Lu Z, Adriaens P, Minchin RE, Caithness A, Woo J (2020) Finance infrastructure through blockchain-based tokenization. Front Eng Manag. https://doi.org/10.1007/s42524-020-0140-2