Best Practices on Improving Gas Consumption through Simplifying Quality Complexity of Solidity code for Smart Contracts in Distributed Network Environments

Janghwan Kim
Software Engineering Laboratory
Hongik University
Sejong, Korea
lentoconstante@g.hongik.ac.kr

Chan Sol Park
Software Engineering Laboratory
Hongik University
Sejong, Korea
chansol53@mail.hongik.ac.kr

So Young Moon
Software Engineering Laboratory
Hongik University
Sejong, Korea
whit2@hongik.ac.kr

R. Youngchul Kim
Software Engineering Laboratory
Hongik University
Sejong, Korea
bob@hongik.ac.kr

Abstract— Recently, power consumption is exponentially increasing due to the huge execution of smart contracts on distributed networks. Every time a smart contract with complex code is executed, performance is degraded, and cost is high. To solve this problem, we propose making the Code complexity simplification for spending low power consumption. Through this, we expect to improve the quality of the code running in the distributed network environment.

Therefore, we propose a method to reduce the relationship between code complexity and gas consumption.

Keywords—Ethereum, Code Visualization, OOP Quality Metrics, C.K Metrics

I. Introduction

With the emergence of Non-fungible Token (NFT) through the development of blockchain technology, the use of distributed network-based applications is increasing day by day. NFT refers to the only token on a distributed network that differentiates itself from other copies by inserting unique codes into digital assets[1]. As the value of scarcity increases due to the development of these functions, the purchasing power of works to which NFT technology is applied is also rising. For this reason, the use of distributed network platforms for registered NFT transactions is also increasing as content creators use the smart contract function that supports NFT to digital assets to issue the digital assets as NFT. This tells that smart contact usage running on the blockchain network also increases. However, the Solidity language, which implements smart contracts on the Ethereum platform, has a lot of research on security aspects, but research is still insufficient in terms of performance and efficiency.

Depending on the type of smart contract and the degree of execution, the amount of resources consumed when the code on the smart contract is executed varies, and the gas cost is paid to the node that provided the resource[2]. Therefore, it is also necessary to study the performance and efficiency of the executed source code because the cost is involved when the smart contract is executed in a distributed environment.

In this paper, we propose a method to reduce gas consumption by reducing the source code complexity through comparing the complexity relationship between the amount of gas consumed in a distributed network and Object-Oriented Programing(OOP) based Quality metrics. Through this study, it is expected to reduce the complexity of the source code to reduce resource usage when executing smart contracts in a distributed network and to perform the same function at a lower cost.

Chapter 2 refers to research on object-oriented software visualization and research on Ethereum yellow paper as related studies. Chapter 3 discusses the relationship between object-oriented quality indicators and gas output, and Chapter 4 discusses conclusions and future research.

II. Relate works

A. *Ethereum Yellow paper*

Ethereum Yellow Paper is an Ethereum design manual produced by Gavin Wood. He explains the principles of the Ethereum virtual environment in this document. In this study, the design of the Ethereum platform, possible implementation problems, and possible obstacles are discussed[3].

In the Ethereum network, gas is a cryptocurrency that rewards contributors according to their contribution to execution in return for the resources that will be used to run the source code. The table below summarizes the gas cost for bytecode.

Table 1. Fee Schedule of Smart Contract Operation[3]

Operation	*Gas*	*Description*	*Fee Name*
ADD/SUB	3	Arithmetic operation	Gverylow
AND/OR/XOR	3	Bitwise logic operation	Gverylow

Operation	*Gas*	*Description*	*Fee Name*
…	…	…	…
MUL/DIV	5	Arithmetic operation	Glow
CREATE	32000	Create a new account using CREATE	Gcreate
CALL	25000	Create a new account using CALL	Gnewaccount

III. Design

The complexity and gas consumption are calculated quantitatively through object-oriented quality indicators and gas consumption calculators. Solidity is a contract-oriented language and its properties are similar to object-oriented programming [4]. Accordingly, based on the object-oriented quality index of the Solidity code, the complexity and gas consumption of the source code are analyzed to show the correlation between the two.

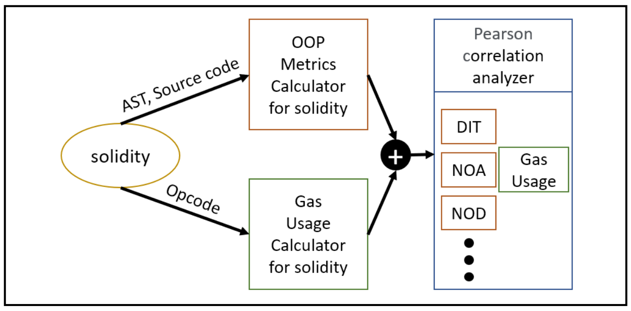
solidity
AST, Source code
Opcode
OOP Metrics Calculator for solidity
Gas Usage Calculator for solidity
+
Pearson correlation analyzer
DIT
NOA
NOD
Gas Usage

Figure 1. Solidity Complexity Relationship Analysis Process

Figure 1 shows the method of extracting the relationship between object-oriented quality indicators and gas consumption of Solidity code. The C.K matrix is used as an object-oriented complexity quality metrics. We calculate the complexity by analyzing the AST structure of the source code using the software visualization technique. According to Solidity Yellow Paper, the gas consumption is measured through the opcode operator[3]. The relationship between the complexity quality index and gas consumption of object-oriented programs is found through the Pearson correlation coefficient.

Table 2. Correlation Result Between OOP Complexity and Gas Consumption

Quality Indicator	*Pearson Coefficient*	*P-value*
DIT	0.719	0.00
NOA	0.731	0.00
NOD	0.731	0.00
Avg. McCC	0.726	0.00
Avg. NOS	0.743	0.00
Avg. NOI	0.791	0.00

Table 2 and Figure 2 shows the correlation graph result between Object oriented programming complexity and gas consumption. As we shown on Table 2, there is a result of measuring a total of 21 indicators, 14 of the 21 object-oriented quality indicators had a Pearson correlation coefficient of 0.5 or higher, and 6 of these indicators were 0.7 or higher.

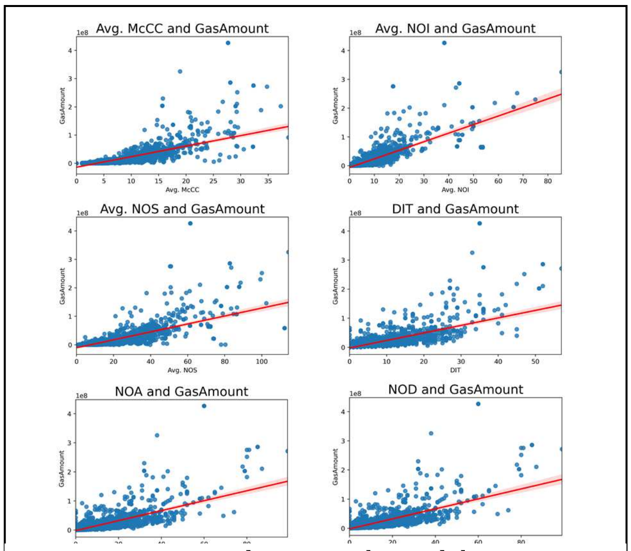

Figure 2. Correlation Graph Result between OOP Complexity and Gas Consumption

These six quality indicators (McCC: McCabe Cyclomatic Complexity, DIT: Depth of inheritance tree, NOA: number of ancestors, NOD: number of descendants, NOS: number of statements, NOI: number of outgoing invocations) indicate that they are closely related to gas output. Therefore, when the Pearson correlation coefficient is close to 1, the two indicators are strongly related. Also, the p-value is less than 0.001, so the result is a significant.

IV. Conclusion

In this paper, we measure the relationship between the amount of gas consumed in a distributed network and the complexity between the OOP-based quality metric. However, there is a limit to applying all quality indicators to the Solidity language in a limited environment. In the future, we will conduct research on reducing the gas consumption of contracts with the same function.

Acknowledgment

The research was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (2021R1I1A305040711) and the Basic Science Research Program through the National Research Foundation of Korea(NRF) funded by the Ministry of Education(2021R1I1A1A01044060) and the BK21 FOUR (Fostering Outstanding Universities for Research) funded by the Ministry of Education (MOE, Korea) (No.2021R1I1A305040711).

Reference

[1] Ante, Lennart. "Non-fungible token (NFT) markets on the Ethereum blockchain: Temporal development, cointegration and interrelations." Available at SSRN 3904683 (2021).

[2] Baird, Kirk, et al. "The economics of smart contracts." arXiv preprint arXiv:1910.11143 (2019).

[3] Wood, Gavin. "Ethereum: A secure decentralised generalised transaction ledger." Ethereum project yellow paper 151.2014 (2014): 1-32.

[4] Hegedűs, P. Towards Analyzing the Complexity Landscape of Solidity Based Ethereum Smart Contracts. Technologies 2019, 7, 6. https://doi.org/10.3390/technologies7010006