Off-chain Data Fetching Architecture for Ethereum Smart Contract

Xiaolong Liu
College of Computer and Information Sciences
Fujian Agriculture and Forestry University
Fuzhou 350002, China
E-mail: xlliu@fafu.edu.cn

Riqing Chen
College of Computer and Information Sciences
Fujian Agriculture and Forestry University
Fuzhou 350002, China
E-mail: riqing.chen@fafu.edu.cn

Yu-Wen Chen
Dept. of Computer Science
National Chiao Tung University
Hsinchu, Taiwan, ROC
E-mail: w369gf523@gmail.com

Shyan-Ming Yuan*
Dept. of Computer Science
National Chiao Tung University
Hsinchu, Taiwan, ROC
E-mail: smyuan@cs.nctu.edu.tw

***Abstract*—Ethereum smart contract makes developers can deploy decentralized applications to inherit features from blockchain, such as decentralization and openness. Although Ethereum provided a decentralized platform, Ethereum Virtual Machine for smart contracts, it lacks of ability to fetch off-chain data. The general solution is Oracle data carrier. However, Oracle results in rising deployment costs. It requires Ethereum smart contract developers to follow format in programing contract, this constraint decreases the readability of smart contract. This paper proposes an off-chain data fetching architecture which is cost-effective and highly elastic for smart contract. It also compatible with exited contract, which makes Ethereum smart contract owner able to automate the reply process.**

Keywords—Blockchain; Ethereum; Smart contract; Oracle.

I. INTRODUCTION

In recent years, blockchain technology has developed rapidly. The most representative application was Bitcoin proposed by Satoshi Nakamoto in 2008 [1], which is a peer-to-peer electronic cash system and a distributed ledger base on blockchain. It eliminates the need for trusted third party for e-commerce payment system. In 2013, blockchain developers came up with the second-generation blockchain application, Ethereum [2], which contains more feature than Bitcoin. Ethereum not only provides a ledger system but also provides the implementation of smart contract. Although there have been a lot of blockchain provides function of smart contracts in recent years, most of them are altcoin, and the foundation is still Ethereum.

The concept of smart contract gives blockchain the ability to do simply computation. However, smart contracts live like in a walled garden, they cannot fetch external data and generate random number on its own. This is due to the computation in Ethereum Virtual Machine should be determined. Despite smart contract provides computation ability, every transaction should be able to verify. In other words, every transaction processed by different Ethereum Virtual Machine spreading in same blockchain should be the same result, fetching off-chain data is not determined, generating random number is not either. This feature highly limits the developing of decentralized application [3]. Practically, smart contract developers have to setup an agent to get desired data, after getting data off-chain, calling the contract function to pass data back to the contract.

The object of this paper is to propose an architecture of data carrier for Ethereum smart contracts that increases little deployment costs, and monitor contract event without subscribing any filter at Ethereum node. The proposed architecture would not depend on Ethereum node to monitor events, and the data source is not limited. It is responsible for the interactions of contract developer register, monitor smart contract, Ethereum node callback and fetch of external data source and computation source. We also proposed selective solutions for filtering smart contract event, decoding event log to fit different requirements. The comparison result with Oracle in terms of deployment cost is presented to show the superiority of the proposed data carrier system.

The remainder of this paper is organized as follows. Section 2 presents the related work of this paper. The design of the proposed data carrier system is described in Section 3. Section 4 presents and discusses the evaluation results. Finally, Section 5 draws conclusions.

II. RELATED WORK

Ethereum [2] was proposed in late 2013 which is an opensource, public, blockchain-based distributed computing platform featuring smart contract, while Ether is the cryptocurrency used on this platform. The intent of Ethereum is to create an alternative protocol for building decentralized applications. The concept of smart contract is first proposed by Nick Szabo in 1997 [5], but it lack of platform to implement until blockchain was proposed. Smart contract should not be seen as something that should be fulfilled or complied with, rather, smart contracts are more like autonomous agents. A smart contract is a set of commitments that are defined in digital form, including the agreement on how contract participants shall fulfill these commitments. Generally, Ethereum based smart contract is programed by Solidity, which is a contract-oriented, high-level language. It was influenced by C++, Python and JavaScript and is designed to target the Ethereum Virtual Machine. One issue related to using smart contracts on a public blockchain is that bugs, including security holes, are visible to all but cannot be fixed quickly. Smart contracts such as casinos require random numbers, decentralized exchanges, and exchange rate information, which require Oracle help to obtain this information.

Fig.1 shows the conceptual architecture of Oracle [4], the concept of it is to enable smart contract fetch off-chain data. Although Oracle has a variety of different ways to implement, in this paper the architecture based on the version issued by commercial company, Oraclize[6], was used. Oraclize provide part of the infrastructure needed to build smart and useful decentralized applications, and its service guarantee the correctness of data [7].

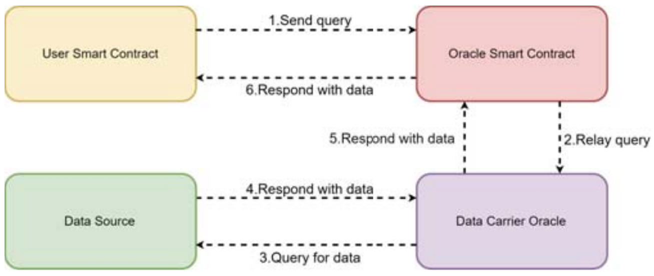

Fig. 1. The architecture of Oracle

The first benefit of Oracle is that if you have multiple contracts which needs external data, traditionally, you may program responder, launching one responder for one smart contract, but if you take the architecture of Oracle, the only event emitted by contract which needs off-chain data would be Oracle contract, this makes Oracle become the agent of all contracts needing off-chain data. The second benefit is that Oracle does not need manage contract's application binary interface. In general, anyone wants to interact with specific contract, he need two elements, i.e. contract address and application binary interface. However Oracle user do not need to provide any application binary interface for Oracle provider. Because the Oracle provided by Oraclize contains a virtual function used for callback, user needs to inherent standard callback function to receive external data.

However, the feature that Oracle does not need application binary interface actually is a double-edged sword, its shortcoming is everyone can easily decode your event, even trigger your callback function when contract programmer does not limit the message sender of callback function Appropriately, although the purpose of application binary interface does not encrypt the transaction, it still increases the risk of smart contract [8].

III. System design

The object of this paper is to propose a smart contract data carrier architecture that is cost-effective, highly flexible and user friendly. This section introduces the overview of the proposed architecture and detail of each component. Fig. 2 shows the interactions of the proposed data carrier system, it is responsible for the interactions of contract developer register, monitor smart contract, Ethereum node callback and fetch of external data source and computation source. Fig. 3 shows the conceptual architecture of the proposed data carrier system. Basically, it contains three components: Mission Manager, Task Publisher and Worker.

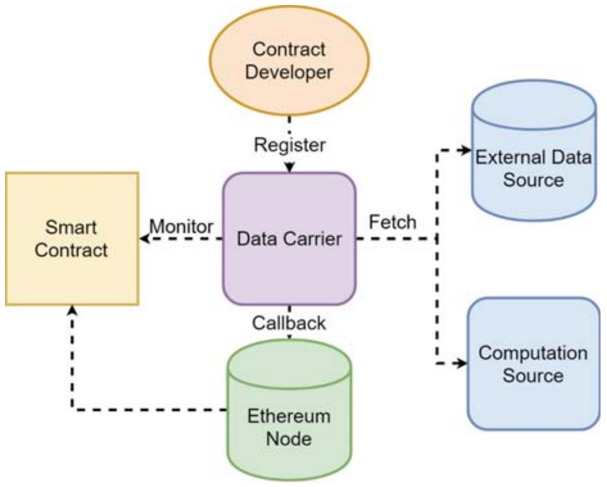

Fig. 2. The interactions of the proposed data carrier

Mission Manager: mission manager is used to receive mission registered by system user, a mission contains event hash and contract address, how to response the event, and the queue channel response for event. In addition database was used to store missions, which provide the necessary information for monitoring Ethereum blockchain, how to send external data back to the smart contract, and how does worker retrieve the external data. A popular web framework Express, which is written in JavaScript and hosted within the node.js runtime environment, is used to build service back-end. We use express to set up a RESTful API for users to register mission, and the document oriented database MongoDB to store the mission.

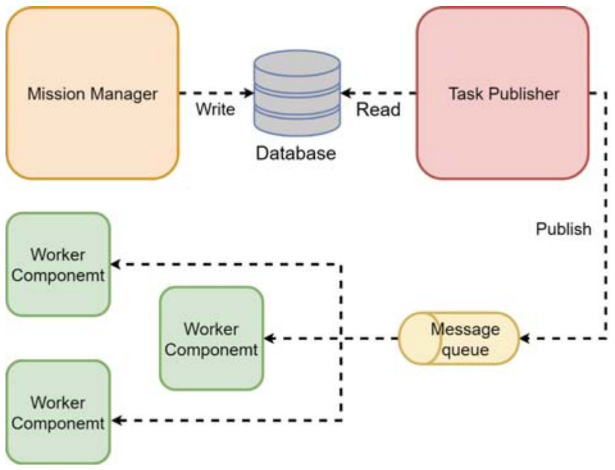

Fig. 3. The architecture of the proposed data carrier

Task Publisher: task publisher will perform four phases action for each pended block: filter out unconcerned transaction, fetch argument in event, send generated task to specific work, and publisher collect transactions on new block. Task publisher is implemented by Node.js with the characteristic of event-driven and non-blocking I/O model. As shown in Fig.4, There are three modules in task publisher: Filter module is triggered when every new block header comes in to find out whether managed Ethereum smart contract is activated by any address. When user call smart contract function these event will be document in the log entry. Decoding module will get the arguments in the event log, decode the event log and replace the command template to generate task for worker. After replacing the command, publishing module will push the task to the message queue.

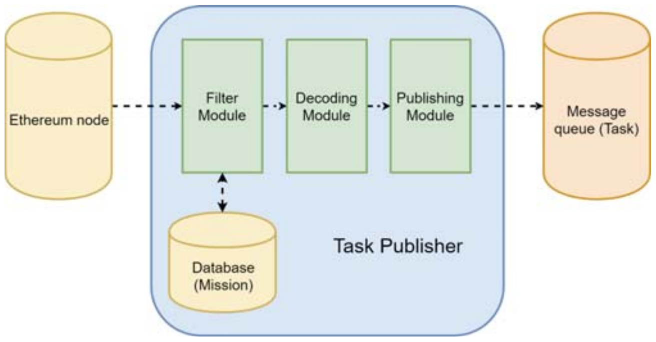

Fig. 4. The architecture of Task Publisher

Worker: worker executes receiving command to obtain data, which could be computation or simply fetch data. In general worker consists of execution module and transaction module. The execution module uses the child_process package of Node.js to generate an external execution program which is specified by task entry "command". This program can be fetching agent or computation agent, and both scenarios needs to output the parameters of smart contract to standard output. The worker will obtain this output for the transaction module as callback parameter. The transaction module is responsible for passing the results generated by the execution module back to the smart contract via function calls.

IV. Evaluation Results

The difference between the proposed data carrier architecture in this paper and the Oracle system is that the deployment costs when deploying smart contracts are different. The example contract is KrakenPriceTicker.sol [9], here we use Remix, Solidity IDE to evaluate deployment cost. KrakenPriceTicker is a smart contract which fetch Bitcoin price at digital asset trading platform, i.e. Kraken. Table 1 and 2 shows the original contract of KrakenPriceTicker cost about 400,000 gas, on the other hand, Oracle contract cost for it is about 1.8million gas after optimization. From table 8 and 9, we can also find that the cost of Oracle contract may even be several times higher than the original smart contract. This is because Oracle provides a lot of additional features. At the time of the inherit Oracle resolver contract, these functions were still inherited, which resulting in a very large storage consume.

Table 1 Deployment Cost - KrakenPriceTicker.sol

Enable compiler Optimization	Deployment Cost (gas)
No	433,800
Yes	393,000

Table 2 Deployment Cost - OraclizeLib.sol

Enable compiler Optimization	Deployment Cost (gas)
No	2,563,800
Yes	1,719,200

The formula for calculating the cost of a smart contract is shown as (1):

$$\text{Deployment Cost} = \text{Gas Used} * \text{Gas Price} \qquad (1)$$

where Gas Used was decided after compiling the smart contract (more specifically, it was decided during the deployment). When deploying smart contracts, the cost mainly comes from the size of the original data, the space occupied by the smart contract after deployment, and the constructor operating costs. Gas price refers to the amount of Ether you're willing to pay for every unit of gas, and is usually measured in "Gwei", which means ten to the negative ninth power Ether.

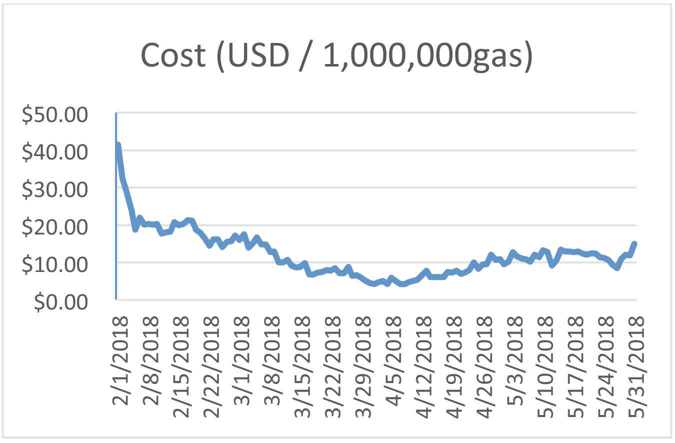

Fig. 5. Ethereum Gas Price per Million Gas

In this evaluation, we use data of Ethereum gas price for per Million Gas from February to May 2018 [10, 11], which was shown in Fig.5, to evaluate the development cost of a smart contract. After calculation, the result in Fig.5 shows the deployment cost for every one million gas cost will be about 11.5 US Dollar. Since Oracle contract takes about 1.8million gas as shown in table 2, we can extrapolate that using our architecture can save about $20 deployment cost in average for a smart contract.

V. Conclusions

In this paper, the architecture of data carrier for Ethereum smart contract is proposed. This work is trying to solve problems causes by Oracle style data carrier, such as expensive deployment, lack of compatibility. We also propose a deployment to solve consensus issue caused by fetching off-chain data. In the evaluation, we evaluated how much deployment cost can be save by the average data from February 2018 to May 2018. Generally, by using our architecture, it will decrease about 20 USD for every smart contract who need data carrier service. In this work, while making consensus on external data, we push external data to the smart contract to do processing.

VI. ACKNOWLEDGEMENT

This work was supported by the fund of the National Natural Science Foundation of China (Grant No. 61702102), Natural Science Foundation of Fujian Province, China (Grant No. 2018J05100), and MOST of Taiwan (Grant No.105-2511-S-009-007-MY3).

REFERENCES

[1] NAKAMOTO, Satoshi, “Bitcoin: A peer-to-peer electronic cash system,” 2008. Available: https://bitcoin.org/bitcoin.pdf

[2] BUTERIN, Vitalik, et al, “A next-generation smart contract and decentralized application platform,” white paper, 2014. Available: https://github.com/ethereum/wiki/wiki/White-Paper

[3] Kieron, O. (2017). Smart contracts - dumb idea. IEEE Internet Computing, 21(2), 97-101.

[4] ELLIS, Steve, “A Decentralized Oracle Network Steve Ellis, Ari Juels, and Sergey Nazarov.” Available: https://icowhitepapers.co/wp-content/uploads/ChainLink-Whitepaper.pdf

[5] Szabo N. Formalizing and Securing Relationships on Public Networks[J]. 1997, 2(9).

[6] Oraclize, API documentation. Available: http://docs.oraclize.it/

[7] MERZDOVNIK, Georg, et al, “Whom you gonna trust? a longitudinal study on TLS notary services,” IFIP Annual Conference on Data and Applications Security and Privacy. Springer, Cham,. pp. 331-346, 2016.

[8] ATZEI, Nicola; “BARTOLETTI, Massimo; CIMOLI, Tiziana. A survey of attacks on Ethereum smart contracts (SoK),” International Conference on Principles of Security and Trust, Springer, Berlin, Heidelberg, pp. 164-186, 2017.

[9] Oraclize, KrakenPriceTicker.sol V0.4.25. Available: https://dapps.oraclize.it/browser-solidity/#version=soljson-v0.4.19+commit.c4cbbb05.js&optimize=false&gist=ad3d1f6007942b727f5909b55e6445d2

[10] Etherscan The Ethereum Block Explorer. Ether Historical Prices, Retrieved June 6, 2018, Available: https://etherscan.io/chart/etherprice

[11] Etherscan The Ethereum Block Explorer. Transaction Fees , Retrieved June 6, 2018, Available: https://etherscan.io/chart/transactionfee

[12] POON, Joseph; DRYJA, Thaddeus. The bitcoin lightning network: Scalable off-chain instant payments. draft version 0.5, 2016, 9: 14. Available: http://coinshp.com/assets/pdf/lightning.pdf