Query Support for Data Processing and Analysis on Ethereum Blockchain

Fariz Azmi Pratama
School of Electrical Engineering and Informatics
Institut Teknologi Bandung
Bandung, Indonesia
farizazmip@gmail.com

Kusprasapta Mutijarsa
School of Electrical Engineering and Informatics
Institut Teknologi Bandung
Bandung, Indonesia
soni@stei.itb.ac.id

***Abstract*—Blockchain technology has gained immense popularity because many researchers believe that it could solve numerous problems and could be applied in various fields of study. Unfortunately, behind its potentials, blockchain also possessed many challenges and limitations. The highlighted problem is the usability aspect of blockchain technology examined from developer and user perspective. This paper tried to address this problem by proposing query functionalities, with the help of query layer system, to facilitate the developer and the user to access blockchain data easily. There are three main query functionalities that will be discussed in this paper: (1) finding blockchain data based on multiple search parameters (retrieval query), (2) providing simple statistical analysis from a collection of blockchain data (aggregate query) and (3) sorting blockchain data according to its blockchain component (ranking query). For the implementation stage, Ethereum is used as platform to provide blockchain network, MongoDB is used as cloud storage service and REST API is used as web services. For the testing stage, throughput and time response are used to evaluate the performance of the developed query functionalities in the query layer system. The results are: (1) the throughput of query layer system is lower than Ethereum service for blockchain data retrieval and (2) the time response of query layer system is affected by the number of thread and the amount of data stored in cloud storage.**

***Index Terms*—blockchain, query, Ethereum, cloud storage.**

I. Introduction

Blockchain technology was first introduced by an author under pseudonym Satoshi Nakomoto in his paper "Bitcoin: A Peer-to-Peer Electronic Cash System". Although the paper was published in 2009, the Bitcoin itself was implemented in 2008 [1]. Simply put, blockchain is a digital ledger that is shared among running nodes on the cryptocurrency network. There are various techniques involved when building a blockchain technology, such as cryptography, cloud computing, distributed system, peer-to-peer network and consensus mechanism. Currently, some of the most popular platform that utilized blockchain technology are Bitcoin and Ethereum [2].

At first, the purpose of blockchain technology is to facilitate cryptocurrency transaction. But after blockchain has been claimed to be a solution to many problems such as performance, privacy, security and extensibility [2], many researchers from various field begin to explore the potential of blockchain technology. There are several researches that discuss about blockchain opportunities and innovations [11], [12]. Zheng et al. [12] explain that blockchain can be utilized for (1) finance, (2) Internet of Things (IoT), (3) public and social services, (4) reputation system, and (5) security and privacy. Those usages can be applied into numerous fields, such as finance and insurance, energy production, mobility and logistics.

Even though blockchain promised many opportunities, there are also several researches that discuss about blockchain challenges and limitations. Yii-Huumo et al. [3] classify these problems into seven main categories: (1) throughput, (2) latency, (3) size and bandwidth, (4) security, (5) wasted resources, (6) usability and (7) versioning, hard forks, multiple chains. This paper aims to solve the usability aspect of blockchain technology. From developer's perspective, usability is defined as the ease to use web services to provide blockchain data for another applications. This problem firstly highlighted by Swan [4] based on the fact that the Bitcoin API is not developer friendly. It is difficult to implement and to be used with or in other applications. From user's perspective, usability of blockchain can be defined as the ability to accessing, including processing and analyzing, blockchain data easily. Therefore, this challenge motivate us to create query functionalities that capable to facilitate user to access blockchain data easily through a web service.

Related Work. The expected product of our proposed solution resembles the blockchain explorer. Blockchain explorer could be defined as an online service that allows user to view information related to blockchain data, such as account, transaction and block. Etherscan is one of the most popular blockchain explorer for Ethereum blockchain. Etherscan provides web service with minimalistic functions for user to search blockchain data through REST API. Besides blockchain explorer, a research by Li et al. [5] proposed a design of query layer for public blockchain. This design provides simple analytic queries that can be used for blockchain data management.

Our Contribution. This paper aims to enhance and expand the query functionalities defined in paper [5] to further improve the usability of blockchain in both user and developer perspective. First, we design blockchain data re-

978-1-5386-6670-8/18/$31.00 2018 IEEE

trieval to support multiple search parameters. The purpose of multiple search parameters is to minimize the effort to search data based on many criteria. Second, we propose some query functionalities to the system based on analytic functions defined in [10]. This feature will enrich the general capability of our query layer system. To minimize the scope of blockchain terminology, we will pick Ethereum as our blockchain platform.

Organization. The rest of this paper is structured into four sections. Section II will explain about definition and blockchain components in Ethereum data model. Then in section III, we introduce the system that will be adopted into our system. The detail of the proposed query functionality that will be integrated in our query layer system is described in section IV. After that, the explanation of the implementation and the testing for our system will be explained in section V. Finally, section VI will cover the conclusions of our work.

II. Ethereum Data Model

Ethereum blockchain data is composed of three technical information: account, transaction and block [6]–[8]. Each of those technical information may consist of several components that can be represented either by using numeric value or hexadecimal string.

A. Account

In Bitcoin, the world state is represented by a collection of Unspent Transaction Output (UTXO). By using UTXO as an input, users can create a transaction on Bitcoin network. Meanwhile in Ethereum, the world state is represented by an account. Every account possessed a unique 160-bit identifier called an address. There are two different types of account, namely externally owned account (EOA) and contract account. The difference between both of them can be found on the ownership. While EOA may be controlled by a person, contract account may not. Besides that, both of them contain same pieces of component, as shown in TABLE I.

TABLE I
Account Component on Ethereum Blockchain

No.	Name	Data Representation
1	balance	hexadecimal string
2	address	numeric value

B. Transaction

Transaction data is used to store the detail of transaction activity that occurred on Ethereum network. Transaction activity is formally defined as a message sending from one account to another. Transaction is created and digitally signed by an external actor, presumed to be a human. There are two different types of transaction, namely contract creation transaction and message call transaction. The former is used to deploy a new contract while the latter is used to utilize an existing contract. Besides that, both of them contain same pieces of component, as shown in TABLE II.

TABLE II
Transaction Component on Ethereum Blockchain

No.	Name	Data Representation
1	blockHash	hexadecimal string
2	blockNumber	hexadecimal string
3	from	hexadecimal string
4	hash	hexadecimal string
5	input	hexadecimal string
6	to	hexadecimal string
7	v, r, s	hexadecimal string
8	gas	numeric value
9	gasPrice	numeric value
10	nonce	numeric value
11	transactionIndex	numeric value
12	value	numeric value

C. Block

In Ethereum, block consists of two types of information, namely block header and transaction. Block header is defined as metadata that provide some information related to block creation (e.g., timestamp or miner information). Table III shows the component that is contained in block header.

TABLE III
Block Header Component on Ethereum Blockchain

No.	Name	Data Representation
1	extraData	hexadecimal string
2	hash	hexadecimal string
3	logsBloom	hexadecimal string
4	miner	hexadecimal string
5	mixHash	hexadecimal string
6	parentHash	hexadecimal string
7	receiptsRoot	hexadecimal string
8	sha3Uncles	hexadecimal string
9	stateRoot	hexadecimal string
10	totalDifficulty	numeric value
11	difficulty	numeric value
12	gasLimit	numeric value
13	gasUsed	numeric value
14	nonce	numeric value
15	number	numeric value
16	size	numeric value
17	timestamp	numeric value

III. Query Layer System

The query layer system described in this paper is adopted from [5] and built on the application layer of the blockchain. The system consist of four modules which the relationship can be shown in Fig. 1. The function of each module can be described as follows.

Sync module. Since on average the block in Ethereum is generated for every 14-30 seconds, there is a need to automatically detect and catch that new block. This module existed to fulfill those demand. Sync module worked by listening on Ethereum network for a new block, making our system capable to maintain with the generation blockchain data.

Handler module. The purpose of this module is to extract the information of the block, which are the account, the

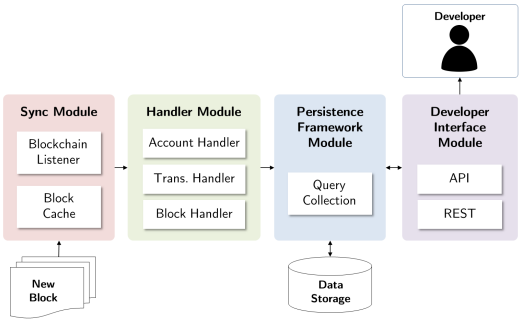

Fig. 1. The design of query layer system.

transaction and the block header. This extraction is needed to transform blockchain data into database objects.

Persistence framework module. This module is a middleware that facilitate the system to retrieve/store data to/from the data storage. It is contained a collection of query functionalities that the query layer system provides. The query functionalities will be described in section IV.

Developer interface module. The service to provide query interface is implemented using RESTful API. The purpose of developer interface module is to design our system to support DaaS (Data as a Service). This service will make the query layer capable to serve applications on top of blockchain platform.

IV. Proposed Query

The query that is used in our system consist of extended implementation of retrieval query in [5] and some analytic queries adopted from [10]. Since the system is utilizing MongoDB environment, the returned data will be in rich BSON data types. Therefore our system needs to convert this data types into JSON before making the result available for user to use.

1) Retrieval Query
   - **Specification.** The purpose of this query is to get data based on the search parameter described in the input parameter. The returned data could be accounts, blocks, and transactions. Several improvements are made for this query compared to previous paper [5]. First, the query supports multiple search parameters; therefore the query is able to obtain the data that matched according to various conditions. Second, because we noticed there are some missing components in each of the accounts, blocks and transactions in the implementation of previous paper, our paper tried to perfect those incompleteness. Therefore, the query will provide full information on the returned data.
   - **Input parameter.** The searching can be done in two manners. First, we can do the search by comparing the parameter to an exact matching string. This can be achieved because there are several unique components in some data, such as hash and miner in block, or hash, to and from in transaction. Second, we can do the search by giving range value for some parameter. This also can be achieved because numeric values are used in some fields, such as difficulty and number in block, or gas, gasPrice and value in transactions.
   - **Output parameter.** The returned data could be account, block or transaction that fulfill the requirement in the input parameter.
2) Aggregate Query
   - **Specification.** The purpose of this query is to give analysis of the data based on aggregate functions. In this paper, we use five functions to realize this query, namely count, max, min, sum and avg. This query is useful for statistical purpose.
     a) Count function is used to count the number of returned data based on the result of the search using input parameters.
     b) Max and min function is used to get the maximumand minimum value of the referenced components based on the conditions given in input parameters respectively. This can be used for value, gasPrice and gasLimit component in transactions.
     c) Sum and avg function is used to get the total amount and average of value of the referenced components based on the conditions given in input parameters respectively. This can be used for value component in transactions.
   - **Input parameter.** Similar to the input parameter of retrieval query, but limited to the component with numeric value. The reason is that all of the aggregate functions worked by calculating numeric value.
   - **Output parameter.** The output will be single value that indicate the result of aggregate function.
3) Ranking Query
   - **Specification.** The purpose of this query is to give sorted data based on the determined criterion. The ordering can be conducted in two method, ascending or descending. This query will be useful in determining the top-k of the data.
   - **Input parameter.** The parameter is similar to aggregate query, but not limited to the component with numeric value.
   - **Output parameter.** An ordered list of account, block or transaction.

V. Implementation and Testing

The implementation and testing stage is conducted on Asus A455L with Intel Core i3-4005U 1.7GHz, RAM 6 GB DDR, HDD 500 GB and Windows 8.1 Pro as operating system.

A. Implementation Detail

The implementation of our query layer will be explained per module that has been described in section III.

Sync module. In the implementation, we use private blockchain based on Ethereum platform. Geth v1.8.3 is used as command line and web3.py v4.6.0 is used as interface to interact with the platform. The sync module is implemented by calling filtering class to listen for events asynchronously.

Handler module. For handler module, we use Python hex and dict function to convert raw blockchain data that the system received from sync module. After converting the data, we can extract the transaction data and block header data. For account data, we can only get the address component from transaction data (described by from and to component). Then to complete the account data, we can use library web3py to retrieve the balance from a given address.

Persistence framework module. MongoDB is used as cloud storage to manage the blockchain data. There are three different types of collection created based on handler module mapping: account collection, transaction collection and block collection. MongoDB shell v3.6.5 is used to connect with MongoDB server. To implement the query functionality, we utilized library pymongo that provides many queries that can be used to interact with the stored blockchain data.

Developer interface module. Flask v0.1.12 is a web application framework that provides the REST API in our system. There are 21 API endpoints that being implemented and can be accessed by user or developer as shown in TABLE IV.

TABLE IV
IMPLEMENTED API ENDPOINT

No.	API Endpoint	Argument
1	/api/select/account	Component TABLE I
2	/api/select/transaction	Component TABLE II
3	/api/select/block	Component TABLE III
4	/api/aggregate/count/account	Component TABLE I, target
5	/api/aggregate/max/account	Component TABLE I, target
6	/api/aggregate/min/account	Component TABLE I, target
7	/api/aggregate/sum/account	Component TABLE I, target
8	/api/aggregate/avg/account	Component TABLE I, target
9	/api/aggregate/count/transaction	Component TABLE II, target
10	/api/aggregate/max/transaction	Component TABLE II, target
11	/api/aggregate/min/transaction	Component TABLE II, target
12	/api/aggregate/sum/transaction	Component TABLE II, target
13	/api/aggregate/avg/transaction	Component TABLE II, target
14	/api/aggregate/count/block	Component TABLE III, target
15	/api/aggregate/max/block	Component TABLE III, target
16	/api/aggregate/min/block	Component TABLE III, target
17	/api/aggregate/sum/block	Component TABLE III, target
18	/api/aggregate/avg/block	Component TABLE III, target
19	/api/rank/account	Component TABLE I, target, limit, mode
20	/api/rank/transaction	Component TABLE II, target, limit, mode
21	/api/rank/block	Component TABLE III, target, limit, mode

In chronological order, the REST resource identifier define the query functionality, the returned blockchain data and the argument. Below are some examples of API endpoint usage. Fig. 2 also shows the interface of accessing the API endpoint. It can be seen that the data returned in JSON format.

- /select/block?number=0x1 means calling a query functionality that returned block data with component number 0x1.
- /aggregate/max/block?number_min=0x1&target=gasUsed means calling a query functionality that will returned the maximum value of gasUsed for every block with component number higher than 0x1.
- /rank/block?limit=5&target=difficulty&mode=asc means calling a query functionality that will returned five block data sorted by component difficulty in ascending order.

Fig. 2. Interface of API endpoint

From TABLE IV, we see various arguments that can be used when calling the API endpoint. In retrieval query, the argument is decided according to each data component defined in TABLE I, TABLE II or TABLE III. For component with numeric value representation, suffix _min or _max can be added to the argument to enable the ability to search data by range. For example, number_min and number_max imply block search for every number component in range [number_min, number_max].

In aggregate query, the argument that can be used is similar to the argument of retrieval query, but with the addition of target argument. Target is used for component with numeric value representation and its purpose is to determine the component that need to be analyzed. In rank query, the argument that can be used is similar to aggregate query, but with the addition of limit and mode argument. Since sorting activity is not affected by data representation, the target argument for rank query can be used by all types of component. Limit is used to cap the result after the data has been sorted. Mode is used to give an option for ordering method.

B. Testing

There are two types of testing conducted in this paper: (1) unit testing and (2) load testing. Both of the testing activity use JMeter, an application designed to load test functional behavior [13].

Unit testing. The purpose of this test is to ensure the query worked correctly. This testing can be done by checking the correctness for every predetermined input and its corresponding output to the query layer system. From the testing that has been conducted using every argument in each of API endpoint, we found that the developed query layer system produced the correct output for every given input.

Load testing. The purpose of this test is to evaluate the performance of our query layer system. The load testing is conducted in two stages. First, we compare the throughput of

our query layer system and Ethereum service for block data retrieval. Second, we measure the time response of our query layer system to grasp the performance in general.

For the first stage of load testing, the dataset is generated from mining activity of Ethereum private network. From 10000 generated block data, we achieved the result shown in TABLE V. The throughput is measured by modifying thread number and using 100 iterations of query calling.

TABLE V
THROUGHPUT OF QUERY LAYER SYSTEM AND ETHEREUM SERVICE

Thread Number	Maximum Time Response		Throughput	
	Query Layer	Ethereum	Query Layer	Ethereum
1	38 ms	15 ms	39.7 req/s	101.1 req/s
10	320 ms	101 ms	39.4 req/s	98.6 req/s
20	548 ms	213 ms	41.0 req/s	97.8 req/s
40	1050 ms	345 ms	41.3 req/s	99.7 req/s
80	1789 ms	583 ms	41.1 req/s	98.3 req/s

For the second stage of load testing, the dataset is gathered from Ethereum public network using Etherscan service. Starting from block with component number 5000000, we collect 100000 block data, 1500000 transaction data and 5000 account data. The result of second stage of the load testing is shown in TABLE VI. The time response measured in the test is maximum time response. The test also conducted using various thread number for each API endpoint. Since there are five different aggregate function, in this test we only use the avg function to represent aggregate query.

TABLE VI
TIME RESPONSE OF QUERY LAYER SYSTEM FOR VARIOUS THREADS

API Endpoint	Thread Number				
	1	10	20	40	80
	Maximum Time Response (ms)				
/api/select/account	38	335	547	1157	2146
/api/select/transaction	3189	30739	60481	-	-
/api/select/block	55	488	1246	1607	3132
/api/aggregate/avg/account	249	1475	2828	5539	10476
/api/aggregate/avg/trans.	3629	32925	67701	-	-
/api/aggregate/avg/block	1349	12137	24634	47450	95112
/api/rank/account	121	654	1335	2077	4486
/api/rank/transaction	4710	40435	82546	-	-
/api/rank/block	122	972	1730	3449	7333

From measuring throughput and time response of the query layer, we can conclude several characteristics of the system:

- The average throughput of the query layer system is 40.5 requests per second, while the average throughput of Ethereum service is 99.1 requests per second. Since there is an additional time caused by network latency in MongoDB server, then the low throughput of the system is within expectation.
- The time response of query layer system is affected by two factors: the number of thread and the amount of data stored in cloud storage. The increasing number of thread results in the rising of time response of the system, vice versa. The trend is also applied with the amount of data in cloud storage. The increasing amount of data in cloud storage (by comparing the test result in TABLE V and TABLE VI) results in the rising time response of the system, vice versa.

VI. CONCLUSION

In this paper, we develop various query functionalities as a solution for blockchain usability viewed on developer and user perspective. Access to these query functionalities is given by adopting query layer system proposed in [5]. Our system utilizes Ethereum as blockchain platform. The query layer system worked by synchronizing new blockchain data from Ethereum to cloud storage into three different collections: account, block and transaction. This data then can be accessed by developer or user through REST API. There are 21 API endpoints that could be accessed to use query functionalities. From the evaluation, the average throughput produced by (a) query layer system is 40.5 requests per second and (b) Ethereum service is 99.1 requests per second. Generally, the throughput of the query layer system is lower that Ethereum service. Then the time response of query layer system for each API endpoint is varied, but it's affected by two factors: the number of threads and the amount of data in cloud storage.

REFERENCES

[1] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2008. [Online]. Available: https://bitcoin.org/bitcoin.pdf

[2] D. Vujii, D. Jagodi and S. Rani, "Blockchain technology, bitcoin, and Ethereum: A brief overview," in Proceedings of 17th International Symposium INFOTEH-JAHORINA (INFOTEH), East Sarajevo, 2018, pp. 1-6.

[3] J. Yii-Huumo, D. Ko, S. Choi, S. Park and K. Smolander, "Where Is Current Research on Blockchain Technology? - A Systematic Review", PLoS ONE vol. 11, no. 10, pp. e0163477, 2016.

[4] M. Swan, Blockchain: Blueprint for a New Economy, O'Reilly Media, Inc., 2015.

[5] L. Yang, Z. Kai, Y. Ying, L. Qi and Z. Xiaofang, "EtherQL: A Query Layer for Blockchain System," *in Proceedings of 22nd International Conference on Database Systems for Advanced Applications (DASFAA)*, Suzhou, China, 2017, pp. 556-567.

[6] S. Tikhomirov, "Ethereum: state of knowledge and research perspectives," in Proceedings of 10th International Symposium on Foundations and Practice of Security, Nancy, Frane, 2017, pp. 206-221.

[7] G. Wood, "Ethereum: A secure decentralised generalised transaction ledger," 2015. [Online]. Available: http://gavwood.com/Paper.pdf

[8] A. Ellervee, R. Matulevicius and N. Mayer, "A Comprehensive Reference Model for Blockchain-based Distributed Ledger Technology" in Proceedings of the ER Forum and the ER Demo Track, Valencia, Spain, 2017, pp 320-333.

[9] Z. Zheng, S. Xie, H. Dai, X. Chen and H. Wang, "An Overview of Blockchain Technology: Architecture, Consensus, and Future Trends," in Proceedings IEEE International Congress on Big Data (BigData Congress), Honolulu, HI, USA, 2017, pp. 557-564.

[10] T. Kyte, Expert One-on-One Oracle. Apress, 2005.

[11] T. Ahram, A. Sargolzaei, S. Sargolzaei, J. Daniels and B. Amaba, "Blockchain technology innovations," in Proceedings IEEE Technology and Engineering Management Conference (TEMSCON), Santa Clara, CA, USA, 2017, pp. 137-141.

[12] Z. Zheng, S. Xie, H. Dai, X. Chen and H. Wang, "Blockchain technology innovations," International Journal of Web and Grid Services, vol. 13, no. 1, pp. 653-659, 2017.

[13] S. K. Chakrabarti and P. Kumar, "Test-the-REST: An Approach to Testing RESTful Web-Services," in Proceedings of Computation World: Future Computing, Service Computation, Cognitive, Adaptive, Content, Patterns, Athens, 2009, pp. 302-308.