2019 IEEE 4th International Workshops on Foundations and Applications of Self* Systems (FAS*W)

Gas Consumption-aware Dynamic Load Balancing in Ethereum Sharding Environments

Sanghyeok Kim, Jeho Song, Sangyeon Woo, Youngjae Kim and Sungyong Park
Department of Computer Science and Engineering
Sogang University, Seoul Korea
{sangh228, oidwin, tkddus121, youkim, parksy}@sogang.ac.kr

***Abstract*—Advances in blockchain technology have made a significant impact on a wide range of research areas due to the features such as transparency, decentralization and traceability. With the explosive growth of blockchain transactions, there has been a growing interest in improving the scalability of blockchain network. Sharding is one of the methods to solve this scalability problem by partitioning the network into several shards so that each shard can process the transactions in parallel. Ethereum places each transaction statically on a shard based on its account address without considering the complexity of the transaction or the load generated by the transaction. This causes the transaction load on each shard to be uneven, which makes the transaction throughput of the network decrease. In this paper, we propose a dynamic load balancing mechanism among Ethereum shards called D-GAS. The D-GAS dynamically balances the transaction load of each shard by relocating the accounts based on the gas consumption to maximize the transaction throughput. Ethereum gas is a unit that represents the amount of computational effort needed to execute operations in a transaction. Benchmarking results show that the D-GAS outperforms existing techniques by up to 12% in transaction throughput and decreases the makespan of transaction latency by about 74% under various conditions.**

Keywords— blockchain; ethereum; sharding; scalability; load balancing;

I. Introduction

Blockchain is a peer-to-peer (P2P) based distributed ledger technology that ensures integrity and reliability without an authorized third party's involvement. Although the blockchain was originally developed as part of Bitcoin [1], it has recently been drawing much attention as an innovative technology that can support a variety of fields such as health-care [2], internet of things (IOT) [3] or medical data management [4]. However, despite the worldwide interest in the blockchain, applying this technology to various areas is sometimes limited due to the scalability problem [5]. When the number of transactions increases, the transaction per second (TPS) of the blockchain decreases severely because the time for sharing a block or reaching a consensus is delayed. For example, when one of the most well-known Ethereum-based games called CryptoKitties [6] was released, the amount of pending transactions was sharply increased [7] and finally broke down the Ethereum network. While simply increasing the block size or shortening the interval to create a block can improve the transaction throughput, it may also weaken the network security as the newly created blocks cannot possibly be delivered to all the nodes in the blockchain network [8].

To solve this scalability problem effectively, several approaches have been proposed. Those include the mechanisms using off-chain payment [9], using byzantine fault tolerance (BFT) consensus instead of existing proof-of-work (PoW) algorithm [10], using a permissioned (or private) blockchain that requires only authorized clients to participate in the consensus process [11], and using sharding [12][13].

Sharding is a method to increase the transaction throughput of the network by partitioning blockchain into several pieces called shards and allowing each shard to process the transactions in parallel. This mitigates the amount of data transferred among the nodes as the size of data within the shard is smaller than that in the entire blockchain. Consequently, sharding can increase the transaction throughput. Whereas sharding seems to be a viable solution for scaling blockchain, it also creates other challenging issues that needs to be solved such as how to allocate each transaction to a specific shard, how to reach consensus between shards, how to access or synchronize the states in a shard with other shards, etc.

Among those important challenging issues, this paper attempts to address the problem of transaction allocation to a shard in Ethereum sharding environments. Ethereum [14] is a distributed, permissionless (or public) blockchain platform that can run smart contracts. Due to its decentralized, secure, and flexible nature, Ethereum has been widely used as a platform for initial coin offering (ICO). Moreover, the smart contracts allow Ethereum to be used in various applications other than digital currency such as insurance, IOT, authentication of certificate or identity, etc. In Ethereum sharding, the client's accounts are statically partitioned based on the account address [15] and distributed to each shard. This scheme is referred as static address-based placement method (S-ACC) throughout this paper. The S-ACC causes transaction load imbalance between shards as it does not consider the complexity of transactions and the load generated from the transactions. When such imbalance occurs in Ethereum sharding environments, the number of pending transactions may increase, which in turn lowers transaction throughput and increases the makespan of transaction latency.

This paper proposes a dynamic load balancing mechanism in the Ethereum, D-GAS, which balances the transaction load of each shard based on the gas consumption. The D-GAS is dynamic in the sense that the accounts between shards can be relocated to improve the transaction throughput and minimize the makespan of transaction latency by periodically checking the gas consumption in each shard. To summarize, this paper makes the following specific contributions:

- The method proposed in this paper uses the gas consumption as an indicator for the complexity of transaction load that changes over time. The gas in the Ethereum is a unit of fee that is paid for the computation resources consumed to execute operations in a transaction. Therefore, the amount of gas consumption in a transaction represents the transaction load more accurately than the number of transactions since each transaction may have different complexity.
- The D-GAS consists of two sub-components: transaction load prediction algorithm and account relocation algorithm. Instead of using transient load in the Ethereum, the amount of gas consumption requesting to the account group is predicted. Based on the prediction, the account relocation algorithm dynamically relocates the account groups of each shard by using a heuristic algorithm. This balances the transaction load of each shard and minimizes the makespan of the transaction latency.
- To evaluate the performance of the proposed technique, we have conducted various experiments with the OMNeT++ 5.4.1 simulator [16]. Because of the difficulty of using real workloads, we generated a variety of synthetic workloads that mimic the real workload as much as possible by analyzing the real trace log from the Etherscan [17]. The performance results showed that the transaction throughput is improved by up to 12%, while the makespan of the transaction latency is decreased by up to 74%.

The rest of the paper is organized as follows. Section 2 presents an overview of Ethereum sharding and the motivation behind the proposed mechanism. Section 3 defines the problem. Section 4 discusses the design issues of the D-GAS and Section 5 evaluates the performance of the proposed method. Chapter 6 finally concludes the paper with future challenges.

II. Background and Motivation

This section briefly introduces the Ethereum and its sharding mechanism, and discusses the motivation for the proposed load-balancing algorithm.

A. *Ethereum*

Ethereum is a permissionless blockchain platform for creating and executing *Dapp*s through the smart contract [14]. Smart contract is an application executed on all participating blockchain nodes, which ensures integrity and reliability of its execution results.

Ethereum provides users with the *Turing-complete* programming language and Ethereum virtual machine (EVM) to enable them to create various smart contracts. Ethereum users create a smart contract using the Turing-complete programming language. The smart contracts created by users are compiled into the bytecode to be deployed in the blockchain network. As a deployed smart contact is considered as an account, the contract can be executed in the similar way that users send a transaction to the account. The EVM is a 256-bit virtual machine (VM) that can execute the deployed smart contract. All nodes can execute the deployed smart contract by using the EVM. Therefore, based on the information in the smart contracts that are deployed through blockchain, all nodes can execute all smart contracts and validate the results executed by other nodes.

Ethereum uses a unit called *gas* that represents the amount of computational effort needed to execute operations in a transaction. For example, if we need to run transactions that require more execution cycles, more gas consumption is expected. As shown in Table 1, the amount of gas required to execute a transaction is calculated based on the gas consumption defined for the operation code executed in the transaction. Thus, the gas consumption of a transaction indicates the complexity of the transaction.

TABLE 1. Gas consumption by operation [18]

Name	Value	Description
G_{base}	2 gas	gas for {*ADDRESS, ORIGIN, CALLER* ...}
$G_{verylow}$	3 gas	gas for {*ADD, SUB, NOT* ...}
G_{low}	5 gas	gas for {*MUL, DIV, SDIV* ...}
G_{mid}	8 gas	gas for {*ADDMOD, MULMOD, JUMP* ...}
G_{high}	10 gas	gas for {*JUMPI*}
$G_{extcode}$	700 gas	gas for {*EXTCODESIZE*}
$G_{balance}$	400 gas	gas for {*BALANCE*}

Due to the inherent scalability limitation on the permissionless blockchain networks, Ethereum proposed a mechanism called *sharding* that partitions the Ethereum network into several shards so that each shard executes the transactions in parallel as shown in Figure 1. Each shard contains a *collation* chain (collation is a block in a shard), which is a data structure to process and store transactions in each shard. One of the challenging issues in Ethereum sharding is how to allocate transactions to each shard. Ethereum assigns each account to a shard statically according to its address prefix (we call this as S-ACC). This leads to a load imbalance problem as the complexity of each transaction and the load condition in each shard are not properly considered.

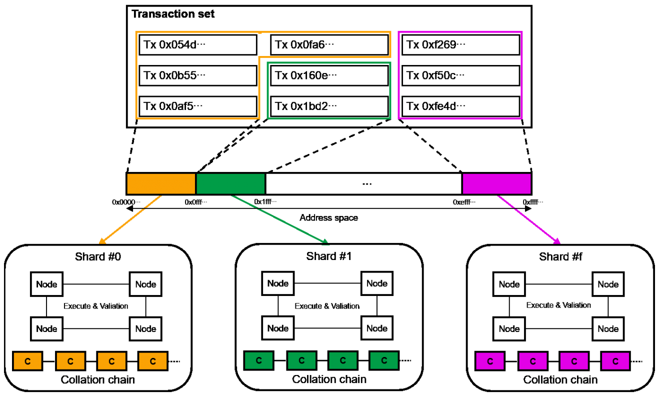

Fig. 1. Overview of Ethereum sharding

B. *Motivation*

We conducted a preliminary experiment with the OMNeT++ 5.4.1 simulator to identify the load imbalance problem between shards based on the S-ACC. We measured the average collation utilization of each shard during the 50 collation cycle, assuming an environment where there are 20 shards. The collation utilization of a shard denotes the actual gas consumption versus the maximum gas consumption that can be included in the collation (i.e., gas consumption / gas limit).

The transaction load used in this experiment was generated using *poisson* distribution and the gas limit of a collation was set to 8,000,000 gas. Figure 2 shows the average collation utilization of each shard. As shown in Figure 2, only 25% of all 20 shards show the maximum utilization, while other 25% shards were under 60% utilization. This indicates that placing accounts on each shard regardless of the complexity of transaction load may cause imbalance in the transaction load. When such imbalance occurs in Ethereum sharding environment, pending transactions may increase, which in turn lowers the transaction throughput and extends the makespan of transaction latency.

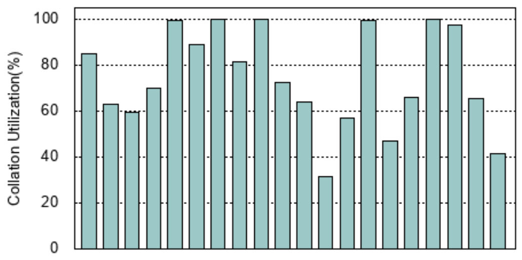

Fig. 2. Collation utilization of 20 shards

III. Problem Definition

The basic system model assumed in the D-GAS conforms to the Ethereum sharding architecture presented in Figure 1, where the number of shards is fixed. As the Ethereum sharding uses the proof-of-stake (PoS) consensus algorithm, every shard has the same collation cycle. Therefore, we can apply the D-GAS to all shards simultaneously. We also assume that the size of collation is limited according to the amount of gas consumption and no malicious attack occurs on the sharding network. All accounts belong to one account group that each account group includes the same number of accounts.

Let us assume that A_i refer to the i-th account group and S_i refer to the i-th shard, respectively. Also assume that N_{ag} refers to number of account group and the number of shards composing the sharding network is defined as N_{sh}. Unlike traditional systems, which create and manage user accounts in the central server, the D-GAS manages the accounts by dividing their addresses into N_{ag} groups. The problem in this study is to assign N_{ag} account groups to N_{sh} shards as evenly as possible so that the transaction throughput is maximized as shown in Figure 3. This is a variation of number partitioning problem [19] that minimizes the difference of the sum of elements in the partitioned subsets. Therefore, the goal of this study is to minimize the sum of the differences between the gas consumption of each shard as shown in Equation (1).

$$\sum_{i=1}^{N_{sh}}(Gas^{limit} - \sum_{for\ all\ A_j \in S_i} Gas_j^{used}) \qquad (1)$$

, where Gas^{limit} is the maximum amount of gas that can be included in a collation, and Gas_j^{used} is gas consumption of j-th account group. Since this problem is considered as NP-complete [20], we propose a heuristic algorithm to solve this problem in the following section.

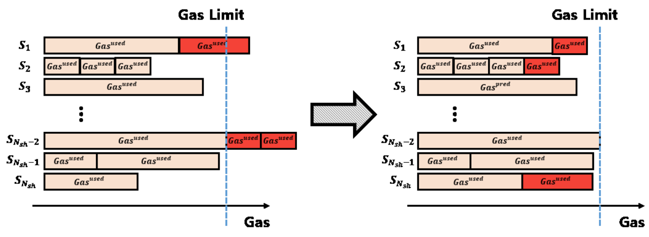

Fig. 3. Problem definition

IV. Design of D-GAS

This section provides an overview of the D-GAS and presents the two algorithms such as the transaction load prediction algorithm and the account relocation algorithm in detail.

A. Overview

This paper proposes a gas consumption-aware dynamic load balancing (D-GAS) mechanism for balancing the transaction load between shards in Ethereum sharding environment. The proposed method consists of the transaction load prediction algorithm and the account relocation algorithm. The D-GAS is executed in a regular interval according to the number of generated collation. Assume that the time interval between two consecutive collations is C^T and the number of collations generated in one shard at every load balancing cycle is N_{col}. Then, P^T, the execution cycle of the D-GAS, is defined as Equation (2). For example, if N_{col} is 5, the D-GAS is executed when 5 collations are generated in one shard.

$$P^T = N_{col} \times C^T \qquad (2)$$

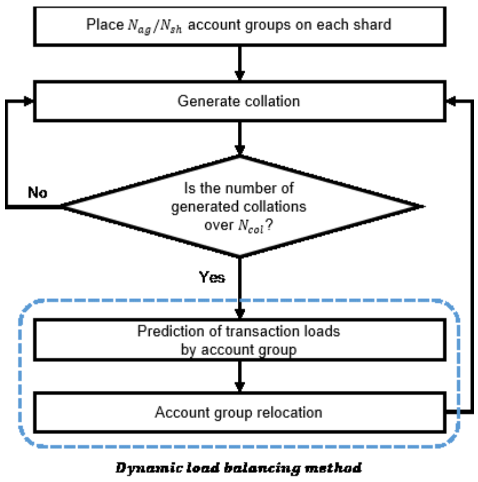
Place N_{ag}/N_{sh} account groups on each shard
Generate collation
Is the number of generated collations over N_{col}?
No
Yes
Prediction of transaction loads by account group
Account group relocation
Dynamic load balancing method

Fig. 4. Overall flow of D-GAS

Figure 4 presents the overall flow of the D-GAS. The D-GAS initially places N_{ag}/N_{sh} account groups in each shard so that the same number of accounts can be located in each shard. When the number of generated collations reaches to N_{col}, the D-GAS runs the transaction load prediction algorithm for each account group based on the gas consumption. With the predicted values, the D-GAS invokes the account relocation algorithm to determine whether the accounts should be relocated or not.

B. Transaction Load Prediction Algorithm

The transaction load prediction algorithm predicts the transaction load that will occur in the future, based on the gas consumption of the previous transactions. This algorithm has the following features. First, it predicts the transaction load with the gas consumption instead of the number of transactions. This is because the transactions in the Ethereum are much more complex and the number of transactions cannot properly reflect the complexity of the transactions. Second, it predicts the future transaction load using the gas consumption that has been processed in one cycle before the last cycle. This allows the Ethereum to have enough time to prepare and validate the collations that were newly relocated in each shard.

Let us assume that $Gas_{i,j}^{used}$ is the gas consumption of the i-th account group in the j-th collation and W_j is the weight value for the j-th collation. Then, Gas_i^{pred}, the predicted gas consumption of the i-th account group defined in Equation (3) is the sum of product of $Gas_{i,j}^{used}$ and W_j , where j is increased from 1 to N_{col}. Moreover, the W_j in Equation (4) is defined such that the weighting for each older gas consumption values decreases exponentially (i.e., put more weight on the nearest past). Figure 5 shows the detailed steps of Gas_i^{pred} calculation in each account group when the N_{col} is 5.

$$Gas_i^{pred} = \sum_{j=1}^{N_{col}} (Gas_{i,j}^{used} \times W_j) \tag{3}$$

$$W_j = \frac{j}{\sum_{k=1}^{N_{col}}(k)} = \frac{2 \times j}{N_{col}(N_{col}+1)} \tag{4}$$

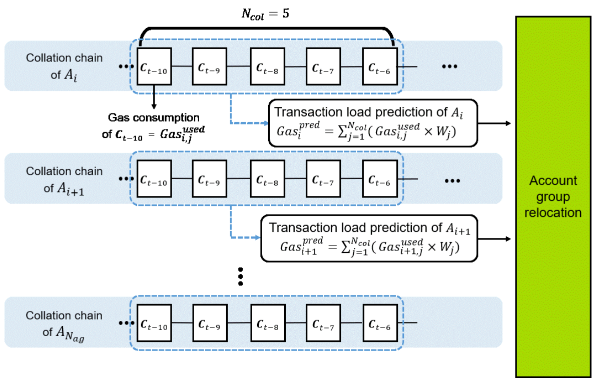
$N_{col} = 5$
Collation chain of A_i
C_{t-10}
C_{t-9}
C_{t-8}
C_{t-7}
C_{t-6}
Gas consumption of C_{t-10} = $Gas_{i,j}^{used}$
Transaction load prediction of A_i
$Gas_i^{pred} = \sum_{j=1}^{N_{col}}(Gas_{i,j}^{used} \times W_j)$
Collation chain of A_{i+1}
Transaction load prediction of A_{i+1}
$Gas_{i+1}^{pred} = \sum_{j=1}^{N_{col}}(Gas_{i+1,j}^{used} \times W_j)$
Collation chain of $A_{N_{ag}}$
Account group relocation

Fig. 5. Transaction load prediction overview

C. Account Relocation Algorithm

The account relocation algorithm uses the predicted transaction load obtained from the transaction load prediction algorithm described above. Figure 6 shows an overview of account group relocation based on the predicted transaction load. The account relocation algorithm creates a priority queue based on the information from the N_{ag} account groups, and puts the account group with the bigger transaction load to come first. Then, it selects an account group from the queue and relocates it to a shard with the minimum gas consumption. That is, the destination shard is the one with the smallest $\sum_{A_j \in S_i} Gas_j^{pred}$ value, where the set of account groups in the i-th shard is defined as S_i. Finally, the previous steps are repeated until every account group is relocated. In order to minimize the time complexity of ordering the account groups in the descending order, we used a max heap data structure for the priority queue. On the other hand, we used a min heap data structure for finding the shard with the smallest $\sum_{A_j \in S_i} Gas_j^{pred}$ value to minimize the time complexity.

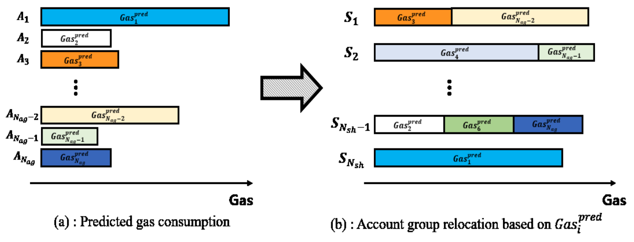

Fig. 6. Account group relocation by predicted transaction load

V. Performance Evaluation

In this section, we evaluate the performance of D-GAS and present the comparison with existing approaches.

A. Experiment Setup

In order to evaluate the performance of the D-GAS, we used OMNeT++ 5.4.1 simulator [16]. For the simulation, we assumed that an Ethereum sharding environment consists of 20 shards and there exist 100 account groups. The gas limit of each collation is set to 8,000,000 gas as is the case in current Ethereum.

TABLE 2. Transaction load workload

Gas consumption		
Type	Value (gas)	Ratio
G-Small	20742	30%
G-Medium	51857	40%
G-High	105153	30%

Table 2 shows the types of workload generated for the simulation. As shown in Table 2, we generated three types of workload based on the amount of gas consumption in a transaction: G-Small, G-Medium, and G-High. For this, we analyzed about 1400 most recent transactions from Etherscan [17] that have actually occurred and classified them as G-Small, G-Medium, and G-High. For example, G-Small includes the transactions with the gas consumption of bottom 30%, while G-Medium and G-High include the transactions with the gas consumption of medium 40% and high 30%, respectively. The gas value represents the average gas consumption of each category.

In order to analyze the performance of the D-GAS, we have also implemented two different allocation schemes: S-ACC and

D-TX. The S-ACC is a static address-based placement method currently used in the Ethereum and the D-TX is an allocation scheme based on the number of transactions. For the comparison, we measured transaction throughput, makespan of the transaction latency and load balancing efficiency

B. Ethereum Performance

In order to analyze the Ethereum performance, we carried out an experiment by varying load balancing execution cycle P^T ($5C^T$, $10C^T$, $15C^T$). As discussed in Section 4 (Equation 2), P^T is a value obtained by multiplying N_{col} and C^T, where N_{col} is the number of collations generated in one shard at every load-balancing execution cycle and C^T is a time interval between two consecutive collations.

Figure 7 (a) and (b) show the transaction throughputs and makespans of three allocation mechanisms normalized with the S-ACC by varying load-balancing execution cycle P^T. As shown in Figure 7 (a), the D-GAS outperforms other allocation schemes by about 9% on average and up to 12% to the maximum in all load-balancing execution cycles. Furthermore, the D-GAS also shows smaller makespans compared to the S-ACC and the D-TX as shown in Figure 7 (b). Compared to the makespan of the S-ACC, the D-GAS lowers the makespan by about 55% on average and by about 74% at the short load-balancing execution cycle. This indicates that the gas-based load balancing mechanism proposed in this paper is efficient in improving the transaction throughput and reducing the makespan of the transaction latency.

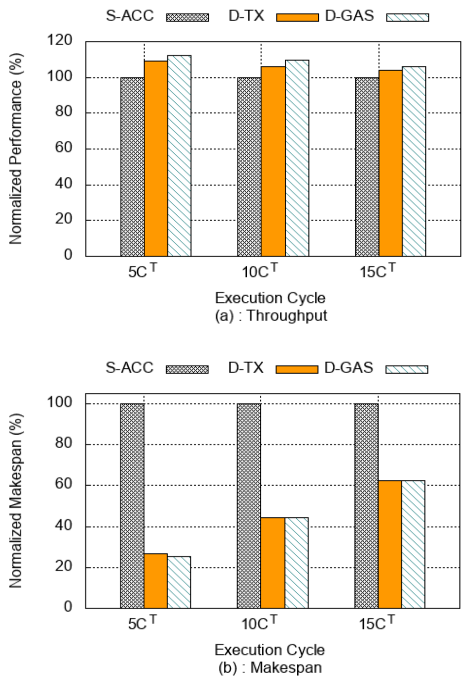

Fig. 7. Normalized performance

C. Load Balancing Efficiency

Figure 8 depicts the standard deviations of transaction throughputs in three allocation mechanisms normalized with the S-ACC. Similarly, as shown in Figure 8, the D-GAS balances the transaction load more evenly than the S-ACC and the D-TX. For example, the standard deviation of the D-GAS is lower than that of the S-ACC by about 55% on average and 75% to the maximum. Moreover, as we shorten the load balancing execution cycle, the performance gap widens. This is because the short execution cycle can easily react to the changes in transaction load.

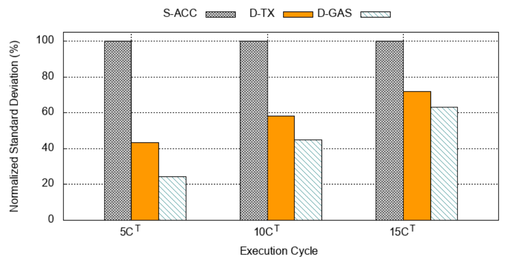

Fig. 8. Normalized load balancing efficiency

VI. Conclusion

This paper proposed a gas consumption-aware dynamic load balancing mechanism called D-GAS for balancing the transaction load of each shard by periodically relocating the accounts between shards. In order to evaluate the performance of the D-GAS, we have implemented two existing allocation algorithms such as the S-ACC and the D-TX, and compared the performance under various conditions. The performance results confirmed that the D-GAS was efficient in load balancing and outperformed other mechanisms in terms of transaction throughput and makespan.

Further investigations on the overhead analysis of the D-GAS mechanism and the evaluation under real workloads will be needed in the future study.

Acknowledgment

This research was supported by the MSIT (Ministry of Science and ICT), Korea, under the ITRC (Information Technology Research Center) support program (IITP-2019-2017-0-01628) supervised by the IITP (Institute for Information & communications Technology Promotion).

References

[1] S. Nakamoto, “Bitcoin: A peer-to-peer electronic cash system”, Available online: https://bitcoin.org/bitcoin.pdf (accessed on 15th Dec 2017).

[2] Mettler. Matthias, “Blockchain technology in healthcare: The revolution starts here”, Proc. IEEE 18th int. Conf. E-Health Netw. Appl. Services (Healthcom), 2016, pp. 1-3.

[3] Dorri. Ali, et al, “Blockchain for IoT security and privacy: The case study of a smart home”, In: Pervasive Computing and Communications Workshops (PerCom Workshops), 2017 IEEE International Conference on. IEEE, 2017. p. 618-623.

[4] Azaria. Asaph, et al, "Medrec: Using blockchain for medical data access and permission management", Open and Big Data (OBD), International Conference on. IEEE, 2016.

[5] Eyal. Ittay, et al. "Bitcoin-NG: A Scalable Blockchain Protocol", NSDI, 2016.

[6] Zheng. Zibin, et al. “Blockchain challenges and opportunities: A survey”, International Journal of Web and Grid Services, 2018, 14.4: 352-375.

[7] Pending Ethereum transactions after Cryptokitties’ realse. Available online: https://www.theatlas.com/charts/rkt8jKMZz (accessed on 2nd Jan 2019).

[8] Wei. Cai, Zehua. Wang, Jason. B. Ernst, Zhen. Hong, Chen. Feng, Victor. C, M. Leung, “Decentralized Applications: The Blockchain-Empowered Software System”, IEEE Access, vol.6, Sept 2018.

[9] Poon. Joseph, Dryja. Thaddeus, “The bitcoin lightning network: Scalable off-chain instant payments”, Available online: https://lightning.network/lightning-network-paper.pdf (accessed on 17th Jan 2018).

[10] VUKOLIĆ. Marko, “The quest for scalable blockchain fabric: Proof-of-work vs. BFT replication”, In: International Workshop on Open Problems in Network Security. Springer, Cham, 2015. p. 112-125.

[11] Cachin. Christian, “Architecture of the hyperledger blockchain fabric”, In: Workshop on Distributed Cryptocurrencies and Consensus Ledgers. 2016.

[12] Eleftherios. Kokoris-Kogias, Philipp. Jovanovic, Linus. Gasser, Nicolas. Gailly, Ewa. Syta, Bryan. Ford, “OmniLedger: A Secure, Scale-Out, Decentralized Ledger via Sharding”, In 2018 IEEE Symposium on Security and Privacy (SP). IEEE,2018. p. 583-598.

[13] Zamani. Mahdi, Movahedi. Mahnush, Raykova. Mariana, “RapidChain: scaling blockchain via full sharding”, In: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, ACM, 2018. p. 931-948.

[14] THE ETHEREUM COMMUNITY. Ethereum white paper. Available online: https://github.com/ethereum/wiki/wiki/White-Paper (accessed on 13rd Feb 2019).

[15] Vitalik. Buterin, Ethereum Sharding FAQ. Available online: https://github.com/Ethereum/wiki/wiki/Sharding-FAQs (accessed on 15th Oct 2018).

[16] A. Varga, et al, “The omnet++ discrete event simulation system”, In Procedding of the European simulation multiconference, 2001, vol. 9, pp. 65.

[17] Etherscan Team, Etherscan: The Ethereum block explorer. Available online: https://etherscan.io/ (accessed on 24th May 2018).

[18] Wood. G, Ethereum: A secure decentralised generalised transaction ledger. Available: https://ethereum.github.io/yellowpaper/paper.pdf (accessed on 20th May 2018).

[19] S. Mertens, "The Easiest Hard Problem: Number Partitioning" in Computational Complexity and Statistical Physics, Oxford Univ. Press, vol. 125, 2006.

[20] Richard E. Korf, “Multi-way number partitioning”, In Proceedings of the 21st international jont conference on Artificial intelligence, 2009, pp. 538–543.