A Method to Predict the Performance and Storage of Executing Contract for Ethereum Consortium-Blockchain

Huijuan Zhang(✉), Chengxin Jin(✉), and Hejie Cui(✉)

Tongji University, 1239 Siping Rd, Shanghai, People's Republic of China
mszhj@tongji.edu.cn, jinchengxin@outlook.com, cuihejie331771@gmail.com

Abstract. As the fundamental technology of Bitcoin, blockchain enables people to deal with trust problems in network. Ethereum, as a well-known public blockchain, is favored by large companies and organizations for its excellent account model and Turing-complete smart contracts, and is widely used to develop consortium-blockchain. However, the performance and storage of executing contract gradually degrade as the transaction volume increases. Meanwhile, compared with public blockchains, companies need a more accurate estimation of prospective performance and storage based on business scale for decisions making or early warnings. In this paper, a prediction model derived from the core structure of Ethereum's "World State" is proposed. The proposed model predicts the performance and storage of executing contract based on transaction volume. The comparison between the experimental and predicted data reveals that this model can perform a relative accurate prediction of the prospective system's performance and storage.

Keywords: Blockchain · Ethereum · Contract · Performance · Storage

1 Introduction

Ethereum [3], as the successor of Bitcoin [5], establishes a Turing-complete smart contract on blockchain to realize distributed application DApps [6]. Meanwhile, the account-based design of Ethereum provides convenience for the docking of existing business models (compared with UXTO model). Thus many companies choose Ethereum to build their consortium-blockchain system or develop on it (for example EEA [1]) based on two points. As a result, Compared with the public Ethereum blockchain, Ethereum consortium-blockchain mainly uses the transaction to execute the contract instead of making a transfer of ETH cryptocurrency. Therefore, this paper focuses on the performance and storage of executing the Ethereum contract.

However, the test results show that when the transaction volume reaches a certain scale, the execution performance of Ethereum will significantly reduce and large storage space will be occupied. (e.g. when the limit of block generation rate was modified to one block per second, the TPS of a contract, which is

S. Chen et al. (Eds.): ICBC 2018, LNCS 10974, pp. 63–74, 2018.
https://doi.org/10.1007/978-3-319-94478-4_5

about 200 at start, would reduce to 100 as the transaction volume reaches one million). For companies, predicting the prospective performance and storage are important indicators in making technical decisions, and also allow companies to prepare for hardware, monitoring and plans in advance. Thus, it is necessary to predict the prospective performance and storage of the system in Ethereum consortium-blockchain based on the business scale.

In public Ethereum blockchain, it's impossible to predict the distribution and complexity of smart contract since anyone can deploy a contract easily. As a result, it's hard to accurately estimate the perspective performance and storage. Nevertheless, it is possible to predict the performance and storage in Ethereum consortium-blockchain resulting from that the participants are the authorized nodes; that the relatively-fixed smart contract with evaluable complexity is determined by business model; and that the transaction volume is determined by business scale.

The prediction method proposed in this paper speculates the performance and storage by analyzing the relationship between transaction volume and "World State" [7]. "World State" is the core part of Ethereum. The account system maps the state data as key/value form and stores them in LevelDB through this special structure [13]. "World State" is implemented using "the modified Merkle Patricia tree (trie)" [4] (hereinafter referred to as MPT). PATRICIA trie (Patricia tree) is a space-optimized version of the traditional trie data structure, in which every node with only one child is merged with its child. This data structure was firstly proposed by Morrison [16] in 1968, and then well analyzed in "The art of computer programming" by Knuth [17] in 1973. The "Merkle" part of the radix trie arises in the fact that a deterministic cryptographic hash of a node is used as the pointer to the node, leading to the fact that the Ethereum could trace the history state through root of "World State" in any block header. The contract in Ethereum is called transaction. Depending on the implementation of Ethereum [2], the time consumed by a transaction call for contract is mainly determined by the execution time of Ethereum Virtual Machine (EVM) [7] and by the modification of the "World State". Meanwhile, the increment of data generated by this transaction depends on the transaction scale and the increment amount of the "World State". With the transaction volume growing, the "World State" becomes larger, resulting in the rise of time consumption and data space. Consequently, the estimation of performance and storage can be obtained under the premise of figuring out the relationship between transactions volume and "World State".

This paper is focused on the relationship between the performance and storage increment of "World State" after the transaction volume reaches a certain scale, and a prediction formula is proposed for this relationship. Using this formula, companies could predict the prospective time consumption of executing a transaction and the storage occupancy based on the transaction volume. At the end of the paper, a suggestion is raised for the design of contract in Ethereum consortium-blockchain.

The rest of this paper is organized as follows. Section 2 reviews relatived development on blockchain. Section 3 gives the key technologies to introduce how to predict the performance and storage. Section 4 provide the experiments to prove the predict methods. Section 5 concludes this paper.

2 Relatived Work

Since blockchain is a distributed ledger maintained by all participants, many researches have been focused on PBFT [11] and other consensus algorithms to improve the efficiency. In this case, the performance of executing contract would become the bottleneck restricting the system, but this research area is rarely explored at present. For the contract of Ethereum, most researches are focused on the security [9] of smart contract or improving its smartness [10], while the effect of execution efficiency is scarcely investigated. High-Performance Computing is discussed for Ethereum Tokens [18], but not the performance bottleneck of Ethereum itself. The assessment [8] results show that the performance will decline and delay be high when the transaction volume grows (10,000 transactions), however the test volume in that paper is small and there are few analysis details. In terms of storage, EtherQL [12] provides highly-efficient query primitives for analyzing blockchain data, but not for predicting the size of data space. On the other hand, the properties of PATRICIA trie are very important due to the highly-correlated relationship between the performance/storage and "World State", as well as the close tie of "World State" MPT implementation with PATRICIA trie. Some existing papers point out that the height of the PATRICIA trie behaves quite differently across regions: it exhibits an exponential of a Gaussian distribution, which satisfies $log(n)$ [15]. There is another paper analyzing the relationship between the average height and random inserting in PATRICIA trie. To sum up, there is not a single paper concentrating on MPT and the relationship between performance/storage and MPT.

3 Key Technologies

3.1 Influencing Factors

In Ethereum, the time consumption of executing a transaction could be divided into two parts: the EVM execution time and the cost of modifying "World State", while the storage increment is composed of transaction volume and state data of changed "World State". Since the contracts in consortium-blockchain are usually fixed and predictable, the time consumption of EVM and the transaction volume tend to be stable. Therefore, the root cause for the change in performance and storage is the maintenance cost of "World State".

After every transaction is executed through a smart contract, it eventually modifies the tree node of "World State". The state of Ethereum is the result of mapping 160-bit address and account state in the tree, which is called `State Trie`. An account state corresponds to a leaf node while its address is depicted as the path from the root to the node. In Ethereum, the smart contract is also an account. Each contract account has its data storage, called `Storage Trie`. Data storage is also implemented using the MPT tree. Hence, the "World State" tree of Ethereum is actually composed of two parts: an upper account tree and a lower storage one. There are two ways to create an Ethereum contract: one is creating from a transaction, the other is from an old contract. The new contract will be stored in State Trie of "World State" and contract data in Storage Trie whose root points to the contract. For a specific business logic, depending on how the contract is implemented, the data entering the "World State" would be mainly distributed in State Tire (constantly create new contract to store more data) or Storage Trie (create a single contract to store a large amount of data) or between (a balanced distribution of State Tire and Storage Trie).

The nodes of an MPT tree are divided into `leaf`, `expansion`, and `branch` (`NULL` is not within the scope of our discussion). Leaf nodes and expansion nodes are similar in size, while branch nodes are much larger. The "World State" would be modified for multiple times according to the design of contract after the transaction calls a contract. Due to the features of tracing history, every modification would generate a new path from the new leaf to the root. In the implementation of Ethereum, when searching a node in the path or inserting a new node, the system would execute function $sha3()$ and read/write LevelDB which brings more time consumption, and the size of new node corresponds to the data increment in this modification. Therefore, if the height of current new leaf can be predicted, it is possible to infer the performance and data increment. The random hash of Ethereum address results in the random modification for MPT, thus the average height of the leaf nodes would increase as the times of modification increase. When a state change occurs for MPT, it is possible to speculate the execution time and storage space increment if the current tree height of new leaf nodes could be estimated.

As a result, predicting the performance and storage occupancy of a transaction can be achieved by analyzing the times of modification for "World State" or testing the business contract, and calculating the time consumption and storage based on the current height of "World State" in MPT.

3.2 Height of MPT

MPT is evolved from PATRICIA trie. In the implementation of MPT, a branch node could hold 16 branches. When the branches reach a certain scale, most nodes in MPT would become branch nodes, while the extension nodes are compressed prefix in PATRICIA trie. Therefore, MPT could be seen as a 16-ray PATRICIA trie.

Considering of the huge space of 160-bit address, the random selection of address could be approximated as an asymptotic distribution. Besides, the address would be executed by $sha3()$ to become the key of MPT, so the probability of choosing which alphabet be inserted into MPT is equal. As the result, the height of MPT can be regarded as the expected height of a 16-ray Patricia tree with the universal asymptotic distribution insertion of equal-probability alphabet.

According to Devroye [14], the expectated height of universal asymptotic PATRICIA trie is given by

$$\mathbf{E}\{H_n\} \sim c\ log\ n$$
$$where\ c = 2/log_2(1/\textstyle\sum_j\ p_j^2)$$

and the highest node is expressed by

$$\frac{H_n - log_2 n}{\sqrt{2log_2 n}} \to 1$$

Hence, the expectation and the max height of 16-ray MPT ($p_j = 1/16, 1 \leqslant j \leqslant 16$) can be obtained as:

$$\mathbf{E}\{H_n\} \sim\ log(n)/2$$
$$H_n = \frac{2\sqrt{2log_2 n} + log_2 n}{4}$$

3.3 The Prediction of Transaction Performance and Storage

A simple instance of Ethereum contract is used to describe the relationship between the transaction and performance/storage, as shown in Fig. 1.

```
pragma solidity 0.4.16;
constract Order {
  uint256 private order_no;
  func Order(uint256 _no) { _no = order_no; }
  func doOrder() { ... }
}

constract Business {
  uint256 private version;
  func createOrder(uint256 _no) returns(address) {
    Order o = new Order(_no);
    o.doOrder();
    return o;
  }
}
```

Fig. 1. Simple business model contract in Ethereum consortium-blockchain.

When a transaction calls the function `createOrder` in contract business, the Ethereum actually loads the contract business code from State Trie into EVM, and calls the function `createOrder` to create a new contract order (account), and then the function in new contract will be invoked. At the committing phase of the entire process, the data in cache would be inserted to "World State" to form a new world state and generate a series of new nodes. The tree root of the current world state is written to the new block so that the state can be extracted from this block. Like `block 3` in Fig. 2, the path from the leaf contract "`Order`" to the root is the new inserted node in `block 3` compared with `block 2`. These nodes are the potential influential factors that affect efficiency and storage. The contract "`Order`" and "`Business`" are stored in State Trie, and the attributes "`order_no`", and "`version`" are stored in the instance of contract "`Order`" and "`Business`". It's obvious that the State Trie and Storage Trie would insert data continually as long as the transaction calls function `createOrder`.

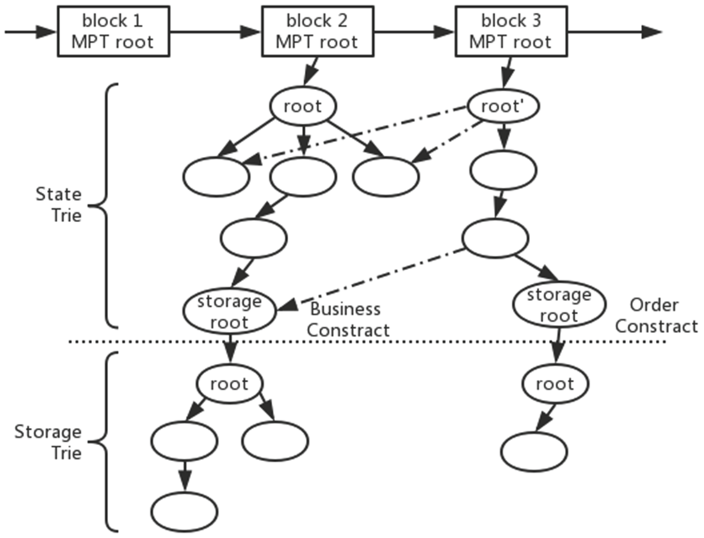

Fig. 2. "World State" in Ethereum.

For a specific business model, the State Trie and Storage Trie would be assigned different proportions of data depending on the design of contract in the consortium-blockchain. In general case, three types of models can be designed, as shown in Fig. 3:

1. The data are distributed on State Trie by the method of creating new contracts from old contract.
2. For the minority contracts with complex storage structure, the data are mainly stored in the Storage Trie of those contracts.
3. New contracts are created and appropriate contracts data storage is designed, and the data are allocated to State Trie and Storage Trie as needed.

Thus, after the prediction of the height of MPT, different designs of contracts would lead to different prediction methods, but the design ideas are the same.

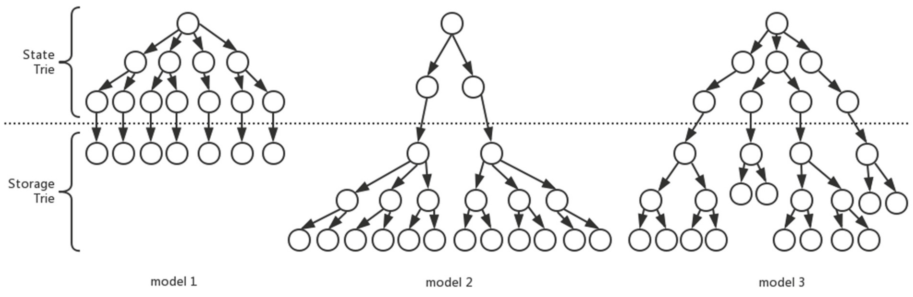

Fig. 3. Three type of business model.

It is supposed that the contract is designed as the first type, which mainly distributes the data on the State Trie. Considering a general business scene, a transaction would deploy a times contract (e.g. the `Order` contract in Fig. 1) for a business request, and the time consumption of executing or generating every node is t (including the calculation of $sha3()$ and database access time). For the situation depicted above, when the transaction volume reaches n and a new transaction is executed, the average time consumption of modifying MPT is given by

$$\mathbf{T}_{mpt}(n) = \frac{t}{2} log(an)$$

The max time consumption is represented by

$$\mathbf{T}_{mpt\_max}(n) = \frac{t}{4}(2\sqrt{2log_2(an)} + log_2(an))$$

Ethereum uses LevelDB as the database to store key/value. The key to accessing database is irregular on account of the discreteness of hash. The LevelDB has an excellent performance in reading/writing continuously, while bad for random key [13]. Therefore, the time t for accessing LevelDB would be longer as the amount of data storage increases. In fact, the test results show that if n is large enough, the value of t will increase and the efficiency will degrade largely for some data which not hit LevelDB cache at times.

Then, the T_{exec} is added to execute the contract in virtual machine. After n transactions are executed, the average execution time and the maximum time of a transaction are given by

$$\mathbf{T}_{avg}(n) = \mathbf{T}_{exec} + \mathbf{T}_{mpt}(n)$$
$$\mathbf{T}_{max}(n) = \mathbf{T}_{exec} + \mathbf{T}_{mpt\_max}(n)$$

Because of the discreteness of hash, the common prefixes between addresses are hardly identical. So, most nodes of MPT are on the branch. When n is large, it can be assumed that the branches are fully filled. At the same time, since the length of address is fixed, only the leaf nodes can store state, rather than the branch or extension nodes. It is assumed that the filled branch size is s_b, the size of the leaf node is s_l, the storage space occupied by the account state is s_a, and the sum of storage trees for each account is represented by s'. After performing n transactions, the storage increment resulting from modifying the MPT tree is calculated by

$$\mathbf{S}_{mpt}(n) = s_b \left(\frac{log(an)}{2} - 1 \right) + s_l + s_a + s^{'}$$

the maximum storage increment is given by

$$\mathbf{S}_{max}(n) = s_b \left(\frac{2\sqrt{2log_2(an)} + log_2(an)}{4} - 1 \right) + s_l + s_a + s^{'}$$

The size of the transaction itself is S_t. The total storage and maximum space occupation after n times of transaction executions are expressed as

$$\mathbf{S}_{sum\_avg}(n) = nS_t + \textstyle\sum_{i=0}^{n} S_{mpt}(n)$$
$$\mathbf{S}_{sum\_max}(n) = nS_t + \textstyle\sum_{i=0}^{n} S_{max}(n)$$

The same analysis also applies to designing the contracts of type 2 and 3.

4 Experiment

4.1 Experimental Method

Experiments are conducted based on the formula in Sect. 3.3 to verify the correctness of the proposed prediction method. The recorded experimental data are compared with the predicted data. Three dimensions of data are collected in the experiment, i.e. the tree height of the State Trie in the "World State", time consumption of execution, and the storage occupancy after transaction.

The testing conditions of the experiment are as follows: 1. Build a single Ethereum node, without networking. 2. Modify the logic of generating block so that one block will only package one transaction (e.g. each transaction would have one submit, to avoid the impact of the Ethereum cache on the experimental results). 3. Using the simple contract presented in Fig. 1, only one new `Order` account contract will be created each time when the `createOrder` method is called in this contract. 4. Execute randomly one million transactions in the system. We use a single server to establish the test environment which consists of two Intel Pentium CPU, 8 G RAM and SSD for storage.

The contract design presented in Sect. 3.3 meets the first-type contract model in Fig. 3, where the data is mainly distributed on State Trie. The design of the contract indicates that each transaction creates a new contract and only changes the State Trie once, that is $a = 1$(do nothing in `doOrder()`). According to the implementation of MPT, when the branch is full, its size is $s_b = 532 + 32 = 564\,B$, the leaf is smaller than $s_l = 96\,B$, and $s_a = 70{+}32 = 102\,B$. The storage s' is regarded as 0, and the average time consumption of each visit to the database is $t = 0.03$ ms (Since the performance of LevelDB random access will decrease drastically as the amount of data increases, t will continue to increase. For the sake of simplicity, the average value in a test is used here). Further, as LevelDB will compress the stored data, we use the sum of key and value which would be stored in LevelDB for space occupancy instead of the amount of space in disk.

Taking the above data into the equations in Sect. 3.3, the prediction formula can be expressed by

$$H_{max}(n) = \frac{2\sqrt{2log_2(n)}+log_2(n)}{4}$$
$$H_{avg}(n) = \frac{log(n)}{2}$$
$$T_{predict}(n) = \frac{0.06*log(n)}{2}$$
$$S_{predict}(n) = 564 * \left(\frac{log(n)}{2} - 1\right) + 96 + 102$$

4.2 Experiment Result

The experimental data are collected and analyzed. In the following figures, the sampling of the experimental data is represented by a red dot, and the blue curve represents the prediction curve.

Transactions and MPT Height. Figure 4 is the prediction for the height of Storage Trie. The fitting curve of $H_{fitting}(n) = 0.356594log(n) + 1.767782$ is obtained based on the actual sampling data. This curve shows the change of the actual tree height, which is represented by the green curve in the figure. The orange curve is the predicted maximum tree height. It can be found from Fig. 4 that our prediction curve is certainly close to the fitting curve, and the sampling points are mainly distributed on both sides of the curve. Besides, there are almost no sample points going over the orange curve, indicating that the prediction curve can predict the tree height of the State Trie better as the transaction volume increases. At the same time, the prediction of the maximum tree height also indicates the change in the upper limit of the tree height.

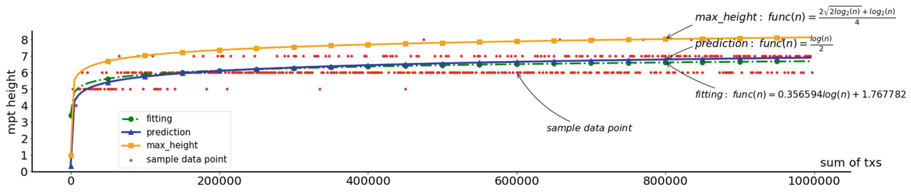

Fig. 4. Transactions and MPT height. (Color figure online)

Transactions and Time Cost. Figure 5 is the prediction for the number of transactions and time cost. As t changes in the production environment, and the change is mainly determined by LevelDB, which is beyond the scope of this paper, a fixed value t is adopted as replacement in the prediction curve. It can be found from Fig. 5 that for the transaction volumes less than 200,000, there are deviations between the predictive value and actual experimental value. This is because the actual t is much smaller than the adopted fixed value. As the transaction volume increases, the experimental data tends to be evenly distributed on both sides of the curve. Although a simple method is used here for the predicted value, the experimental results validate this prediction. An ideal prediction curve can be obtained if the fixed t is replaced by getting the relationship between the read time t and data volume of the LevelDB from an actual test or a theoretical calculation.

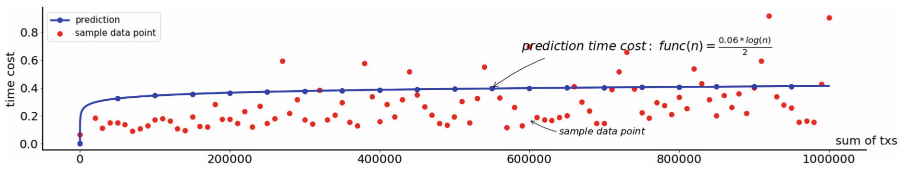

Fig. 5. Transactions and time cost.

Transactions and Storage. Figure 6 is the prediction for the number of transactions and storage occupation. On account of the multi-level cache in the implementation of Ethereum, a scheme of one-transaction-one-block is adopted in testing. Thus, the testing object here is the change in maximum space occupation. In production environment, if multiple transactions are packaged in one block, the data cache will cause the actual stored data to be smaller than the prediction (it is found in testing that if more than 100 transactions are stored in a block, the storage occupancy is roughly half of the prediction). Adding LevelDB compression, the space will be smaller. Thus, it can be concluded that the growth of the storage under the current test conditions is consistent with the prediction.

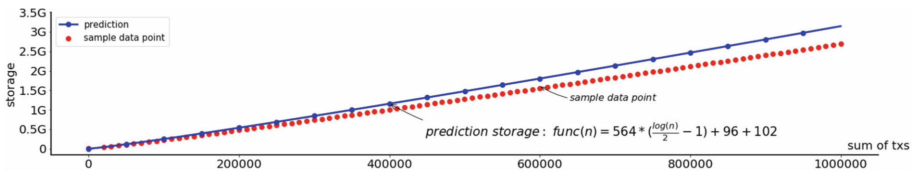

Fig. 6. Transactions and storage

From the results of these three experiments, we can see the prediction method presented in the third chapter is more consistent with the experimental expectation.

5 Conclusions

When companies use the Ethereum consortium-blockchain, they need to make predictions about the prospective performance and storage of the system if the transaction reaches a certain scale. This paper analyzes the core issues that affect the performance and storage of Ethereum is the “World State”. According to the analysis result that “World State” is implemented by MPT, the relationship between MPT performance or storage increment and transaction volume n is obtained to be $log(n)$. On the other hand, the “World State” is made up of the upper layer of State Trie and the underlying layer of Storage Trie. The data distribution would be different based on the organization of contracts. The formulas are offered for a business model to predict the relationship between transaction volume and performance/storage based on State Trie. Other business models can be derived by the same method. In this way, the companies can deduce the prospective performance and storage of blockchain according to their own contracts under the premise that its business scale can be predicted (transaction volume).

At the same time, when transaction scale could be estimated, we can properly design the contracts in order to minimize the consumption of performance and storage, so that the data distribution between the State Trie and Storage Trie could reach an inflection point, which minimizes the costs of performance and storage. Therefore, it is suggested that the contract developer can estimate the future transaction volume in advance and properly allocate the data in the state tree and storage tree to achieve the optimal efficiency and storage when writing the contract.

References

1. Enterprise Ethereum Alliance. https://entethalliance.org/
2. GitHub. ethereum/cpp-ethereum. https://github.com/ethereum/cpp-ethereum
3. Etherchain.org. etherchain.org - The Ethereum Blockchain Explorer. https://etherchain.org/
4. GitHub. ethereum/wiki. Merkle Patricia Trie Specification (also Merkle Patricia Tree). https://github.com/ethereum/wiki/wiki/Patricia-Tree
5. Nakamoto, S.: Bitcoin: A peer-to-peer electronic cash system (2008). https://bitcoin.org/bitcoin.pdf
6. Buterin, V.: A Next-Generation Smart Contract and Decentralized Application Platform, Ethereum White Paper. https://github.com/ethereum/wiki/wiki/White-Paper
7. Wood, G.: Ethereum: a secure decentralised generalised transaction ledger (2014). http://gavwood.com/paper.pdf
8. Pongnumkul, S., Siripanpornchana, C., Thajchayapong, S.: Performance analysis of private blockchain platforms in varying workloads. In: 2017 26th International Conference on Computer Communication and Networks (ICCCN), Vancouver, BC, pp. 1–6 (2017)

9. Atzei, N., Bartoletti, M., Cimoli, T.: A survey of attacks on ethereum smart contracts (SoK). In: Maffei, M., Ryan, M. (eds.) POST 2017. LNCS, vol. 10204, pp. 164–186. Springer, Heidelberg (2017). https://doi.org/10.1007/978-3-662-54455-6_8
10. Loi, L., Chu, D.H., Olickel, H., Saxena, P., Hobor, A.: Making smart contracts smarter. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security (CCS 2016), pp. 254–269. ACM, New York (2016). https://doi.org/10.1145/2976749.2978309
11. Sankar, L.S., Sindhu, M., Sethumadhavan, M.: Survey of consensus protocols on blockchain applications. In: 2017 4th International Conference on Advanced Computing and Communication Systems (ICACCS), Coimbatore, pp. 1–5 (2017). https://doi.org/10.1109/ICACCS.2017.8014672
12. Li, Y., Zheng, K., Yan, Y., Liu, Q., Zhou, X.: EtherQL: a query layer for blockchain system. In: Candan, S., Chen, L., Pedersen, T.B., Chang, L., Hua, W. (eds.) DASFAA 2017. LNCS, vol. 10178, pp. 556–567. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-55699-4_34
13. LevelDB github page. https://github.com/google/leveldb
14. Devroye, L.: Universal asymptotics for random tries and PATRICIA trees. Algorithmica **42**, 11 (2005). https://doi.org/10.1007/s00453-004-1137-7
15. Knessl, C., Szpankowski, W.: Heights in generalized tries and PATRICIA tries. In: Gonnet, G.H., Viola, A. (eds.) LATIN 2000. LNCS, vol. 1776, pp. 298–307. Springer, Heidelberg (2000). https://doi.org/10.1007/10719839_31
16. Morrison, D.R.: PATRICIA practical algorithm to retrieve information coded in alphanumeric. J. ACM **15**(4), 514–534 (1968)
17. Knuth, D.E.: The Art of Computer Programming. Sorting and Searching, vol. III. Addison-Wesley, Redwood City (1973)
18. Dhillon, V., Metcalf, D., Hooper, M.: Ethereum Tokens: High-Performance Computing. Blockchain Enabled Applications. Apress, Berkeley (2017)