A Detailed and Real-time Performance Monitoring Framework for Blockchain Systems

Peilin Zheng
Sun Yat-sen University, China
zhengpl3@mail2.sysu.edu.cn

Zibin Zheng
Sun Yat-sen University, China
zhzibin@mail.sysu.edu.cn

Xiapu Luo
The Hong Kong Polytechnic University, China
csxluo@comp.polyu.edu.hk

Xiangping Chen
Sun Yat-sen University, China
chenxp8@mail.sysu.edu.cn

Xuanzhe Liu
Peking University, China
liuxuanzhe@pku.edu.cn

ABSTRACT

Blockchain systems, with the characteristics of decentralization, irreversibility and traceability, have attracted a lot of attentions recently. However, the current performance of blockchain is poor, which becomes a major constraint of its applications. Additionally, different blockchain systems lack standard performance monitoring approach which can automatically adapt to different systems and provide detailed and real-time performance information. To solve this problem, we propose overall performance metrics and detailed performance metrics for the users to know the exact performance in different stages of the blockchain. Then we propose a performance monitoring framework with a log-based method. It has advantages of lower overhead, more details, and better scalability than the previous performance monitoring approaches. Finally we implement the framework to monitor four well-known blockchain systems, using a set of 1,000 open-source smart contracts. The experimental results show that our framework can make detailed and real-time performance monitoring of blockchain systems. We also provide some suggestions for the future development of blockchain systems.

KEYWORDS

Blockchain, Smart Contract, Performance, Monitoring

ACM Reference Format:
Peilin Zheng, Zibin Zheng, Xiapu Luo, Xiangping Chen, and Xuanzhe Liu. 2018. A Detailed and Real-time Performance Monitoring Framework for Blockchain Systems. In *ICSE-SEIP '18: 40th International Conference on Software Engineering: Software Engineering in Practice Track, May 27-June 3, 2018, Gothenburg, Sweden.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3183519.3183546

1 INTRODUCTION

Blockchain, originated from Bitcoin [26], is a continuously growing list of records, called blocks, which are linked and secured using cryptography. A blockchain is maintained by peers in the P2P transaction network, where peers record transactions in a period of time and packaged them together into a block to join the blockchain. Blockchain is decentralized, tamper-resistant, and traceable. On blockchain, a smart contract is an event-driven promise defined by the programming language, which no body controls and therefore everybody can trust. As shown in Figure 1, a smart contract on blockchain can be invoked in a way of sending a "transaction" (named *invoking transaction* which includes the address of the contract, the calling function, and the parameters) to the validating peers. After that, the smart contract will be executed independently by every peer [7]. Finally different peers reach the consensus through a consensus protocol and save the execution result into the blockchain. Since blockchain-based smart contracts can enhance trustworthy, decrease the cost of central trusted authority, and have wide applications (e.g., finance [17], supply chain [20], IoT [13] etc.), they have gained a lot of attentions from both industry and academic in recent years [21, 35].

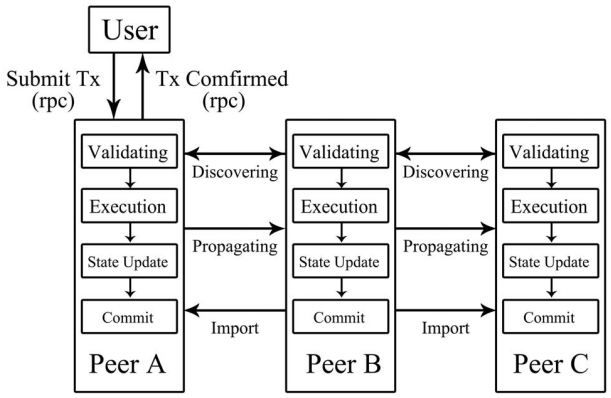

Figure 1: An Example of Smart Contract Invocation on Blockchain

Performance is one of the most important issues of various blockchain systems. Current blockchain systems (e.g., Ethereum [7], Parity, Fabric [8], etc.) suffer from various performance problems, especially when executing complex smart contracts. Real-time performance monitoring of blockchain systems is urgently needed. We divide the performance of blockchain systems into *overall performance* and *detailed performance*. Overall performance

ICSE-SEIP '18, May 27-June 3, 2018, Gothenburg, Sweden

ACM ISBN 978-1-4503-5659-6/18/05…$15.00
https://doi.org/10.1145/3183519.3183546

(e.g., throughput, latency) can be employed to select the optimal blockchain system to fit the actual application scenario so that it is valuable for users. The detailed performance provides more detailed information of the whole process which is valuable for blockchain developers to know the performance bottlenecks.

There are some studies [16, 32] focusing on the overall performance evaluation of blockchain systems. However, overall metrics cannot reflect the detailed performance in different process stages as shown in Figure 1. Detailed performance information of blockchain is urgently required and metrics are lacking. Moreover, the overhead of real-time monitoring as well as scalability of the monitoring framework need to be comprehensively investigated. Thus the challenges of performance monitoring for blockchain systems can be summed up as what to monitor and how to monitor.

To attack the above challenges, we propose a detailed and real-time performance monitoring framework for blockchain systems. The framework gets the performance data in real time by log analysis and daemon process. The proposed monitoring framework has the following advantages: (1) **Low overhead**: The framework has negligible performance impact on the running blockchain systems, and thus is suitable for real-time monitoring. Our experiments show that the overhead is 95% less than the previous RPC-based monitoring method; (2) **Detailed monitoring**: Via log analysis, we can get more detailed information about different stages of the blockchain system. Our framework can monitor 12 different performance metrics; and (3) **Scalability**: The framework can be easily extended to monitor new peers or blockchain systems.

The contribution of this paper is three-fold:

- This paper proposes various overall and detailed performance metrics for different blockchain systems.
- A scalable framework is proposed for detailed and real-time monitoring of blockchain systems, which has much lower overhead compared with previous approaches.
- We collect and classify 1,000 open-source smart contracts to conduct comprehensive performance monitoring and evaluation of 4 well-known blockchain systems. The experimental result shows the feasibility of our proposed framework. Moreover, we come out with some suggestions for the further blockchain development.

The rest of this paper is organized as follows. Section 2 introduces the related work. Section 3 describes the basic concepts. Section 4 introduce the performance metrics. Section 5 details the framework of performance monitoring. Section 6 shows the experiments of our framework on different blockchain systems and Section 7 concludes the paper.

2 RELATED WORK

Nakamoto described how to build up a peer-to-peer decentralized electronic cash system called Bitcoin without any credit basis in 2008 [26], which has been proved to be stable but inefficient [15]. In 2013, Vitalik et al. developed Ethereum [7], which has big progress compared with the Bitcoin blockchain. Ethereum has a turing-complete virtual machine to execute smart contracts. In 2015, IBM developed Fabric, executing smart contracts (chaincode) in the docker [8]. In 2016, Onchain developed Antshare blockchain, using smart contracts to record the transferring of the digital asset. Antshare blockchain is reputed to handle 10,000 transactions per second. Qtum presented a smart-contract framework that aims for sociotechnical application suitability [14]. Different blockchain systems use different consensus protocols, code execution engines, and so on. Vukolic compares the execution efficiency of different consensus protocol including PoS and PoW [31]. Idelberger et al. proposed logic-based smart contracts for blockchain systems [18].

Blockchain-based smart contracts have a wide range of applications. McCorry et al. proposed an open board voting system which maximizes the privacy of the voters [24]. Christidis et al. investigated the application and limitations of smart contracts for Internet-of-Things, indicating that the performance and efficiency is too low to execute the contracts on IoT [13]. The applications above use the blockchain-based smart contract for its decentralized, tamper-resistant, and traceable features. However, all of them have the same problem of long execution time and low efficiency, which makes it still unable to replace the centralized solution. Performance is one of the key factors restricting the application of blockchain-based smart contacts.

There are some studies about black-box monitoring for distributed systems, such as Project5 [3], WAP5 [27] and the Sherlock system [5]. But these monitoring approaches cannot be used directly for blockchain systems. Dinh et al. proposed Blockbench [16], an evaluation framework for analyzing private blockchains. It serves as a fair means of comparison for different private blockchains. Weber et al. provided an analysis of the issues that can negatively impact commit times in permissionless proof-of-work blockchains, and provided a way to limit the effect [32]. Kalodner et al. present BlockSci [19], an software platform for blockchain analysis focus on the transaction data. Luu et al. proposed four kinds of vulnerabilities of smart contracts and a tool to detect the vulnerabilities based on symbolic execution [22]. Chen et al. proposed a method to detect Ponzi schemes on public blockchain [12]. Bhargavan et al. proposed a formal verification approach of smart contracts [6]. Marino et al. set some standards for altering and undoing smart contracts when the contracts are being attacked [23]. Chen et al. designed a tool to optimize the smart contracts on Ethereum [10].

These previous research did not provide detailed performance monitoring in different process stages. Moreover, the RPC method used by previous research could not meet the requirement of low overhead and has poor scalability. In this paper, we propose a detailed and real-time performance monitoring framework for blockchain systems to solve above problems.

3 BASIC CONCEPTS

This section introduces the basic principles and concepts of blockchain, smart contract, and common blockchain systems.

3.1 Blockchain

Blockchain was firstly proposed as a kind of data structure for peer-to-peer Bitcoin payments [26]. In a P2P transaction network, each peer called miner packaged the transactions into blocks and then joint them to a chain-like data structure named blockchain. Each peer has a local blockchain and they keep it the same via the consensus protocol. Thus, in a narrow sense, blockchain is a

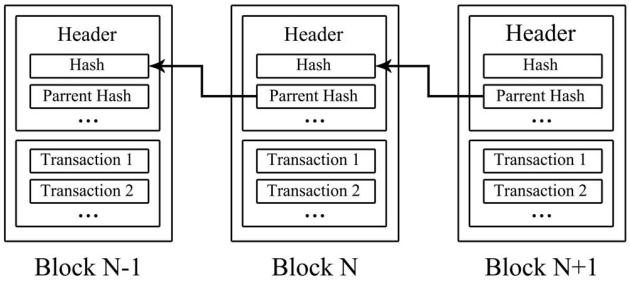

Figure 2: Data Structure of Blockchain

decentralized database which is kept by all the peers in the P2P transaction network. On the other hand, blockchain can also be considered as a distributed ledger kept by peers [29]. Important concepts of blockchain include:

- **Transaction:** A transaction represents an operation of the ledger, such as transferring money. Taking Bitcoin blockchain as an example, if someone wants to send Bitcoin to others, the owner of the money is represented by a address generated by his public key. The owner makes a signature through the corresponding private key. Then the owner package the signature with other information (e.g., value, receiver) into a data structure named transaction. After that, the transaction is broadcast to the network. Every peer that receives the transaction will validate the transaction through the signature in it.
- **Block:** Block is used to record a set of transactions occurred in a period of time. Each block consists of two parts: the header and the content. The block header records the basic information (e.g., hash, parent hash, Merkle-tree root, timestamp, difficulty, nonce, etc.). The block content records the number of transactions and the transaction details.
- **Chain:** Chain includes a number of blocks which are linked together. For example, in Bitcoin blockchain, each block is identified with a hash value. Every block is generated after the previous one so that they record the hash of the previous block, called parent hash. Therefore, with the parent hash recorded in the block header, we can search a sequence from the last block to the first block. The chain-like structure is similar to the list structure, as shown in Figure 2.
- **Consensus:** Consensus protocol is a peer-to-peer protocol that is run by the peers/miners to secure and maintain the blockchain. In a P2P transaction network, due to the high network latency, the order of transactions received by each peer may not be exactly the same. Therefore, the blockchain system needs to design a protocol to reach consensus on the transaction order. For example, Bitcoin blockchain uses PoW (Proof of Work) and Fabric blockchain uses PBFT (Practical Byzantine Fault Tolerance [9]) as the consensus protocol.

3.2 Smart Contract

Smart contract, proposed by Nick Szabo [30], is a promise that is defined by digital form. A blockchain-based smart contract can be considered as an event-driven program which stores its state on a distributed ledger. This program can be run by peers who maintain the ledger independently. Benefiting from the characteristics of blockchain, blockchain-based smart contracts [33] are decentralized, tamper-resistant, traceable, and distributed-executable, which make it reliable to transfer assets on the blockchain. Taking Hyperledger Fabric [4, 8] blockchain (v0.6) as an example, the key concepts of smart contracts are introduced as follows.

- **Chaincode:** Chaincode, or so-called contract code is the code stored on the blockchain with the world state. The chaincode can be read and executed in the virtual machine (e.g. Ethereum Virtual Machine) or container (e.g. docker). The chaincode is compiled by the owner and then put into a transaction to deploy onto peers in the P2P transaction network. Chaincode defines the main execution logic of the smart contract.
- **World State:** World state is a key-value database. Smart contracts use this database to store related states of transactions and the information of each account (e.g., balance, number of tokens, etc.). Most blockchain systems put the world state into a data structure of tree to record the root onto the blockchain (e.g., Merkle Patricia Tree in Ethereum, Bucket Tree in Fabric).
- **Ledger:** Ledger consists of the blockchain and the world state. Blockchain records the history of the ledger and the world state records the balance of each account in the ledger.
- **Validating Peer:** Validating peers are core peers in the blockchain P2P transaction network that maintain the ledger, participate in the consensus, validate transactions, execute the chaincode, and update the state.
- **Non-validating Peer:** Non-validating peers only respond to REST requests from clients. Non-validating peers do not commit the block but propagate the transaction to the validating peers after basic validation.
- **Deployment:** After a smart contract is signed by every participant, it will be included to a transaction in the format of creation code of the chaincode. Then the transaction will be propagated, validated and committed in the blockchain. When the transaction is confirmed, the chaincode will be stored on the blockchain with the world state. This action is called the deployment of the smart contract.
- **Invoke:** Invoke refers to the action that sending a transaction including the information of the function and corresponding parameters of a smart contract. When peers receive the transaction, they will read the chaincode and execute the specified function independently.

3.3 Common Blockchain Systems

There are many blockchain systems. The most common ones are introduced as follows:

- **Ethereum** is the most popular blockchain system with smart contracts. It is a public blockchain with a built-in fully fledged turing-complete programming language that can be used to create "contracts" that can be used to encode arbitrary state transition functions, allowing users to create any of

the decentralized systems. It uses the consensus protocol of PoW (Proof of Work).

- **Hyperledger Fabric** is a blockchain framework implementation. It allows components, such as consensus and membership services, to be plug-and-play. Hyperledger Fabric leverages container technology to host smart contracts called "chaincode" [8].
- **Parity** is another version of client of Ethereum [2]. It provides another consensus protocol called PoA (Proof of Authority). With this consensus protocol, developers can build up private chains (or so called consortium blockchains) with higher throughput than the PoW chains of Ethereum.
- **CITA** is called Cryptape Inter-enterprise Trust Automation [34]. It adopts a microservice architecture to boost performance of the network. With the microservice architecture, a logical node can be scaled to a cluster of servers. Consensus and transaction execution are decoupled as separate microservices. The consensus service is only responsible for transaction ordering, which can finish independently before transaction execution.

In summary, there are many blockchain systems supporting blockchain-based smart contracts. Performance is one of the key factors restricting the application of these blockchain systems, especially when running complex smart contracts. In the next section, we will propose the performance metrics for blockchain systems.

4 PERFORMANCE METRICS

In this section, we will explain why we consider Gas (the uniform metric for resources consumption in Ethereum) as an unsuitable metric by its one-sidedness and floating. Then, we will propose metrics for performance monitoring of blockchain systems. We divide the metrics into two parts to meet different requirements: the overall metrics for the users and the detailed metrics for the developers.

4.1 Gas: an Unsuitable Metric

In Ethereum and many blockchain systems which use EVM as their execution engine of smart contracts [33], developers define "Gas" as the fundamental network cost unit. Gas is paid by Ether through the exchange rate called gasprice. Gas does not exist outside of the internal Ethereum computation engine; its price is set by the transaction and miners are free to ignore transactions whose gasprice is too low. In public blockchain such as Ethereum, miners used Gas to evaluate the performance and resource consumption of a smart contract. However, our research found that Gas is an unsuitable metric for performance monitoring for the following reasons:

A. One-sidedness

As Figure 1 shows, there are several stages in the process of a transaction on the blockchain. Gas is a cost unit only used in the execution stage and sometimes the state update stage. But Gas does not reflect the cost of other stages (e.g., validating, block committing, and so on). In other words, if a miner wants to charge someone that sends him a smart contract transaction, with the metric of Gas, he can only estimate the cost in the execution, without any other metrics to cover the fees during the RPC-response or transaction validating stage.

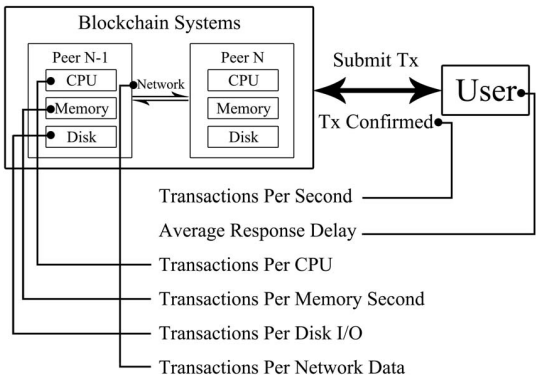

Figure 3: Overall Metrics Corresponding to the System

It is unfair for the blockchain miners because some malicious users could send lots of invalid transactions to them to consume their computing resources in validating the transactions without any cost. Therefore, Gas is one-sided as a metric.

B. Inaccurate

Gas is paid for by Ether exclusively, but the exchange rate called gasprice between Gas and Ether is floating case by case. Thus it could be floating for the blockchain miners to charge the users by Gas. Chen et al. conducted an experiment to execute EVM operations and then recorded the time consumptions [11]. Their results show that the latest Gas costs are not proportional to the consumptions of CPU resources. For example, DIV (division) has the same Gas of SDIV (signed division), but the execution time needed by DIV is about 23% of that needed by SDIV. Therefore, using Gas to evaluate the consumed computing resources of smart contracts is inaccurate.

4.2 Overall Performance Metrics of Blockchain

For the users or managers of the blockchain, it is important for them to know an overview of the performance of the blockchain. Combining the data of the blockchain and the resources consumption, we propose the overall performance metrics for blockchain as shown in Figure 3.

Transactions Per Second

Different blockchain systems show different speeds for deploying, invoking, and executing smart contracts. We need to monitor the throughput in a period of time, which is the number of transactions per second.

During a period of time from t_i to t_j, *Transactions Per Second* of $peer_u$ can be calculated by the following equation, (we abbreviate transaction as *Tx*, the same below):

$$\mathrm{TPS}_u = \frac{Count\left(Tx\ in\left(t_i, t_j\right)\right)}{t_j - t_i}(txs/s). \tag{1}$$

When considering the throughput of N peers, we can take the average by:

$$\overline{TPS} = \frac{\sum_u TPS_u}{N}(txs/s). \tag{2}$$

Average Response Delay

There is a gap between the time when transaction is firstly sent into the network and the time when it is confirmed (e.g., be committed in a block and the block is accepted by all peers). If users deploy a smart contract on the blockchain, users have to wait some time until the contract is confirmed, and then the contract can respond for other operation. In other words, compared with the centralized contracts or softwares, smart contracts on blockchain improve the reliability at the expense of increased response delay, and hence we need to monitor this expense.

During a period of time from t_i to t_j, the action of each transaction firstly sent to the peer is marked as Tx_{input} and the action when Tx is confirmed is marked as $Tx_{confirmed}$. *Average Response Delay* of $peer_u$ can be found by the following equation:

$$ARD_u = \frac{\sum_{Tx}(t_{Tx\ confirmed} - t_{Tx\ input})}{Count\left(Tx\ in\left(t_i, t_j\right)\right)}(txs/s). \quad (3)$$

When we consider to the response delay of all smart contracts, we can take the average by:

$$\overline{ARD} = \frac{\sum_u ARD_u}{N}(txs/s). \quad (4)$$

Transactions Per CPU

During the execution of smart contracts, it consumes a lot of CPU resources. The degree of CPU consumption is determined by the business logic implemented in the contract. The contract with encryption, loops will consume a lot of CPU resources. The action of committing the block, computing the hash of the world state also consumes a lot of CPU resources. Note that different blockchain systems are running on the peers equipped with different CPUs. Thus we need a metric to monitor the utilization of the CPU when running the smart contracts, and hence we propose it as *Transactions Per CPU*.

During a period of time from t_i to t_j, *Transactions Per CPU* of $peer_u$ can be computed by the following equation:

$$TPC_u = \frac{Count\left(Tx\ in\left(t_i, t_j\right)\right)}{\int_{t_i}^{t_j} F * CPU(t)}(txs/(GHz \cdot s)), \quad (5)$$

where F is the frequency of a single CPU core and *CPU(t)* is the CPU usage of the blockchain program at t. When considering the whole utilization of CPUs in the network, we can take the average by:

$$\overline{TPC} = \frac{\sum_u TPC_u}{N}(txs/(GHz \cdot s)). \quad (6)$$

Transactions Per Memory Second

During the contract execution, the virtual machine/docker will load the relevant account data from the world state and open up some arrays. These operations will consume the memory. Thus we propose *Transactions Per Memory Second* to represent the utilization of the memory by the following equation:

$$TPMS_u = \frac{Count\left(Tx\ in\left(t_i, t_j\right)\right)}{\int_{t_i}^{t_j} RMEM(t) + VMEM(t)}(txs/(MB \cdot s)), \quad (7)$$

where *RMEM(t)* is the real memory used by the blockchain program at t and *VMEM(t)* is the virtual memory of it. When considering the whole the network, we can take the average by:

$$\overline{TPMS} = \frac{\sum_u TPMS_u}{N}(txs/(MB \cdot s)). \quad (8)$$

Transactions Per Disk I/O

The blockchain program has separate storage space in the hard disk for storing the data including the world state. Moreover it consumes the resources of I/O while maintaining the blockchain (e.g., block committing, contract execution). Similar to the *TPMS*, we propose *Transactions Per Disk I/O* to represent the utilization of I/O by the following equation:

$$TPDIO_u = \frac{Count\left(Tx\ in\left(t_i, t_j\right)\right)}{\int_{t_i}^{t_j} DISKR(t) + DISKW(t)}(txs/kilobytes), \quad (9)$$

where *DISKR(t)* is the size of the data read from the disk in the second t and *DISKW(t)* is the size of the data written into the disk. To give an overview of the disks of all the peers, we take the average by:

$$\overline{TPDIO} = \frac{\sum_u TPDIO_u}{N}(txs/kilobytes). \quad (10)$$

Transactions Per Network Data

Different blockchain systems use different consensus protocol to keep every peer in the same state. The consensus protocol will transfer the data of blocks or appending transactions in the network, thus it consumes the network flow. During a period of time from t_i to t_j, we propose *Transactions Per Network Data* to monitor this consumption of network flow by the following equation:

$$TPND_u = \frac{Count\left(Tx\ in\left(t_i, t_j\right)\right)}{\int_{t_i}^{t_j} UPLOAD(t) + DOWNLOAD(t)}(txs/kilobytes), \quad (11)$$

where *UPLOAD(t)* is the size of upstream in the network at t and *DOWNLOAD(t)* is the size of downstream. Considering the whole network flow, we can take the average by:

$$\overline{TPND} = \frac{\sum_u TPND_u}{N}(txs/kilobytes). \quad (12)$$

4.3 Detailed Performance Metrics of Blockchain

As shown in the Figure 1, we divide transaction process into several subprocesses. For the users of the blockchain, it is enough for them to know the overall performance by the metrics we have proposed in the above subsection. However, for developers of the blockchain, it is essential to know what is going on in the whole blockchain. Moreover, they need to know how the performance is in each stage. So we propose the metrics corresponding to each important stage of the process, as Figure 4 shows.

Peer Discovery Rate

In public blockchain, a peer may change the peers which are connected to it because of the block height or the QoS. In private blockchain or permissioned blockchain, although the connections in the network are stable in most of time, there are still some peers to be joined or removed in some special case. In the process of peer discovery, we use *Peer Discovery Rate* to monitor the speed that

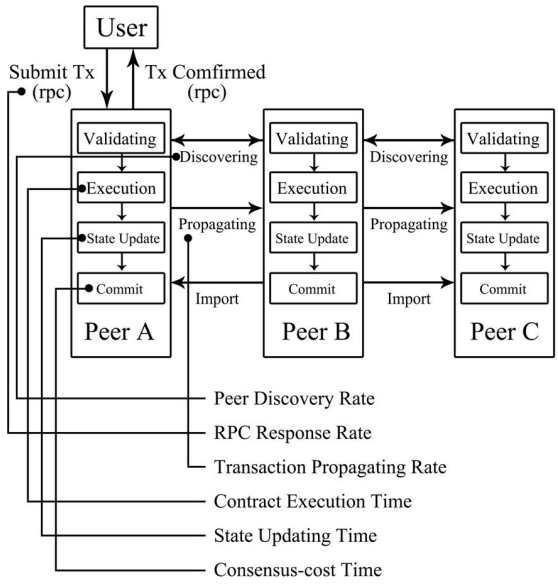

Figure 4: Detailed Metrics Corresponding to the Stages of Process

peers are added into the network by the following equation:

$$\mathrm{PDR}_u = \frac{P_u}{MAX(t_{peer\ added}) - MIN(t_{peer\ pong})}(peers/s), \quad (13)$$

where P_u is the number of the peers connected to peer u, $t_{peer\ added}$ denotes the timestamp when a new peer is added to u (so the maximum of $t_{peer\ added}$ denotes the time when the last peer is added), $t_{peer\ pong}$ denotes the timestamp when a new peer responds to the ping from u (so the minimum of $t_{peer\ pong}$ denotes the time when the first peer get into the discovery process). Considering to the whole network, we can take the average by:

$$\overline{PDR} = \frac{\sum_u PDR_u}{N}(peers/s). \quad (14)$$

RPC Response Rate

Most of the blockchain systems use RPC-API as their interactive means with the user. Therefore when we consider the limit of the transactions, we should not only focus on the limit of the execution, but also focus on the external interfaces. During a period of time from t_i to t_j, *RPC Response Rate* of peer_u can be computed by the following equation:

$$\mathrm{RRR}_u = \frac{Count\left(RPC\ in\left(t_i, t_j\right)\right)}{t_j - t_i}(resps/s), \quad (15)$$

where *RPC* is an RPC response from peer_u at the second *t*. We can take the average to know the whole system by:

$$\overline{RRR} = \frac{\sum_u RRR_u}{N}(resps/s). \quad (16)$$

Transaction Propagating Rate

When a transaction is sent to a peer, before it is mined in a block, it would be propagated (or so-called promoted) to other peers [25]. Hence the limit of the transaction propagating might restrict the performance of the blockchain system. We propose *Transaction Propagating Rate* to see how transaction is promoted by the following equation:

$$\mathrm{TPR}_u = \frac{\sum_t TxPromoted_u(t)}{t_j - t_i}(txs/s), \quad (17)$$

where $TxPromoted_u(t)$ is the number of transactions promoted from peer_u at the second *t*. We can also take the average by:

$$\overline{TPR} = \frac{\sum_u TPR_u}{N}(txs/s). \quad (18)$$

Contract Execution Time

In blockchain systems, the execution time depends on the business logic implemented by the smart contracts. We propose *Contract Execution Time* to see how transaction executed by the following equation:

$$\mathrm{CET}_u = \frac{\sum_{Tx}(t_{Tx\ EXEDONE} - t_{Tx\ EXESTART})}{Count\left(Tx\ in\left(t_i, t_j\right)\right)}(s/tx), \quad (19)$$

where $t_{Tx\ EXESTART}$ denotes the time when the transaction execution is started and $t_{Tx\ EXEDONE}$ denotes the time when the transaction execution is finished. We can also take the average by:

$$\overline{CET} = \frac{\sum_u CET_u}{N}(s/tx). \quad (20)$$

State Updating Time

In Ethereum, after executing the contract , the world state is changed. Then state will be updated with pending changes [1]. Therefore we propose *State Updating Time* by the following equation:

$$\mathrm{SUT}_u = \frac{\sum_{Tx}(t_{Tx\ STATEDONE} - t_{Tx\ EXEDONE})}{Count\left(Tx\ in\left(t_i, t_j\right)\right)}(s/tx), \quad (21)$$

where $t_{Tx\ EXEDONE}$ denotes the time when the transaction execution is finished and $t_{Tx\ STATEDONE}$ denotes the time when the state updating is finished. We can also take the average by:

$$\overline{SUT} = \frac{\sum_u SUT_u}{N}(s/tx). \quad (22)$$

Consensus-Cost Time

Although blockchain systems always use different consensus protocol, we can regard them as the layer of consensus. In this layer, the main function is to package the transactions with the results into the block then propagate it to other peers to be confirmed. Hence we consider the time from when the transaction is firstly processed to when the block including the transaction is confirmed as the *Consensus-cost Time* by the following equation:

$$\mathrm{CCT}_u = \frac{\sum_{Tx}(t_{Tx\ confirmed} - MAX_{block}(t_{Tx\ EXEDONE}))}{Count\left(Tx\ in\left(t_i, t_j\right)\right)}(s/tx), \quad (23)$$

where $t_{Tx\ confirmed}$ denotes the time when the transaction is confirmed and $MAX_{block\ i}(t_{Tx\ EXEDONE})$ denotes the max value of timestamp of the state updating finished of the transaction in the same block. In other words, $MAX_{block\ i}(t_{Tx\ EXEDONE})$ is the timestamp of the execution finishing of the last transaction in the same block as transaction *Tx*. We can also take the average to know the whole system by:

$$\overline{CCT} = \frac{\sum_u CCT_u}{N}(s/tx). \quad (24)$$

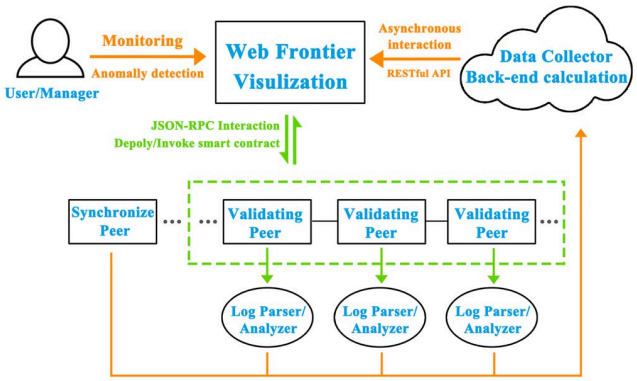

Figure 5: Performance Monitoring Framework

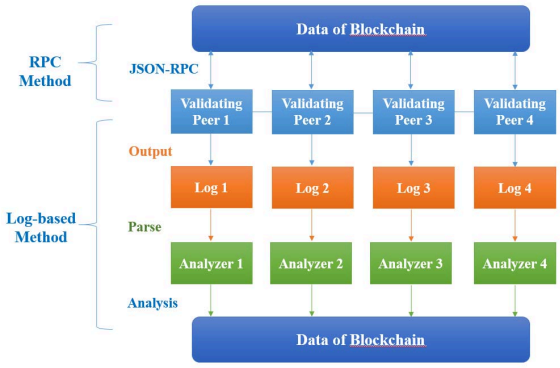

Figure 6: RPC Method and Log-based Method

5 FRAMEWORK OF PERFORMANCE MONITORING

In this section, we first give an overview of our monitoring framework for the blockchain systems in 5.1. Then we will detail the log-based method and the traditional RPC method of blockchain monitoring in 5.2.

5.1 Overview

Figure 5 shows the performance monitoring framework. It uses a low coupling architecture to make it efficient and low-overhead. Next we will introduce every part of it.

- **Validating Peer** is the specific object of the blockchain monitoring. It participates in the process of maintaining the blockchain. Since it is the underlying object of the contract execution, we focus on its performance to speculate on the performance of the entire blockchain system.
- **Log Parser/Analyzer** is a terminal corresponding to each validating peer. In this part, we create a daemon process to collect and parse the logs printed by the validating peer. We also implement the hardware resource monitoring in this part. The main goal of this part is to collect the data about the blockchain and hardware consumption (e.g., count of transactions, CPU usage). We need get second-level data because we want to make real-time monitoring. And we also need to get the data with low overhead. Hence we propose a log-based method for monitoring (We will discuss it later in section 5.2.). In some cases, the overhead of the log parser/analyzer is small enough to be deployed directly on the validating peer.
- **Synchronize Peer** is the peer which is participating in keeping the blockchain. It is similar to the non-validating peer (we have talked about it in section 3.2). In some special cases, we cannot collect all the data we want in the logs from the validating peer. For example, in the client called GETH (a Golang implement of Ethereum), we cannot get the timestamp when the transaction is confirmed in the block from the logs. Thus we need to get this information from an RPC query request (e.g., web3.eth.gettransaction(hash)). But we do not want this request to make extra overhead with the blockchain (Test it later in section 6.1). Hence we need a peer from which we can get the information without affecting on the blockchain.
- **Data collector/calculator** collects the data from each log analyzer and checkout the synchronize peer in a period of time. Then it calculates every performance metrics. The developer can store the data in the memory or write it into the database. In some situation, storing the data in the memory is enough and more lightweight. To output the result of the calculation, we design some RESTful interfaces. The RESTful interfaces also satisfy the asynchronous requests, which make it easier to display the performance in the front-end visualization.
- **Web Frontier** is the visualization that displays the data of performance from the metric calculator. Users can monitor the blockchain easily through the visualization. Furthermore, the managers or developers can use it to do some optimization or anomaly detection.

5.2 Log-based Method: Source Data Acquisition

In this subsection we detail the log parser/analyzer and the method of source data acquisition. Figure 6 shows the processes of RPC method and log-based method.

RPC method is a way that users interact directly with the validating peer. Most of the research up to now use the RPC method to get the data of the blockchain (e.g., the count of transactions). However, this kind of data acquisition is quite inefficient, as it introduces big overhead onto the peer itself in the situation of real-time monitoring. Moreover, the consumption resource for calling the JSON-RPC interface would be counted into the consumption of the blockchain program itself, making the monitoring inaccurate. Moreover the RPC method can only get a little information about the blockchain system, which is not enough for us to measure all the metrics we propose in section 4.

In distributed systems, developers usually use the log to trace the data. (e.g., in Google, developers use Dapper [28] to analysis the log and trace the data in their servers) Inspired by the this, we collect the data of the blockchain through a log-based method.

As Figure 6 shows, we set up an analyzer for each log from each validating peer of the blockchain system. As Figure 7 shows, we

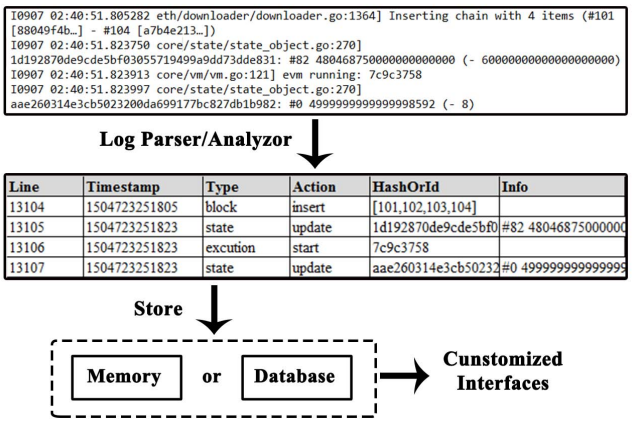

Figure 7: Detailed View of Log-based Method

parse the log and then distribute the log to different types (e.g., block, transaction, execution, state). There are many kinds of logs among the different blockchain systems, but we can extract the uniform or similar actions. We propose a tuple of **<line, timestamp, type, action, hashOrId, info>** to record every common actions in different blockchain systems. In this way, not only can we get more details about the process of the blockchain, but also monitor different blockchain systems by the same metrics.

In the data processing, we can easily store the data in the memory or database. Then we provide some customized interfaces to make it easy for the data calculator and front-end visualization. In this way, we can decrease the times of request. For example, in the period of one second, if there are 100 blocks generated per second and we want to know their timestamps, the log-based method can transfer the data in one request but the RPC method usually needs 100 requests (e.g., web3.eth.getBlock). Moreover, we also avoid the resource consumption of the RPC method to be counted in the blockchain program.

In summary, compared with the RPC-method, our log-based method has the following advantages

- Low overhead: The monitoring system has negligible performance impact on blockchain services.
- Detailed monitoring: It allows users to know how the resources are consumed in every stage and help the developers to optimize the performance.
- Scalability: It is easy to be extended to monitor new blockchain systems or peers in the situations where the number of the peers may increase or decrease.

6 EVALUATION

We implement our framework by Node.js and conduct experiments to answer three research questions:

Question 1: How is the overhead caused by the log-based method, compared with the RPC method?

Question 2: How is the overall performance between different blockchain systems and smart contracts?

Question 3: How is the detailed performance of the blockchain in different situations?

Table 1: Change of Throughput with Log-based Method and RPC Method

Requests	Normal	Log-based	RPC
500	6.8400	6.4406(-5.83%)	6.1904(-9.49%)
1000	6.8552	6.8269(-0.41%)	5.6436(-17.67%)
2000	6.1566	6.0166(-2.27%)	5.5366(-10.07%)
3000	5.0854	5.0266(-1.15%)	3.6200(-28.81%)

In section 6.1 ,we measure the overhead of the log-based method and RPC method to answer Q1. For Q2, we run and monitor the most popular blockchain systems: Ethereum (v1.5.9), Parity (v1.8.9), Fabric (v0.6), CITA (v0.13). Then we collect and classify different kinds of smart contracts, and then deploy and invoke them on Ethereum to measure the overall metrics of them. For Q3, we run Ethereum in different situations to monitor its detailed performance, with 1,000 open-source contracts collected from the real world public chain. All the evaluations are conducted on the peers with i7-4790 3.60GHz CPU, 8GB RAM. We will upload the configuration files and logs during the evaluation onto our website[1]. The experimental results show that our framework can make detailed and real-time performance monitoring of blockchain systems.

6.1 Overhead of the Log-based Method

In order to compare the overhead of log-based method and RPC method, we use both of them on the same blockchain system using the GETH (Ethereum client). First, we broadcast transaction requests in the batch of 500, 1,000, 2,000 and 3,000 to measure the throughput without any real-time monitoring. (We calculate the TPS after all the transactions have been confirmed.) That is the normal throughput. Then we repeat the experiment twice, one is monitoring the transactions with the log-based method and the other with the RPC method. By doing so, we can see how the throughput changes to compare the overhead. The result is shown in Table 1, showing that the change of throughput with log-based method is much lower than the one with RPC method. In the experiment with 1,000 requests, the impact of log-based method is 95% less than the impact of RPC method. We also record the usage of computing resources (CPU, memory, network socket and disk read) with the 500 transaction queries per second. The overhead of log-based method and RPC method is shown in Figure 8, we can see that the overhead of log-based method is much lower and even negligible.

6.2 Overall Performance

We monitor the most popular blockchain systems: Ethereum, Parity, Fabric, CITA with running a DoingNothing [16] contract to see the overall performance metrics. The result is shown in Table 2. We can see that Farbic and CITA have larger *Transactions Per Second* than others, since they are mainly designed as the permissioned blockchain and the consensus protocol is much faster. For Ethereum and Parity-PoW, their utilization of computing resources (*Transactions Per CPU*, *Transactions Per Memory Second* is quite low since

[1] http://ibase.site

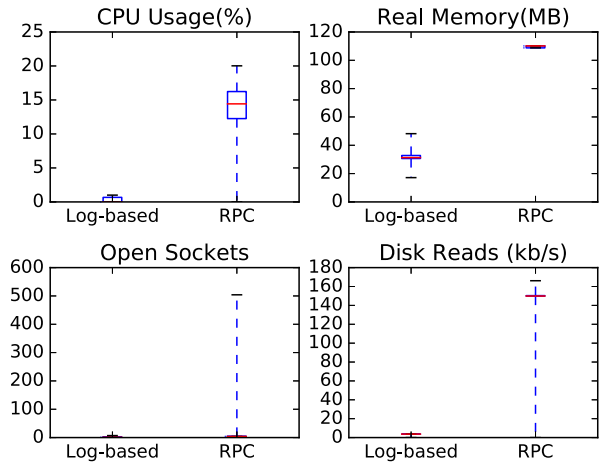

Figure 8: Overhead of Resources of Log-based Method and RPC Method

Table 2: Overall Performance of Different Blockchain Systems

Blockchain	TPS	TPC	TPMS	TPDIO	TPND
Ethereum	5.55578	0.00195	0.01060	0.26573	0.22206
Parity-PoW	3.95500	0.00140	0.06814	0.00264	0.07167
Fabric	600.611	2.65340	4.28265	0.13816	0.10122
CITA	256.636	1.33393	0.43244	0.59888	0.16208

the PoW consensus protocol make them use lots of computing resources on the hash computing. The *Transactions Per Network Data* of CITA and Fabric is lower than Ethereum because the network consumption of PBFT [9] and microservice architecture of CITA might be high.

For different smart contracts, we first collect and classify them, and then deploy and invoke them on Ethereum to measure the overall metrics of them. The result is shown in Table 3. We can see that the contracts which are mainly used to store the arrays have the fast throughput but the lower *TPDIO*, which is caused by the world state read and written frequently into the disk. The contracts with many loop operations have the lowest *TPC* and *TPMS*, resulted from the great consumption of computing resources by the loop operations. Further more the contracts with many account-related operations (Acc10000 and Acc50000) also consume lots of computing resources since the operations to the accounts will cause the hash computing of the world state. We will not describe the *ARD* here because it will be more meaningful to test it in the WAN rather than the LAN.

In this experiment, we observe that the blockchain with PoW consensus protocol shows lower overall performance than others. Thus we suggest the users to avoid using PoW blockchain systems in the high frequency trading scene.

Table 3: Overall Performance of Different Smart Contracts on Ethereum

Contract	TPS	TPC	TPMS	TPDIO	TPND
Store	7.93134	0.00347	0.01849	0.02569	0.40015
Loop	2.40587	0.00276	0.00629	0.04130	0.19420
Acc10000	2.82720	0.00107	0.00735	0.04195	0.14316
Acc50000	0.56219	0.00023	0.00197	0.00852	0.04876

Table 4: Detailed Performance of Ethereum

Peers (peers)	PDR (peers/s)	RRR_{max} (resps/s)	TPR_{max} (txs/s)	CET (s/tx)	SUT (s/tx)	CCT (s/tx)
Pub-1	0.06058	60.606	-	0.0002	0.0001	-
Pri-1	-	81.677	-	0.0006	0.0003	9.55
Pri-2	333.30	73.313	72.0	0.0009	0.0007	5.13
Pri-4	333.33	86.910	90.0	0.0011	0.0007	3.49

6.3 Detailed Performance

To measure the detail metrics of Ethereum, we run GETH (Ethereum client) in different environments with different number of peers to monitor its detailed performance using 1,000 open-source contracts collected from the real world public chain. The result is shown in Table 4. In this experiment, we first set up one peer in the public network, which is synchronized with the public blockchain of Ethereum to monitor its performance. Then, we run a private blockchain of Ethereum in the LAN environment with different number of peers. As shown in Table 4, the peer of the public chain shows much lower *Peer Discovery Rate* than the peers in private chain, because the peers of private chain are known by each other in the beginning and the network environment is faster in LAN than Internet. We deploy the same 1,000 smart contracts in the private blockchain, and observe that the *Contract Execution Time* and *State Updating Time* are nearly unchanged. But the *Consensus-Cost Time* is lower in the system of more peers because the hash computing at the same difficulty is shared by all the peers to make it faster.

In this experiment, we use our framework to find out the bottlenecks of Ethereum by monitoring the detailed performance. Since peer discovery, transactions propagating and consensus-cost mainly restrict the performance of Ethereum, developers might optimize the network module and consensus module to improve the performance of Ethereum.

For more information, please visit http://ibase.site to get all the configuration and results.

7 CONCLUSION AND FUTURE WORK

This paper proposes overall and detailed performance metrics for different blockchain systems. We also propose a scalable framework for detailed and real-time monitoring of blockchain systems, which has much lower overhead and more details about the blockchain systems compared with previous approaches. We collect and classify 1,000 open-source smart contracts to conduct comprehensive performance monitoring and evaluation of 4 well-known blockchain

systems. The experimental result shows the feasibility of our proposed framework. Moreover, we come out with some suggestions for the further blockchain development.

We will upload our implement and logs during the evaluation on http://ibase.site. In the future, our work can be extended in different aspects: **(1)Monitoring more blockchain systems:** Blockchain is very popular nowadays. More and more different blockchain systems are springing up. There are some new kinds of "blockchain" systems such as IOTA, Corda, which do not have the structure of blockchain but also enable peers to make consensus. And there are many other actions (e.g., permission checking) that could be monitored. **(2)Tracing of blockchain systems:** We used log-based method as a way for performance monitoring. It can also be applied in the tracing of blockchain systems. By this way, developers can make it easier for users to trace the transaction and know the whole process of the transaction. **(3)Anomaly detection:** The log-based method provides lots of structured data. Researchers and developers can easily use the data to detect anomalies (e.g., early warning when a blockchain system is being partitioned).

ACKNOWLEDGMENTS

The work described in this paper was supported by the National Key Research and Development Program (2016YFB1000101), the National Natural Science Foundation of China (61672545, 61722214, 61472338), the Program for Guangdong Introducing Innovative and Enterpreneurial Teams (2016ZT06D211), and the Pearl River S&T Nova Program of Guangzhou (201710010046). Zibin Zheng and Xiangping Chen are the corresponding authors.

REFERENCES

[1] 2017. *Go Ethereum.* https://github.com/ethereum/go-ethereum/.
[2] 2017. *Parity documentation.* https://paritytech.github.io/wiki.
[3] Marcos K Aguilera, Jeffrey C Mogul, Janet L Wiener, Patrick Reynolds, and Athicha Muthitacharoen. 2003. Performance debugging for distributed systems of black boxes. *ACM SIGOPS Operating Systems Review* 37, 5 (2003), 74–89.
[4] E Androulaki, C Cachin, K Christidis, C Murthy, B Nguyen, and M Vukolić. 2016. Hyperledger fabric proposals: Next consensus architecture proposal. (2016).
[5] Paramvir Bahl, Ranveer Chandra, Albert Greenberg, Srikanth Kandula, David A Maltz, and Ming Zhang. 2007. Towards highly reliable enterprise network services via inference of multi-level dependencies. In *ACM SIGCOMM Computer Communication Review*, Vol. 37. ACM, 13–24.
[6] Karthikeyan Bhargavan, Antoine Delignat-Lavaud, Cédric Fournet, Anitha Gollamudi, Georges Gonthier, Nadim Kobeissi, A Rastogi, T Sibut-Pinote, N Swamy, and S Zanella-Beguelin. 2016. Formal verification of smart contracts. In *Proceedings of the 2016 ACM Workshop on Programming Languages and Analysis for Security-PLAS*. 91–96.
[7] Vitalik Buterin et al. 2013. Ethereum white paper. (2013).
[8] Christian Cachin. 2016. Architecture of the Hyperledger blockchain fabric. In *Workshop on Distributed Cryptocurrencies and Consensus Ledgers*.
[9] Miguel Castro, Barbara Liskov, et al. 1999. Practical Byzantine fault tolerance. In *OSDI*, Vol. 99. 173–186.
[10] Ting Chen, Xiaoqi Li, Xiapu Luo, and Xiaosong Zhang. 2017. Under-optimized smart contracts devour your money. In *International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 442–446.
[11] Ting Chen, Xiaoqi Li, Ying Wang, Jiachi Chen, Zhihao Li, Xiapu Luo, Man Ho Au, and Xiaosong Zhang. 2017. An Adaptive Gas Cost Mechanism for Ethereum to Defend Against Under-Priced DoS Attacks. In *Proceedings of Information Security Practice and Experience - 13th International Conference, ISPEC 2017, Melbourne, VIC, Australia, December 13-15, 2017.*
[12] Weili Chen, Zibin Zheng, Jiahui Cui, Edith Ngai, Peilin Zheng, and Yuren Zhou. 2018. Detecting Ponzi Schemes on Ethereum: Towards Healthier Blockchain Technology. In *Proceedings of the 27th International Conference on World Wide Web (Accepted), WWW*. ACM.
[13] Konstantinos Christidis and Michael Devetsikiotis. 2016. Blockchains and smart contracts for the internet of things. *IEEE Access* 4 (2016), 2292–2303.
[14] Patrick Dai, Neil Mahi, Jordan Earls, and Alex Norta. 2017. Smart-Contract Value-Transfer Protocols on a Distributed Mobile Application Platform. (2017).
[15] Christian Decker and Roger Wattenhofer. 2013. Information propagation in the bitcoin network. In *International Conference on Peer-to-Peer Computing (P2P)*. IEEE, 1–10.
[16] Tien Tuan Anh Dinh, Ji Wang, Gang Chen, Rui Liu, Beng Chin Ooi, and Kian-Lee Tan. 2017. BLOCKBENCH: A Framework for Analyzing Private Blockchains. In *Proceedings of the 2017 ACM International Conference on Management of Data*. ACM, 1085–1100.
[17] Ye Guo and Chen Liang. 2016. Blockchain application and outlook in the banking industry. *Financial Innovation* 2, 1 (2016), 24.
[18] Florian Idelberger, Guido Governatori, Régis Riveret, and Giovanni Sartor. 2016. Evaluation of logic-based smart contracts for blockchain systems. In *International Symposium on Rules and Rule Markup Languages for the Semantic Web*. Springer, 167–183.
[19] Harry Kalodner, Steven Goldfeder, Alishah Chator, Malte Möser, and Arvind Narayanan. 2017. BlockSci: Design and applications of a blockchain analysis platform. *arXiv preprint arXiv:1709.02489* (2017).
[20] Henry M Kim and Marek Laskowski. 2016. Towards an ontology-driven blockchain design for supply chain provenance. (2016).
[21] Xiaoqi Li, Peng Jiang, Ting Chen, Xiapu Luo, and Qiaoyan Wen. 2017. A Survey on the security of blockchain systems. *Future Generation Computer Systems* (2017).
[22] Loi Luu, Duc-Hiep Chu, Hrishi Olickel, Prateek Saxena, and Aquinas Hobor. 2016. Making smart contracts smarter. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 254–269.
[23] Bill Marino and Ari Juels. 2016. Setting standards for altering and undoing smart contracts. In *International Symposium on Rules and Rule Markup Languages for the Semantic Web*. Springer, 151–166.
[24] Patrick McCorry, Siamak F Shahandashti, and Feng Hao. 2017. A Smart Contract for Boardroom Voting with Maximum Voter Privacy. *IACR Cryptology ePrint Archive* (2017), 110.
[25] Mahesh Murthy. 2017. *Life Cycle of an Ethereum Transaction.* https://medium.com/blockchannel/life-cycle-of-an-ethereum-transaction-e5c66bae0f6e.
[26] Satoshi Nakamoto. 2008. Bitcoin: A peer-to-peer electronic cash system. (2008).
[27] Patrick Reynolds, Janet L Wiener, Jeffrey C Mogul, Marcos K Aguilera, and Amin Vahdat. 2006. WAP5: black-box performance debugging for wide-area systems. In *Proceedings of the 15th international conference on World Wide Web*. ACM, 347–356.
[28] Benjamin H Sigelman, Luiz Andre Barroso, Mike Burrows, Pat Stephenson, Manoj Plakal, Donald Beaver, Saul Jaspan, and Chandan Shanbhag. 2010. *Dapper, a large-scale distributed systems tracing infrastructure.* Technical Report. Technical report, Google, Inc.
[29] Melanie Swan. 2015. Blockchain thinking: The brain as a dac (decentralized autonomous organization). In *Texas Bitcoin Conference*. 27–29.
[30] Nick Szabo. 1997. The idea of smart contracts. *Nick Szabo's Papers and Concise Tutorials* (1997).
[31] Marko Vukolić. 2015. The quest for scalable blockchain fabric: Proof-of-work vs. BFT replication. In *International Workshop on Open Problems in Network Security*. Springer, 112–125.
[32] Ingo Weber, Vincent Gramoli, Alex Ponomarev, Mark Staples, Ralph Holz, An Binh Tran, and Paul Rimba. 2017. On availability for blockchain-based systems. In *Proceedings of the 36th International Symposium on Reliable Distributed Systems (SRDS). IEEE.*
[33] Gavin Wood. 2014. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum Project Yellow Paper* 151 (2014).
[34] Jan Xie. 2017. *CITA Technical Whitepaper.* https://github.com/cryptape/cita.
[35] Zibin Zheng, Shaoan Xie, Hong-Ning Dai, and Huaimin Wang. 2016. Blockchain challenges and opportunities: A survey. *International Journal of Web and Grid Services, accepted* (2016).