Performance Analysis of a BESU Permissioned Blockchain

Leonardo Mostarda, Andrea Pinna, Davide Sestili, and Roberto Tonelli

Abstract

We present a set of tests on a real permissioned blockchain where nodes are maintained by different independent public organizations in various geographic sites. Such configuration sets up real working conditions where a permissioned blockchain is not ruled and run by a single entity. Used platform is Ethereum-Hyperledger BESU implemented through docker technology. We compared standard "Caliper" tests against more detailed and customizable tests executed by launching different transactions to stimulate the answer of the entire network under typical working conditions. Results show that, unexpectedly, not all blockchain nodes work equivalently: under certain conditions, only some nodes contribute to validate transactions and to include them into blocks, while others only append empty blocks to the ledger. This work has a twofold purpose. First, behind the analysis of the specific permissioned blockchain, the aim is to investigate and detect general issues and pitfalls related to this kind of blockchain. Second, providing and improving a tool that can be customized for performance analysis.

1 Introduction

Permissioned blockchain have possible employments in private or consortium organizations [1, 2, 3, 4]. A major difference with respect to public blockchain is that every node is controlled by the consortium giving the possibility to check and apply some level of control also on performances as well. In public blockchains every node can join and network load and performances are an emerging property. Delay in transaction approval are related to fees and can be unpredictable [5]. In a permissioned one bad performing nodes can be removed, their number can be tuned and transaction load can be set and ruled in order to optimize blockchain performance. This can depend not only on the internal configuration, but also from external fea-

L. Mostarda and D. Sestili
University of Camerino, IT, e-mail: leonardo.mostarda@unicam.it-davide.sestili@unicam.it

A. Pinna and R. Tonelli
University of Cagliari, IT, e-mail: pinna.andrea@unica.it-roberto.tonelli@unica.it

tures, mainly from network latency and nodes reachability. We analyze what are the main features influencing blockchain performances that can be taken care of, namely those variables that can be tuned and ruled by the blockchain consortium itself. Other research works tried to address this problem [6, 7, 8, 9, 10, 11] which is a general issue in Blockchain Software Engineering [12, 14].

We built a customizable tool and compared the result with *Caliper* [15], obtaining more details on performances. The performed tests can be useful for organizations or consortium aiming at implementing a permissioned blockchain by means of the docker container technology and the Hyperledger BESU blockchain under the IBFT2 consensus protocol. The is a first step forward for identifying the main features that is possible to work on for blockchain performance optimization, for identifying bottlenecks, critical issues and for configuring an overall testing and diagnostic framework for this kind of analysis. The approach is useful for consortium where blockchain nodes are maintained by independent authorities and single policies may differ from one organization to another or from one node to another.

2 The blockchain network.

The blockchain network is built up as a private permissioned network. Permissioning is implemented by the Consensys [16, 17] Dapp mechanism where new nodes can be onboarded if added into the permissioning smart contracts by administrators. Administrator can be onboarded and registered as well and can manage the addition of new nodes or new administrators with their cryptographic credentials.

The technology is Hyperledger BESU with IBFT2.0 consensus protocol where *validator nodes* in a Round Robin turn collect transactions into blocks and propose blocks to other validators. Empty blocks can be validated as well and added to the chain. Since the network is private and permissioned transactions are free and gas price has been set to zero.

Nodes are run by different and independent organizations which may adopt different access policies and firewalls and different machines and software. They are geographically distributed in different sites so that internet interaction can play a true role in the exchange of information among them. Every organization implements a node by means of the same docker-container technology but BESU versions run by the nodes can be different.

Network set-up required four bootnodes as validator nodes to start with the minting process. After setting up the four validators, two more standard nodes where added to the network. New validators can be added or existing validators can be removed as well through a voting process ruled by *curl* queries addressed to existing validator nodes. Smart contracts are written in Solidity 0.8 and only authorized addresses can deploy them while every address can query the network and send standard transactions. Some smart contracts have been deployed for testing purposes in order to analyze nodes performances against specific smart contracts calls.

3 Tools

In this work we aim not only at investigating our specific blockchain set-up, but also at getting hints and detect general issues and pitfalls related to permissioned/private blockchain and to the IBFT protocol more in general. Furthermore we aim at building and improving a tool that can be customized and fine tuned for this kind of analysis and can provide more details with respect to existing tools. To these purposes we used "Caliper" and compared the results with a custom built tool presented in the next sections.

3.1 Hyperledger Caliper

Hyperledger Caliper [15] is a blockchain performance benchmarking tool for Hyperledger Besu and other Ethereum-like blockchain technologies. Caliper allows to set up a workload specifying the nodes to which send requests (via web socket) and information about the transaction load. Transaction loads are completely customizable, allowing the user to define the fields of the transactions that will be sent, as well as the number of transaction in the load and the *send rate*: the number of transaction to submit per second.
The metrics measurable with Caliper are:

- *Read latency*: the time elapsed between a request is submitted and a reply is received;
- *Read througuput*: the total amount of read operations successfully submitted for which a reply has been received per second;
- *Transaction latency*: the amount of time a transaction requires to be part of the network from the moment it was submitted;
- *Transaction througuput*: the rate at which valid transactions are committed to the blockchain.

3.1.1 Custom Java benchmark tool

The custom java benchmark tool we developed allows us to send custom ethereum transactions to a node at a steady configurable rate and retrieve various metrics. The application allows us to configure various parameters to be used in a benchmark run, including:

- *Transaction send delay*: Number of milliseconds the application waits in order to send a new transaction after sending the previous one;
- *Transaction number*: number of transactions to be sent in a given run;
- *Timeout*: the number of milliseconds after which a transaction is considered to be not included in the blockchain if its transaction receipt hasn't been received;

- *Communication protocol*: The protocol employed for communicating with the node (either HTTP or Web socket);
- *Node address*: The node IP address and port number used to communicate with the blockchain node;
- *Sender key*: Ethereum account private key used for signing transactions.

The transactions that can be sent by the application are either simple value transfer transactions or smart contract method calls, more specifically, the smart contract method call that the application natively supports is the *transfer* method of the ERC20 smart contract. In case the application is run in the *smart contract method call mode*, the application will first commit a new ERC20 smart contract, wait for its inclusion in a block, and then will start sending the method call transactions.
After a benchmark run, the application outputs the *average transaction delay* – which is the average number of milliseconds elapsed from when a transaction has been sent to when it has been included in a valid block– and the number of blocks that have been generated during the run. Furthermore, for every transaction committed successfully, the application outputs information regarding the transaction and the block that contains the transaction. This information contains: block number; the timestamp when the transaction was sent to the node; the timestamp of the block; the ethereum address of the validator that proposed the block; the *transaction delay*.

4 Tests

The Smart Contract code we used for Caliper testing is a standard one and is reported in Listing 1.

Listing 1 Hyperledger Caliper benchmark smart contract deployed in the blockchain

```
pragma solidity >=0.4.22 <0.6.0;

contract simple {
    mapping(string => int) private accounts;

    function open(string memory acc_id, int amount) public {
        accounts[acc_id] = amount;
    }

    function query(string memory acc_id) public view returns (int amount) {
        amount = accounts[acc_id];
    }

    function transfer(string memory acc_from, string memory acc_to, int amount)
        public {
            accounts[acc_from] -= amount;
            accounts[acc_to] += amount;
        }
}
```

The code we used for Customized testing is a standard ERC20 Token and can be found at "OpenZeppelin"[18]. Caliper allows to call the three functions depicted in the SC. One call can create accounts (Token), one is only a query, the third one is the most interesting and sends a transaction. Since simple queries do not consume gas, do not change blockchain state and are not included into blocks and since account (Token) creation is by far simpler and less interesting (there is not a reverting process), we choose to test performances using transactions to transfer Tokens. It must be noted that our blockchain set-up sets gas cost to zero and the used addresses does not have Ethers (the balance is zero). This means that transactions sending Ethers are reverted but nevertheless they consume gas, so that must be included in blocks. With this in mind we tested our tool against calls to the "Transfer" method of the ERC20 SC aimed at transferring a Token from sender's address to another address. Each transaction has been inserted into blocks but they triggered the revert procedure of the method since the address called the transfer method having no Tokens. As a consequence Smart Contract's balances cannot be updated but gas is consumed anyway. Transactions were sent with different transaction request rates per second and with a different total number of requests spanning different configurations in order to test the blockchain answer under different workloads. The details of each configuration are reported in following section.

5 Results

Blokchain network performance metrics were first measured using the Hyperledger Caliper tool and then via the Custom Java benchmark tool. In particular, we will discuss the use of the custom Java benchmark tool as a diagnostic tool which provides more details useful to determine the causes of anomalies in network metrics.

5.1 Caliper results

Hyperledger Caliper shows measured metric values for three different blockchain interactions: *open*, *query*, and *transfer*. The *open* and *transfer* tests are performed with gas-consuming transactions to the smart contract. The *query* test is a call to the view function of the smart contract. Transactions were sent for a programmed time of 100 seconds at the different send rates of 0.2, 1, 2, and 10 transactions per second (TPS). An additional test at the rate of 100 transactions per second was set up but was not carried out by the tool. Caliper metric results are summarized in Tab. 1 where we report the results of the "transfer" test.

The results show how the average latency time of the transactions tends to increase with the increase of the send rate. This value should not depend on the number of transactions and should be close to half the block time (which in our case is 6 seconds). However, it seems that the throughput is close to optimal. Caliper there-

Table 1 Results of the execution of Hyperledger Caliper tests at four different transaction send rates. Only the results relating to the *transfer* tests are reported.

Tx Send Rate (TPS)	0.2	1	2	10
Maximum latency (s)	5.20	6.66s	12.39	13.42
Minimum latency (s)	0.23	0.64	0.69	0.97
Average latency (s)	2.70	3.58	6.41	7.23
Throughput (TPS)	0.2	0.9	1.9	9.5

fore allows us to detect an anomaly in latency metrics but does not provide further information to understand finer details.

5.2 Custom tool results

We configured the Custom tool to send transactions using WebSocket as a protocol, to use the closest node (in our case, the local host) as destination, and to use a single sender key. We set proper values for the parameters in order to have

$$transaction\ number/transaction\ send\ delay = 100$$

in other words, transactions were sent for a programmed time of 100 seconds. Different values for *transaction send delay* were set in order to send transactions at the different send rates of 0.2, 1, 2, 10, and 100 transactions per second (TPS). We performed the test twice (i.e., test 1 and test 2 in the following) to verify the repeatability of the observations. After running the tests, our custom tool produces the results in the form of serialized data on a file that can be easily processed. All of the data shown below are the results of processing the custom tool's data without the need for any additional queries to the blockchain.

Tab. 2 and Tab. 3 report the block metrics of the two tests performed with the same configuration of the tool. In these tables we can read for each send rate: the number of blocks created during the test; the total block time elapsed between the first and last block created; and the average block time.

Thanks to the data produced by the Custom Java Benchmark tool, we can understand how the blockchain network acquires transactions and how and when these are inserted into new blocks. For both tests, the average block time is stable for any TPS (6 seconds).

Table 2 The blocks metrics of the test 1 are reported in each column of the table for a different value of the transaction sending rate.

Tx Send Rate (TPS)	0.2	1	2	10	100
Number of Tx	20	100	200	1000	10000
N. of Blocks	16	17	19	19	24
Total Block Time (s)	90	96	108	108	139
Average block time (s)	6.0	6.0	6.0	6.0	6.043

Table 3 The blocks metrics of the test 2 are reported in each column of the table for a different value of the transaction sending rate.

Tx Send Rate (TPS)	0.2	1	2	10	100
Number of Tx	20	100	200	1000	10000
N. of Blocks	17	18	18	19	25
Total Block Time (s)	96	102	102	108	144
Average block time (s)	6.0	6.0	6.0	6.0	6.0

The tool allows the evaluation of latency metrics and to compare the results with the measurements made by Caliper. The transaction latency (i.e., the time interval between the instant of sending the transaction and the instant of confirmation) allows us to verify if the transactions are correctly inserted in the blocks. The latency is expected to be a local maximum if the transaction is requested immediately after creating a block. Conversely, minimal latency is expected if the transaction is sent close to the creation of a new block. Furthermore, an average latency of close to 3 seconds is expected for each test (or half of the block time established during the blockchain's genesis, which in our case is 6 seconds). As for the number of transactions per block, each block is expected to contain the same number of transactions, except for the first block and the last block. Any noticeable difference between expected and measured data is worth further investigation and can be traced back to network node operational anomalies.

The Tab. 4 and Tab. 5 show the latency measures of test 1 and 2 respectively. The results confirm the values of the Hyperledger Caliper metrics obtained for the transaction send rates up to 10 TPS.

In both tables, for rates up to 10 TPS, the minimum delay remains below one second. At a rate of 100 TPS the minimum delay increases considerably up to 4.7 and 2.8 seconds for test 1 and test 2 respectively. For the rates of 2, 10, and 100 TPS, a significantly higher than expected average latency is observed, and, for both test1 and test2, the average latency increases with higher send rates.

Table 4 Test 1. The values of the transaction confirmation delays are reported in each column of the table for a different value of the sending transaction frequency parameter.

Tx Send Rate (TPS)	0.2	1	2	10	100
Maximum latency (s)	5.99	6.289	12.235	13.001	22.112
Minimum latency (s)	0.897	0.190	0.402	0.337	4.720
Average latency (s)	3.405	3.206	5.246	6.759	12.638
Throughput (TPS)	0.208	0.997	1.779	8.927	69.136

Therefore, an anomaly is found for the transaction send rates of 2 transaction per second and above. The data produced by our Custom benchmark tool allows the examination, transaction by transaction, of the latency, the block number, and the block's miner. This allows for delving into the reasons for the increasing in the latency. To investigate the causes of this anomaly, we consider the smallest sending rate that causes anomalies and the highest send rate, i.e. 2 TPS and 100 TPS.

Table 5 Test 2. The values of the transaction confirmation delays are reported in each column of the table for a different value of the sending transaction frequency parameter.

Tx Send Rate (TPS)	0.2	1	2	10	100
Maximum latency (s)	6.917	6.664	12.196	12.774	18.528
Minimum latency (s)	0.833	0.567	0.289	0.377	2.835
Average latency (s)	3.781	3.596	5.092	6.580	10.712
Throughput (TPS)	0.199	0.960	1.855	8.879	66.454

5.2.1 Investigation at 2 TPS

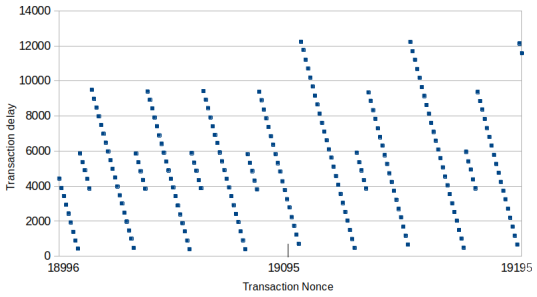

Fig. 1 Transaction delay (latency) measured in the test 1 at a transaction send rate of 2 TPS.

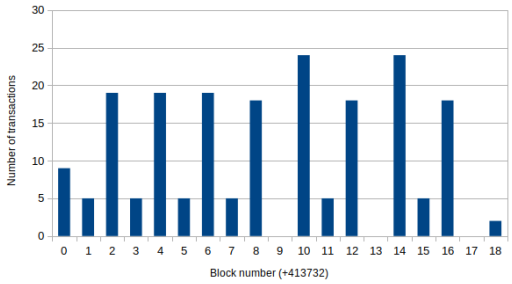

Fig. 2 Number of transactions per block in the test 1 at a transaction send rate of 2 TPS.

The first investigation is based on the data from both tests, at a transaction request rate of 2 TPS. We will show below the graphs relating to the transaction latency and to the number of transactions per block. Latencies (i.e. the confirmation delays) of transaction sent in test 1 at 2 TPS are shown in the graph of Fig. 1. In this figure, each point in the graph represents a transaction and its confirmation latency. Transactions are sorted by nonce. The y-axis represents the latency time in milliseconds. An expected pattern would show the points forming a transversal alignment, and each alignment would be formed by the same number of points (except the first and last). For each alignment, the upper-left point corresponds with the transactions with higher latency (sent just after the creation of a block, so that they need to wait the

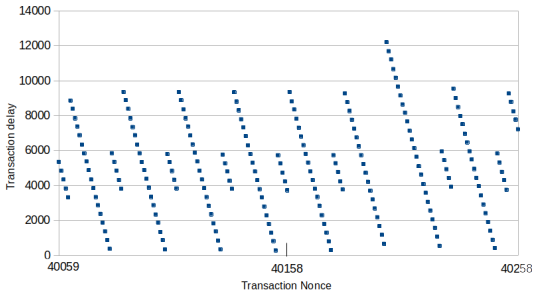

Fig. 3 Transaction delay (latency) measured in the test 2 at a transaction send rate of 2 TPS.

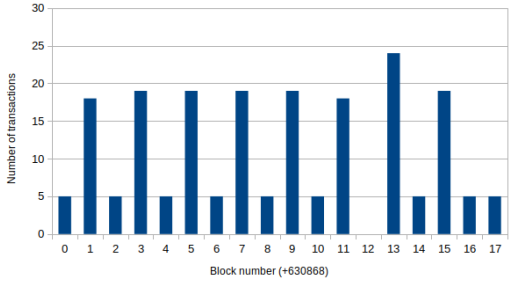

Fig. 4 Number of transactions per block measured in the test 2 at a transaction send rate of 2 TPS.

Table 6 Validators' addresses starting from the first block created during the test 1 at 2 TPS.

Block number	0	1	2	3	4	5	6	7
Validator	0x5a	0x7c	0xb3	0x2d	0x5a	0x7c	0xb3	0x2d

Table 7 Validators' addresses starting from the first block created during the test 2 at 2 TPS.

Block number	0	1	2	3	4	5	6	7
Validator	0x7c	0xb3	0x2d	0x5a	0x7c	0xb3	0x2d	0x5a

next block to be included), and the bottom-right point corresponds with the transactions with lower latency (sent close to the next block creation).

Alignments of different lengths are shown in Fig. 1 for test 1 at 2 TPS. Longer alignments are interspersed with shorter alignments, and two alignments appear missing. This is due to an uneven distribution of transactions in the blocks. Proof of this can be found in the graph in Fig. 2 where each bar represents the number of transactions per block. Blocks are ordered by their number (or height), starting at zero for the first block created during the execution of the test. The y-axis represents the number of transactions in each block. We can see that odd blocks contain only 5 or zero transactions sent during the experiment. In particular, two blocks (9 and 13) are empty.

Similar results are showed in Fig 3 and Fig. 4 for the test 2 at 2 TPS, at the same configuration. In this second test, even blocks contains only five or zero transactions sent during the test 2 at 2 TPS.

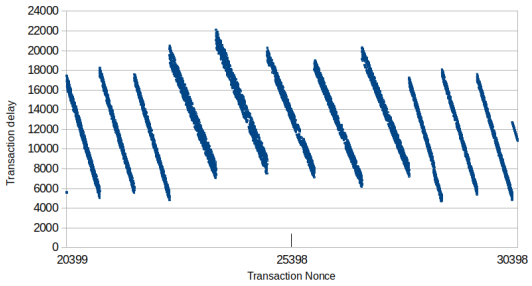

Fig. 5 Transaction delay (latency) measured after the test 1 at a transaction send rate of 100 TPS.

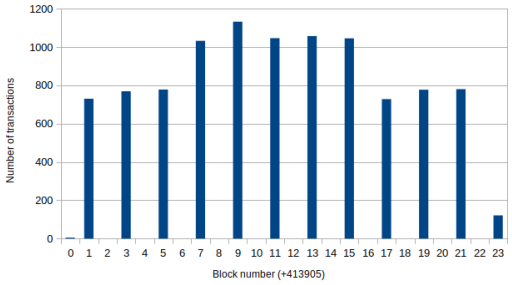

Fig. 6 Number of transactions per block measured after the test 1 at a transaction send rate of 100 TPS.

The test results confirmed that the validators are selected for the creation of the next block via a simple rotation. Tab. 6 provides the order in which the validators create new blocks, as determined in test 1 at 2 TPS, and tab. 7 as determined in test 2. For the sake of simplicity, only the first byte of each validator's address is reported. Given the regularity of the validators' sequence, the nodes that mine blocks with few or zero transactions are always the same two: 0x7c and 0x2d.

5.2.2 Investigation at 100 TPS

Similar examinations were made on the results of test 1 and test 2 at the rate of 100 transactions per second. A more peculiar behavior can be observed at this rate. As regards the latency, in Fig. 5 and Fig. 7 it can be seen that the distribution of the points in the graph does not have the same slope in each group. Also, narrow groupings are no longer present.

At this transaction send rate, there is a significant increase in the minimum, maximum, and average latency, with respect to the results of lower send rate. The reason of the increasing of the latency can be seen in the graph in Fig. 6 for the test 1 at 100 TPS, where every odd block with hundreds of transactions is followed by an even block with zero transactions. Similar results are reported in the Fig. 8 for the test 2 but this time the blocks without transactions are the even blocks. This means that

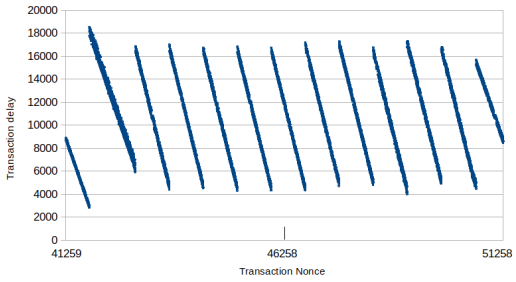

Fig. 7 Transaction delay (latency) measured after the test 2 at a transaction send rate of 100 TPS.

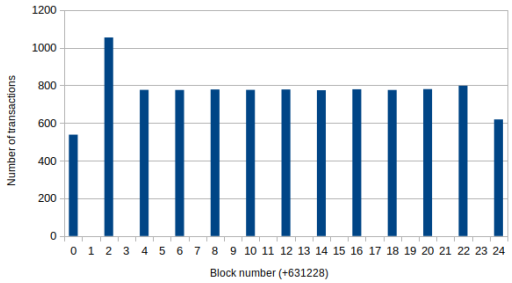

Fig. 8 Number of transactions per block measured after the test 2 at a transaction send rate of 100 TPS.

transactions could wait two blocks or more until they are included in a block. At this rate, the same circular order of the validators already determined in the 2 TPS rate is confirmed. We can assert that the validator "0x7c" and the validator "0x2d" are working differently than expected. These validators systematically enter a limited number of transactions per block, and do not confirm transactions in blocks at rates of 10 and 100 TPS.

6 Conclusions

In this work we presented a tool for the analysis of blockchain performances under various and customizable loading conditions. Tool's results has been briefly compared with those of "Hyperledger Caliper", used as a benckmark tool for analyzing blockchain performances. While further analysis is quite limited with Caliper, our tool is able to provide more details on the role of each blockchain node, of every block and on every transaction that can be traced by the sending time and the nonce. This allowed us to detect validator behaviours that resulted in anomalies when adding, or missing to add, transactions to the due blocks. The tool can be further improved and is virtually able to test any kind of possible behaviour or misbehaviour of validator nodes.

References

1. Baliga, A., Subhod, I., Kamat, P., and Chatterjee, S. (2018). Performance evaluation of the quorum blockchain platform. arXiv preprint arXiv:1809.03421.
2. Shapiro, G., Natoli, C., and Gramoli, V. (2020, November). The performance of Byzantine fault tolerant blockchains. In 2020 IEEE 19th International Symposium on Network Computing and Applications (NCA) (pp. 1-8). IEEE.
3. Helliar, C. V., Crawford, L., Rocca, L., Teodori, C., & Veneziani, M. (2020). Permissionless and permissioned blockchain diffusion. International Journal of Information Management, 54, 102136.
4. Vukolić, M. (2017, April). Rethinking permissioned blockchains. In Proceedings of the ACM workshop on blockchain, cryptocurrencies and contracts (pp. 3-7).
5. Pierro, Giuseppe Antonio, et al. "A user-oriented model for Oracles' Gas price prediction." Future Generation Computer Systems 128 (2022): 142-157.
6. Dabbagh, M., Choo, K. K. R., Beheshti, A., Tahir, M., & Safa, N. S. (2021). A survey of empirical performance evaluation of permissioned blockchain platforms: Challenges and opportunities. computers & security, 100, 102078.
7. Leal, F., Chis, A. E., & González–Vélez, H. (2020). Performance evaluation of private ethereum networks. SN Computer Science, 1(5), 1-17.
8. Choi, Wonseok, and James Won-Ki Hong. "Performance Evaluation of Ethereum Private and Testnet Networks Using Hyperledger Caliper." 2021 22nd Asia-Pacific Network Operations and Management Symposium (APNOMS). IEEE, 2021.
9. Fan, C., Lin, C., Khazaei, H., & Musilek, P. (2022, August). Performance Analysis of Hyperledger Besu in Private Blockchain. In 2022 IEEE International Conference on Decentralized Applications and Infrastructures (DAPPS) (pp. 64-73). IEEE.
10. Abdella, J., Tari, Z., Anwar, A., Mahmood, A., & Han, F. (2021). An architecture and performance evaluation of blockchain-based peer-to-peer energy trading. IEEE Transactions on Smart Grid, 12(4), 3364-3378.
11. Mazzoni, M., Corradi, A., & Di Nicola, V. (2022). Performance evaluation of permissioned blockchains for financial applications: The ConsenSys Quorum case study. Blockchain: Research and applications, 3(1), 100026.
12. Porru, S., Pinna, A., Marchesi, M., & Tonelli, R. (2017, May). Blockchain-oriented software engineering: challenges and new directions. In 2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C) (pp. 169-171). IEEE.
13. Bistarelli, S.,Mazzante, G., Micheletti, M., Mostarda, L., Sestili, D., Tiezzi, F., (2020). Ethereum smart contracts: Analysis and statistics of their source code and opcodes. Internet of Things Elsevier Journal.
14. Marchesi, L., Marchesi, M., & Tonelli, R. (2020). ABCDE—Agile block chain DApp engineering. Blockchain: Research and Applications, 1(1-2), 100002.
15. https://hyperledger.github.io/caliper/
16. https://consensys.net/
17. https://github.com/ConsenSys/permissioning-smart-contracts
18. https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/token/ERC20/ERC20.sol