Hammer: A General Blockchain Evaluation Framework

Gang Wang[†], Yanfeng Zhang[*†], Chenhao Ying[‡], Xiaohua Li[†], Ge Yu[†]
[†]School of Computer Science and Engineering, Northeastern University, Shenyang, China
[‡]Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai, China
{1910636@stu, zhangyf@mail, lixiaohua@cse, yuge@mail}.neu.edu.cn, yingchenhao@sjtu.edu.cn

2024 IEEE 44th International Conference on Distributed Computing Systems (ICDCS) | 979-8-3503-8605-9/24/$31.00 ©2024 IEEE | DOI: 10.1109/ICDCS60910.2024.00044

***Abstract*—With the rising proliferation of blockchain systems and applications, choosing the appropriate blockchains to deploy applications is critical to achieving optimal performance. Evaluation frameworks provide a systematic approach to assessing and comparing different blockchain systems, guiding application developers to choose the most suitable one. However, existing evaluation frameworks still have limitations that affect their accuracy. First, most frameworks utilize workloads initially designed for traditional databases, which fail to capture the unique characteristics and requirements of blockchain systems. Second, these frameworks fail to generate correct results under heavy workloads due to their imbalanced task processing algorithms. Third, existing frameworks are tailored only for non-sharding blockchain architectures, limiting their ability to evaluate diverse blockchains.**

This paper introduces Hammer, a general blockchain evaluation framework that addresses the above limitations. It consists of two key components: workload prediction and asynchronous task processing. Workload prediction accurately predicts real-world workload trends by expanding the scope of temporal control sequences, providing a more realistic evaluation of blockchain performance. Asynchronous task processing handles heavy-load situations, enabling accurate evaluation of blockchain performance. Extensive experiments on various blockchains under Smallbank workload empower application developers to make informed decisions about blockchain selection and optimization.

***Index Terms*—Evaluation framework, asynchronous task processing, Hammer**

I. Introduction

With the dramatic development of blockchain technology, more and more new blockchain architectures [1], [2] have been proposed. The diversity in architectures has prompted evaluation and comparison of the performance and functionality across different types of blockchains [3]. Evaluation plays a crucial role in building blockchain frameworks since it provides valuable insights from various perspectives of blockchain, such as security [4], scalability [5], and availability [6]. Moreover, just like other evaluation tools, precision, speed, and scalability are three crucial requirements in the design of blockchain evaluation [7]. However, the existing evaluation cannot own these desirable capabilities together.

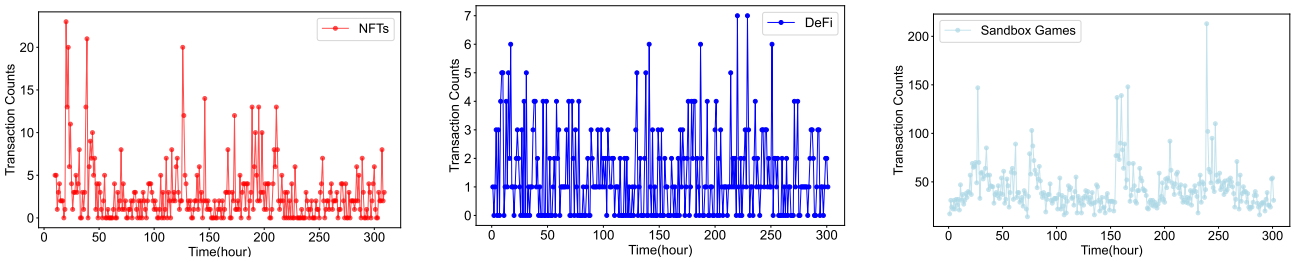
First, the workload in real-world blockchain applications varies at different times. In blockchain, time series are used to represent the number of concurrent transactions within a specific time interval, which often exhibit bursty and periodic features. For example, we select the recent 300 hours transaction information of Non-Fungible Tokens (NFTs) [8], Decentralized Finance (DeFi) [9], and Gaming [10] from the last 300 hours, and show the temporal distribution result in Fig.1. It reveals that the actual workloads exhibit rapid variations and bursts across different durations. Furthermore, the temporal features of different applications are different. For instance, compared to the distributions of Sandbox Games, DeFi and NFTs are more stable. However, the current blockchain evaluation frameworks fail to capture the temporal characteristics when generating workloads. In contrast, they simply generate an equal number of workloads during the evaluation of the blockchain's performance. Therefore, it is challenging for the evaluation frameworks to accurately show the actual performance of blockchain across different time periods, including processing speed, response time, and resource utilization.

Second, blockchain performance is usually evaluated by employing batch testing or interactive testing. In fact, batch testing periodically listens, intercepts, and retrieves new blocks, which are parsed to extract transaction IDs for matching transactions in the local queue. The corresponding ID will be removed from the queue once the match is correct. The testing requires a time complexity of $O(mn)$, where n is the length of the queue and m is the number of transaction IDs detected within the new blocks. Therefore, it suffers a serious delay with the increase in the number of transactions. In contrast to batch testing, interactive testing keeps listening and processes each new block once it appears. Although the blocks can be processed in time, it consumes a lot of resources for continuous listening.

Third, the existing blockchains are built based on sharded or non-sharded architectures. In fact, compared with the non-sharded blockchains, such as Fabric [11], Ethereum [12], and Neuchain [1], the sharded blockchains like Meepo [2] are attracting more and more attention since they offer higher throughput and less confirmation time. However, the existing evaluation cannot adapt to the non-sharded blockchain. Moreover, to leverage the strengths of different programming languages, existing blockchains are built employing various languages. For instance, Ethereum and Fabric employ Go, while Neuchain and Meepo are developed with C++ and Rust, respectively. Nevertheless, the previous evaluations can only accommodate a subset of them. For example, Blockbench only supports Rust and Go, while apart from Go, Caliper also

*Corresponding Author: Yanfeng Zhang

2575-8411/24/$31.00 ©2024 IEEE
DOI 10.1109/ICDCS60910.2024.00044



(a) NFTs (b) DeFi (c) Sandbox Games

Fig. 1: Temporal distribution of real workload

supports Java and C++. The limited scalability in adapted architecture and supported language restricts the applicability of different evaluation frameworks.

To overcome the limitations of existing works, we design a novel evaluation framework, namely, Hammer, comprising two key components. Workload temporal prediction that is employed to enhance evaluation accuracy, and asynchronous task processing utilized to reduce processing time in batch testing. In fact, the workload temporal prediction is a learning-based algorithm meticulously trained with actual workload data extracted from NFTs, DeFi, and Sandbox Games so that it can grasp the temporal intricacies of blockchain workloads and predict their future trends. Furthermore, to adeptly handle scenarios with substantial workloads, transaction processing optimizes the matching process by establishing a hash index, which can effectively reduce the time complexity to $O(1)$. In fact, it polls and parses the transaction IDs in the last block, after which, a bloom filter is utilized to sift out non-existent transactions. For potentially existing transactions, hash indexing is deployed to determine their exact locations. In handling hash collisions, it further expands the length of the hash table. Finally, to ensure broad applicability and scalability, Hammer also offers a set of generic remote procedure call (RPC) interfaces to accommodate different blockchain architectures.

In summary, our contributions are as follows.

1) We develop a general blockchain evaluation framework, namely, Hammer. It contains two key components. The first one is workload prediction aiming to effectively predict real workload trends by expanding the scope of temporal control sequences. The second is the asynchronous transaction processing that is applied to handle heavy-load situations. Offering a set of generic remote procedure call (RPC) interfaces, Hammer is able to adapt to both sharded and non-sharded blockchains constructed by various programming languages, such as C++, Rust, Go, and Java [1]. To the best of our knowledge, this is the first generic blockchain evaluation framework.
2) The workload prediction lays a solid foundation for integrating time sequences into standard benchmark testing. In fact, it is a learning-based algorithm meticulously trained with actual workload data extracted from NFTs, DeFi, and Sandbox Games. Compared with the traditional transformer and revolution neural networks (RNN), the experiments on DeFi, NFTs and Sandbox Games show that the mean absolute error (MAE) of workload prediction is decreased by more than 56%.
3) The asynchronous task processing enables accurate and efficient evaluation of blockchain performance and responsiveness. In fact, it optimizes the matching process by establishing a hash index, which can effectively reduce the time complexity to $O(1)$. Compared with the traditional batch testing, the experiment results show that the execution time of asynchronous task processing is decreased by more than 50% when the number of transaction is $100,000$.

The rest of the paper is organized as follows: Section II provides the necessary background. Section III presents the system design. Section IV presents a learning-based time series model. Section V shows the evaluation results and analysis of them. Section VI reviews related work, and we conclude in Section VII.

II. Background

In this section, to promote understanding of our evaluation frameworks, we first briefly introduce several types of blockchain architectures. Then, we highlight the workload limitations of existing evaluation frameworks and then outline the interactive methods of blockchain evaluation.

A. Blockchain Architectures

1) Non-Sharded Architecture: Bitcoin [13] introduces the concept of blockchain and categorizes blockchains into permissioned and permissionless based on the permission mechanism of blockchain nodes. In permissionless blockchains, nodes can join or leave the network at any time, while in permissioned chains, nodes must be authorized when joining the network. Furthermore, according to the scope of node partitioning, blockchain can be further classified into non-sharded and sharded architectures. In the Non-Sharded, every node in the blockchain is required to maintain and process all the data of the entire network. Therefore, it imposes higher

[1]The system is also compatible with all mainstream programming languages, including Solidity and JavaScript.

requirements on data processing and storage capabilities. Currently, the most widely-recognized blockchain systems, including Bitcoin [13], Ethereum [12], Fabric [14], and Neuchain [1] are based on the non-sharded architecture.

2) Sharded Architecture: The core concept of sharding involves a 'divide and conquer' strategy, where all nodes are segmented into distinct groups, each referred to as a shard. Subsequently, tasks are allocated across these shards for simultaneous processing, aiming to enhance the overall performance of the blockchain. The sharded blockchain works as follows. i) Divide the blockchain network into several shards, where each shard contains a subset of nodes and the corresponding ledger. ii) Perform actions in each shard, including transaction validation, status updates, and so on. iii) To adapt to various workloads and network scales, the network dynamically forms new shards to optimize performance and meet requirements. In fact, the sharded architecture is an optimization method to accelerate the execution and verification processes. However, the sharded architecture brings new challenges to the evaluation framework, including the impact of unbalanced loads within and between partitions on the overall performance of the blockchain.

B. Benchmark Workloads

BlockBench [4] is the pioneering blockchain benchmarking framework that focuses on evaluating micro/macro metrics for permissioned blockchains. BlockBench evaluates blockchains based on database benchmarks, which in fact cannot adapt to the complicated blockchain applications and workloads. Similarly, [11] applies the classic TPC-C benchmark to Hyperledger Fabric, and presents a structured approach for transforming the original database scheme into a smart contract based data model. The Diablo Benchmark Suite [6] offers various workloads, such as FIFA and YouTube. However, these workloads do not have the characteristics of blockchain applications. Blockbench V3 [7] provides four kinds of real-world blockchain workloads, such as token exchange, and NFT minting. Although Blockbench V3 captures the characteristics of some real blockchain workloads, it fails to follow the temporal features of workloads. In fact, apart from the real workloads, the evaluations such as YCSB [15] and SmallBank [16] also utilizes the synthetic workloads. The real workloads are extracted from the real applications while lacking controllability and flexibility. In contrast, the synthetic workloads are more flexible and controllable but can not reflect the actual application characteristics.

Therefore, we apply self-defined workloads, which are generated by a learning-based algorithm trained by the real application data. Compared with the real and synthetic data, they can reflect the characteristics of real workloads while maintaining controllability and flexibility, making the evaluation of blockchain systems more comprehensive and adaptable.

C. Blockchain Benchmarking Tools

Performance benchmark frameworks such as TPC-C [17], YCSB [15] and SmallBank [16] are well-established and have essentially formed the industrial standards. However, these frameworks cannot be directly applied to benchmark distributed ledger systems due to their specific architecture and APIs [3]. With the proliferation of an increasing number of blockchain systems, it is crucial to design a standardized and generic evaluation framework to ensure a fair and accurate performance evaluation. There are several popular performance benchmarks dedicated to evaluating blockchain systems, as listed in Table I. We will focus on thoroughly understanding the interaction-based blockchain evaluation frameworks.

1) Batch Testing: Blockbench and its extended version Blockbench V3 utilizes the batch testing method. It works as follows. i) When the client submits a transaction to the blockchain system, it returns a transaction ID to check transaction status later. ii) The driver maintains an unconfirmed and incomplete transaction queue. New transaction IDs are added to the queue by worker threads. The polling thread periodically calls the API to get the confirmed blocks as well as the height H. The driver then extracts the transaction list from the contents of the acknowledgment block and removes the matching transaction list from the local queue. iii) Calculate the throughput of successfully committed transactions.

However, the batch testing method brings two challenges. $\xi 1$) Choosing an appropriate polling thread time interval is crucial for both system performance and accuracy. If the interval is too large, the calculated transaction latency will be skewed since the method relies on the time to poll for a new block as the transaction's completion time. A large time interval leads to missing block generation time and thus results in overestimating transaction latency. Conversely, if the interval is too small, the polling thread over-polls and wastes computing resources. $\xi 2$) When facing a large volume of test transactions and a significant queue length, it consumes more time for the extraction of transaction lists from confirmed blocks and the subsequent deletion of matching transaction lists in the local queue. This leads to an increase in the operation time, while cannot accurately reflect the instantaneous performance of the system.

To formally prove the increase in time complexity when facing a large volume of test transactions and a significant queue length, we can define the time complexity of the operation as a function of the input size. Let's denote: n as the size of the queue, and m is the number of transactions in the confirmation block, and $T(n, m)$ as the time complexity of the operation. Now, let's assume that the time of extracting transaction lists from confirmed blocks and deleting matching transaction lists in the local queue are proportional to n and m, and is denoted as

$$f(n, m) = k \cdot n \cdot m \tag{1}$$

where k is a constant. The time complexity of the overall operation can then be defined as the sum of this function over all possible inputs:

$$T(n, m) = \sum_{i=1}^{n} \sum_{j=1}^{m} f(i, j) \tag{2}$$

TABLE I: Summary of blockchain benchmarking tools

Frameworks	Supported Type	Supported Language	Supported Architectures	Workloads	Testing Methods
Blockbench [4]	Permissioned	Rust, Go	Non-Sharding	Synthetic workload	Batch
Blockbench v3 [7]	Permissioned	Rust, Go	Non-Sharding	Real workload	Batch
Caliper [18]	Permissioned	Java, C++, Go	Non-Sharding	Self-defined	Interactive
Bctmark [19]	Permissioned	Go	Non-Sharding	Synthetic workload	Interactive
Diablo-v2 [6]	Permissioned	Move, Go	Non-Sharding	Real workload	Interactive
HyperledgerLab [20]	Permissioned	Go	Non-Sharding	Real workload	Interactive
Gromit [21]	Permissioned	Go, C++, Rust, Move	Non-Sharding	Synthetic workload	Interactive
BlockCompass [5]	Permissioned	Go, Python	Non-Sharding	Self-defined	Interactive
DLPS [22]	Permissioned	Go, Python, Rust	Non-Sharding	Synthetic workload	Interactive
Hammer (ours)	**Permissioned and Permissionless**	**Go, C++, Rust, Java and Python**	**Non-Sharding and Sharding**	**Self-defined**	**Batch+Task processing algorithm**

which increases with the growth of n and m.

2) Interactive Testing: This method is commonly utilized by most blockchain benchmarking frameworks. It works as follows: i) The client sends transactions to the blockchain system, and waits for processing by blockchain nodes. ii) The evaluation driver thread is responsible for receiving responses from the blockchain network, parsing the responses, determining the success or failure of each transaction, and recording the results for subsequent performance analysis. iii) Based on the recorded data, the evaluation framework calculates overall throughput and transaction latency. Obtaining real-time responses for each transaction can provide real-time performance metrics for the system, this method requires monitoring and parsing responses for each transaction, potentially resulting in significant resource wastage.

III. Framework Design

In this section, we present a blockchain evaluation framework that is compatible with both permissionless and permissioned blockchains. We first provide an overview of the system design and introduce the key components in Section III-A. Then, we describe the execution flow details in Section III-B. The task processing algorithm is presented in Section III-C, after which, we provide two key techniques in Section III-D.

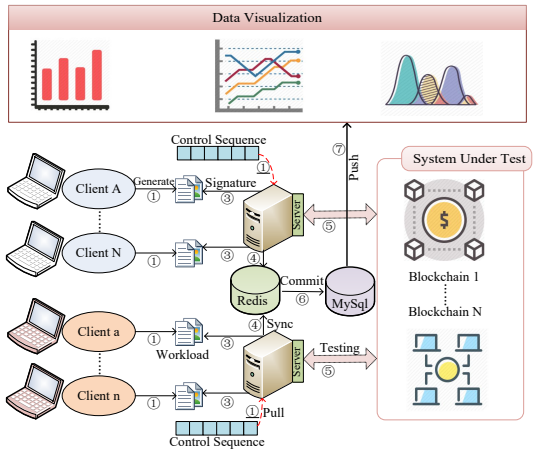


Fig. 2: Overall system architecture.

A. Overview and Key Components

This section provides an overview of our evaluation framework, Hammer. The overall architecture is shown in Fig. 2. The client parses the workload configuration files and generates executable workloads according to her offline workload generation strategy. All workloads are stored in the configuration files, and the server processes and signs them asynchronously through a pipeline. We replaced the queue with a vector list for storing transaction IDs, due to the high overhead associated with enqueue and dequeue operations in queues. The local vector list is updated only upon obtaining a new block. To efficiently handle the state of vector lists, we utilized Redis [23]. As a memory-based key-value storage system, it offers higher data processing speed and efficiency compared to MySQL [24]. Redis periodically polls, fetches, and merges the statuses of all vector lists. Subsequently, the records of Redis are committed to MySQL to meet the customization of the visualization layer for different analysis requirements. Some key components are described as follows.

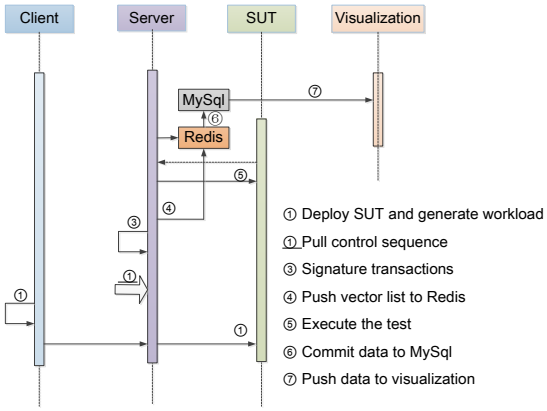


Fig. 3: Execution flow of hammer.

1) Client: In our framework, the client has two functionalities. i) The client is required to deploy the testing environment. In existing blockchain systems, deployment and configuration are complex and often suffer from building failures due to mismatched dependencies. To address this issue, we utilize the Ansible [25] component to develop automated deployment scripts, simplifying the deployment and configuration

processes of the blockchain environment. ii) After building System Under Test (SUT), the client is required to generate workloads as follows. First, the workload profile is parsed to obtain information such as workload read/write ratio, distribution, and so on. Secondly, the payload is generated based on custom application actions.

2) Server: The server is a key component that serves as the bridge between the client and SUT. Since SUTs have two kinds of architecture, the sharded and the non-sharded, the driver requires to address the following two issues. i) There is no unified communication mechanism. ii) Differences in the design language and architecture. To tackle this problem, we design a generic interface, which integrates SDKs of various blockchain platforms and introduces JSON-RPC. This innovation not only enhances the interoperability of the system but also addresses compatibility issues among different codes by applying standard formats. The server has two additional crucial functionalities: new block monitoring and transaction status updating. Since our system is based on the batch testing method, the monitoring is indispensable. Once the server detects a new block, it records the corresponding block time. Subsequently, the server extracts the list of transactions from the content of the confirmed block and updates the status of these transactions in the local vector list.

3) Visualization: The data visualization layer adopts an OLAP-like schema to enable users to perform in-depth analysis of blockchain performance based on temporal states. This layer is developed according to Grafana [26], which supports multiple table types, such as histograms and line charts. The SQL engine is employed to provide complex queries, pull data from MySQL, and display it. This comprehensive design aims to provide a powerful and intuitive set of tools to meet their diverse needs for system performance and data state analysis.

B. Exection Flow

Fig. 3 shows the execution flow, which consists of three phases: preparation, execution, and visualization. In the preparation phase, we deploy the SUT and generate workloads that match real-world applications to ensure the test environment is ready. The execution phase involves executing the workload on the blockchain and monitoring its completion, including collecting performance metrics such as throughput and latency. In the visualization phase, we process the collected data for visualization and analyze the SUT's performance.

1) Preparation Phase: Before starting a test, it is necessary to make adequate preparations. First, we deploy the SUT and provide automated deployment scripts to replace the manual deployment process. Currently, automated deployment scripts are available for four typical blockchain systems. Clients can automatically deploy SUTs using Ansible's playbook feature. After deploying SUT, the client reads the local JSON configuration file to parse information about the SUT and workload. As shown in step ① in Fig. 3, the client executes the corresponding commands to generate workload, which are persisted to a file and sent to the server via secure copy (SCP). Second, the server (step ①) pulls the control sequence, which is a time sequence to control the number of concurrent transactions within a time period. It simulates the timing features of real-world blockchain applications. The details of the control sequence will be further presented in Section IV. Note that steps ① and ① can be executed simultaneously. Finally, the workload file is read and signed by the server (step ③). We optimize the signature process by applying an asynchronous signature method, which will be further presented in Section III-D1.

TABLE II: Description of throughput and latency

Metrics	SQL Statement
TPS	SELECT COUNT(*) AS TPS FROM Performance WHERE STATUS = '1' AND TIMESTAMPDIFF(SECOND, start_time, end_time) $\leq$ 1
Latnecy	SELECT tx_id, start_time,end_time, TIMESTAMPDIFF(MILLISECOND, start_time,end_time) AS Latency FROM Performance

2) Execution Phase: In the execution phase, we propose pipelined preparation and execution strategies to eliminate the bottleneck of long transaction waiting times. The pipelining preparation and execution method will be presented in III-D2. In step ④ the server pushes the initialized vector list to the Redis cluster. Then, during the testing process, the driver will regularly update the vector list on the Redis cluster. In step ⑤ we perform tests by sending transactions to SUT and polling to retrieve the latest block. Next, we parse the block and update the transaction status in the vector list. If the transaction is successfully committed, the status of the corresponding transaction is set to 1. To optimize the performance of vector lists and improve the overall response time of the system, we propose an algorithm for dynamic indexing and fast query and update. The details can be found in Section III-C. In step ⑥, the Redis cluster periodically transfers data to the MySQL database to facilitate data processing during the visualization phase.

3) Visualization Phase: In the visualization phase, we employ the Prometheus middleware. Prometheus (a state monitoring system) [27] is used as a benchmark suite for collecting information. It pulls the internal metrics of each node during or after our evaluation, including CPU, memory, and network input and output consumption. In our system, we use the node-exporter (a Prometheus plugin) to monitor Docker containers or Ethereum Virtual Machine (EVM). The collected data will be inserted into a MySQL table named Performance. To create a chart in Grafana, we can choose the appropriate chart type and metrics, write SQL query logic, and set the refresh rate for data retrieval from the database. Additionally, data can be periodically pushed to Grafana for visualization. In step ⑦, data is periodically pushed to Grafana for visualization. For example, if we want to measure the throughput and latency of the blockchain, we need to write SQL statements to determine the calculation logic of transactions per second (TPS) and latency. Table II shows the calculations for throughput and latency. We retrieve the transaction start time and end time from the table using a SQL statement, and the latency is

calculated as the difference between the two values. Therefore, if the transaction latency is less than one second, the counter will be incremented.

C. Task Processing Algorithm

The workload is sent from the client to the local server and is processed in parallel on the local server. We refer to the process from sending the workload to its complete processing as a task. Algorithm 1 describes the task processing logic. First, it initializes the start and end times for each transaction. For each transaction in set T, the algorithm extracts the client id (c_id) and server id (s_id) (Lines 1-3). In transaction processing, c_id and s_id play two roles. One is for security (e.g., preventing flooding attacks), and the other is for monitoring the load on each client and server. When sending a transaction, the server obtains the start time and transaction id (t_id, Line 4). Then, a transaction structure is created, including the start time, end time, and transaction status. The structure is inserted into the vector list and indexed with the transaction id (Lines 5-7). To ensure concurrent transaction execution without sacrificing system performance, we propose the method of dynamic index creation. Transaction ids are unordered, so it is impractical to use data structures like B-tree or B+-tree. Our approach utilizes a hash table and leverages its efficient updating capabilities. However, this approach also presents new challenges. For example, how to address the hash collisions? In our strategy, we attempt to minimize the occurrence of hash collisions by expanding the length of the hash table to improve the efficiency of the index (Lines 8-9). When a new block is generated, we first record the time of block creation, which is considered as the time when transactions are successfully committed (a.k.a, end time). Subsequently, we initiate the block fetching operation. This operational approach mitigates the impact of network bandwidth on latency. Finally, the block is parsed to extract transaction information (Lines 10-13). In the querying process, we initially utilize a Bloom filter for rapid exclusion of transactions not in the index. Such process can significantly save time and bring some other benefits in distributed testing. If the query result is present in the Bloom filter, we proceed to locate it using a hash table. When the hash table query yields no conflicts, we directly modify the transaction's status to 1 and append the corresponding end time. However, in case of a collision in the hash table, we meticulously search each element in the bucket sequentially (Lines 14-19). This approach ensures logical consistency when handling conflicts.

D. Optimization Techniques

In Hammer, we present two key techniques as follows.

1) Asynchronous Signature Method: Unlike database workload, each blockchain workload contains some client signatures. The preparation phase in the existing system generates and loads the workloads into a message queue, where the system requires to wait for the completion of loading process before moving to the execution phase. After extensive

Algorithm 1 Task Processing Algorithm

Input: A set of transactions T
Output: Return success or false

1: Initialize the start time (S_t) and end time (E_t) of the transaction;
2: **for all** t **in** T **do**
3: Extract the transaction generation client id (c_id) and the sending server id (s_id);
4: $S_t, t\_id \leftarrow$ Send transaction t ;
5: $transaction\_info \leftarrow$ Create a structure that includes S_t, c_id, s_id, t_id, E_t, $chainname$, and $contractname$;
6: Store the $transaction\_info$ in a $vector\_list$.
7: $index \leftarrow$ Dynamically create the index for the t_id
8: Update the index table with the new $index$
9: **end for**
10: **while** New blocks are being generated **do**
11: $E_t \leftarrow$ Set E_t to the current time
12: $block \leftarrow$ Get the newly generated block;
13: $transactions \leftarrow$ Extract all transactions from the $block$;
14: **for all** $transaction$ **in** $transactions$ **do**
15: **if** t_id does not satisfy Bloom filter criteria **then**
16: **return** false
17: **end if**
18: $index \leftarrow$ Find the index in the index table by t_id;
19: Update the status and E_t of the transaction in the index table;
20: **end for**
21: **end while**
22: **return** success

research, the workload generating process is serialized. As illustrated in the example, the Caliper implementation is depicted in Fig. 4a. The signature of a transaction does not

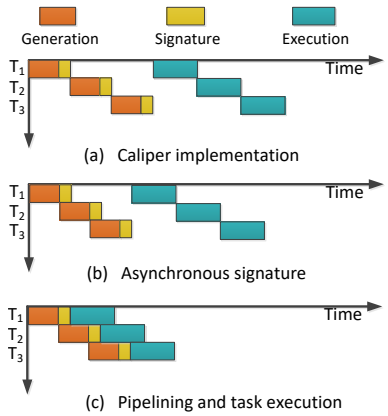


Fig. 4: Asynchronous signature and pipelining.

depend on any previous result. Therefore, we optimize it with an asynchronous signatures method. This process is illustrated

in Fig. 4. It reduces the time of signing process but requires to wait for all transactions to be ready for execution.

2) Pipelining Preparation and Execution: As mentioned in the previous subsection, tasks require to wait until the preparation phase is completed before execution. For the first arriving task, it suffers a long waiting time, which seriously affects the efficiency of the evaluation. Therefore, we optimize it with pipelining preparation and execution. Specifically, as shown in Fig. 4c, tasks T_1, T_2, and T_3 can be executed simultaneously, which decreases wait time and improves response time to better meet user needs. In other words, the preparation phase and the execution phase can be overlapped.

IV. Learning-Based Solution

In the previous section, we have emphasized the importance of control sequences. However, the length of the control sequence for real workloads is limited, which cannot meet the requirements of large-scale testing. In this section, we propose a learning-based solution. It requires to address the following questions. 1) how to capture long-term dependencies? 2) how to capture short-term dependencies? 3) how to capture sudden burst?

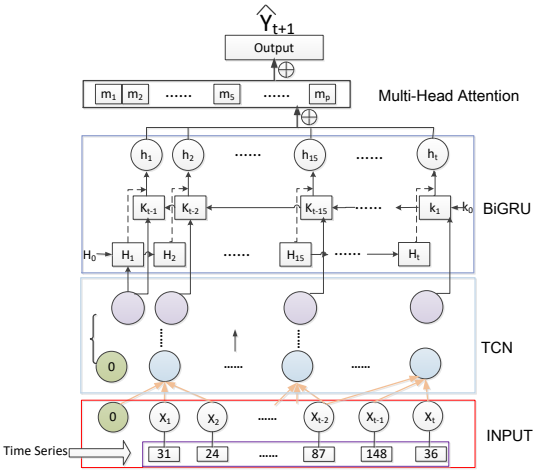


Fig. 5: Model Architecture

We propose a learning-based time control sequence model. It consists of two key models: i) The TCN [28] model, which is able to effectively capture long-distance dependencies. ii) The BiGRU [29] model, which focuses on capturing short-distance dependencies. Fig. 5 illustrates the overall architecture of time series model. The working procedure of model is as follows. First, we determine a time threshold in the real workload, count the concurrent transactions within the time threshold, and eventually generate a control sequence that increases over time. By leveraging the time series control sequence as training data, the TCN model is able to learn the long-term dependency information within the control sequence (e.g. periodicity). This permits the model to better understand and capture long-term dependencies in the sequence data. Second, the TCN model's output is fed into the BiGRU model, which comprehensively captures the features in the sequence data by simultaneously capturing both previous and forward information. Finally, the multi-head attention mechanism captures sudden bursts in the temporal control sequence. We next present the model design.

A. Algorithm Illustration

Our target is to predict time control sequences, which can be formally defined as: $x = \{x_1, x_2, x_3, \ldots, x_t\}$, where x is a set of control sequences for $x_i \in R$, $i \in (1, 2, 3, \ldots, t)$. t with the size t of control sequence. We predict x_{h+t} based on $x = \{x_1, x_2, x_3, \ldots, x_t\}$, where h is the prediction horizon.

We first describe the TCN module. TCN has a convolution structure that controls the length of the sequence memory, which is a sequence modeling structure including one-dimensional convolution, causal convolution, and dilated convolution. Causal convolution ensures that the model can only use past information for prediction and cannot rely on future information. This allows the model to more effectively capture the causal relationships in time series data so that it is more suitable for time series modeling tasks. Dilated convolution addresses the issue of feature extraction limitations imposed by the size of the convolutional kernel in causal convolution. It can capture a larger receptive field. The formalization of feeding the input from the embedding layer into the TCN model can be represented as,

$$F(x_t) = (F * x)(x_t) = \sum_{i=1}^{k} f(x_t) \cdot x_{t-k \cdot d} \tag{3}$$

where $F = \{f_1, f_2, f_3, \ldots, f_k\}$ is the filter, $*$ is the convolutional operation, d is the expansion convolution and $t - k \cdot d$ is the direction of the past. When $d = 1$, a dilated convolution reduces to a regular convolution. Larger dilations expand the convolutional network's receptive field, allowing the output of top layer to represent a wider range of inputs. Next, the TCN model's output is fed into the BiGRU model. GRU is a practical variant of the LSTM network. Compared to LSTM, GRU is easier to train and can significantly improve training efficiency [30]. The GRU has two gates: one is a reset gate r_t and the other is an update gate z_t. The update gate regulates the information preserved from the previous state h_{t-1} and the information received from the candidate state $\tilde{h}_t$ at the current time step. The reset gate determines whether the computation of the candidate state is dependent on the state of the previous time step. The specific calculation process of GRU is illustrated by

$$\begin{cases} r_t = \sigma\left(W_r \cdot [h_{t-1}, \tilde{x}_t]\right) \\ z_t = \sigma\left(W_z \cdot [h_{t-1}, \tilde{x}_t]\right) \\ \tilde{h}_t = \tanh\left(W_{\tilde{h}} \cdot [r_t \cdot h_{t-1}, \tilde{x}_t]\right) \\ h_t = (1 - z_t) \cdot h_{t-1} + z_t \cdot \tilde{h}_t \end{cases} \tag{4}$$

where $\tilde{x}_t = F(x_t)$, σ is a decay factor, $\tilde{x}_t$ is the input vector and h_t is the hidden state vector at time t. BiGRU consists of GRUs in two directions. This allows each output unit to

combine previous and current states to determine the value of the current state as

$$\begin{cases} \vec{h}_t = GRU\left(\tilde{x}_t, \vec{h}_{t-1}\right) \\ \overleftarrow{h}_t = GRU\left(\tilde{x}_t, \overleftarrow{h}_{t+1}\right) \\ h_t = \vec{h}_t \oplus \overleftarrow{h}_t \end{cases} \tag{5}$$

Considering the potential sudden bursts in the dataset, we also employ a multi-head attention mechanism. The single-head attention mechanism is formulated as

$$Attention(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right) \tag{6}$$

where Q, K, and V are the query vector matrix, key vector matrix, and value vector matrix, respectively. Multi-Head Attention utilizes multiple queries to perform parallel computations to select multiple pieces of information from the input. Each attention focuses on a different part of the input, which is then concatenated as

$$\begin{cases} MultiHead(Q, K, V) = \\ Concat\left(head_1, head_2, \ldots, head_h\right) W^O \\ head_i = Attention\left(Q_i, K_i, V_i\right) \end{cases} \tag{7}$$

where W^O is the weight matrix of the output.

Model Training and Inference. The loss function for this model is the mean absolute error (MAE). Equation is as follows.

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|Y_i - \widehat{Y}_i| \tag{8}$$

where Y_i is the real control sequence, and $\widehat{Y}_i$ is the estimated control sequence. The training process concludes when the model's loss converges. In the inference phase, the model preprocesses the original transaction logs by selecting a time slice length and counting the concurrency for each time slice. This statistical data is then added to the embedding vector list for evaluation.

V. Experimental Results

In this section, we evaluate Hammer from different aspects. The experiment setup is briefly described as follows.

Environment. We conduct the evaluation on an Aliyun ECS cluster with 5 nodes. Each node is an "ecs.e-c1m2.large" instance that has 2 vCPUs and 4GB memory with Ubuntu 22.04 LTS OS. The network bandwidth between nodes is about 100Mbps. For Fabric, one node is dedicated as the orderer and the others are configured as peer nodes. For Ethereum, all nodes are workers. For Neuchain, one node works as an epoch server, another works as a client proxy, and other nodes as block servers. These blockchains all belong to the non-sharded architecture.

Sharding. Our evaluation framework also supports sharded architecture. In the Meepo, a sharded blockchain, we deploy two shards, where three nodes are configured to participate in both consensus and transaction processing for both shards. The experiments are based on static sharding, as the shard scale and account distribution are static. We set 5,000 accounts in each shard. Then, let one account in each shard generate random transfers. In this paper, we do not specifically distinguish between intra-shard and inter-shard transactions, aiming to demonstrate the versatility of our evaluation framework.

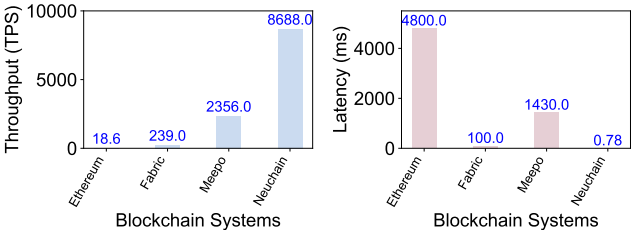


Fig. 6: Throughput and latency of different blockchains.

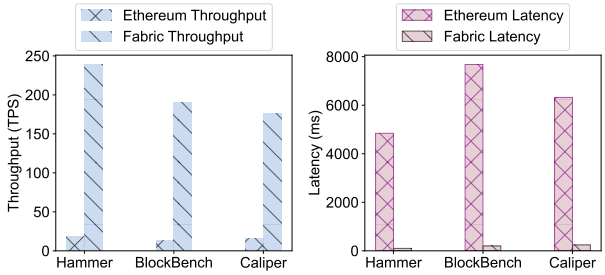


Fig. 7: Comparing the peak performance of the blockchains.

Workload. SmallBank is employed to simulate a basic banking system and offers several fundamental banking operations. Its primary operations typically include deposit, withdraw, transfer, and amalgamate. The access patterns of these four operations follows a uniform distribution.

A. Overall Performance

To validate the versatility of our evaluation framework, we conduct tests on the peak throughput and latency of various types of blockchain systems. All the experiments are performed three times. As illustrated in Fig. 6, Ethereum's throughput is the lowest at 18.6 TPS and it has the highest latency at 4.8 seconds due to the large overhead of the proof of work (PoW) consensus mechanism during the ordering phase. Meepo, employing sharding technology to improve throughput, also exhibit high latency. In contrast, Neuchain, leveraging a deterministic consensus mechanism and omitting the ordering phase, achieves a high throughput of 8688 TPS while maintaining low latency.

We compare Blockbench and Caliper, and conduct peak performance tests on Ethereum and Fabric, as these frameworks do not support testing for sharded architectures. The results of all three evaluation frameworks in Fig. 7 show that the performance Ethereum is the worst. This is primarily due to the time-consuming process of solving mathematical problems in the PoW mechanism, which leads to congestion in the Ethereum network and prolongs transaction confirmation times under heavy request loads. Since all evaluation frameworks

monitor transactions at the millisecond level, it is challenging to significantly distinguish their differences. In the performance evaluation of Fabric, Hammer reports a throughput of 239 TPS, significantly higher than 176 TPS reported by Caliper. This discrepancy may stem from two reasons: First, under heavy load, network congestion may lead to the loss of response information for transactions successfully submitted on the blockchain. Second, the event listening process consumes substantial computational resources, especially under heavy load, so that the resource is unavailable for processing transactions and blocks. The performance of Blockbench on Fabric is worse than that of Hammer, since the task processing algorithm of Blockbench is more complex. Compared to Blockbench, Hammer demonstrates superior performance due to its optimized algorithms.

B. Effect of Optimizations

In blockchain performance evaluation, generating workloads is a time-consuming process. If clients only have lower computational power, generating workload will require long time, so that the blockchain is always in an inactive state, which leads to a inaccurate measurement of the peak throughput. To address this problem, we propose an asynchronous signature to accelerate this process in Section III-D1. We also propose the pipelining technique to overlap the preparation phase and the execution phase in Section III-D2, which can further decrease the processing time. In Fig. 8, the y-axis represents the load generation time. The results show that by implementing asynchronous pipelining techniques (Asynchronous Pipeline) achieves approximately 6.88x acceleration compared to naive synchronous serial execution (Serial). Blockbench conducts tests using batch processing. However, during the execution phase, it may encounter high complexity issues due to matching and deletion operations, especially in a distributed environment. If a transaction to be matched does not exist locally, it is necessary to traverse the entire queue, thus increasing processing time and complexity. To address this issue, we present a task processing algorithm. The batch testing algorithm periodically monitors and collects new blocks. It then extracts transaction ids to match with a local queue and removes the corresponding ids after the successful matching. In Fig. 9, the x-axis represents the queue length (i.e., the number of transactions), the three differently colored bars indicate the number of transactions parsed from blocks, and the y-axis denotes the algorithm's execution time. We observe that with varying block quantities, the execution time of our algorithm remains stable and is reduced by 4x compared to the batch testing algorithm. This is attributed to the index hash tables utilized in our approach with increasing length to minimize conflicts, bringing the time complexity close to $O(1)$, whereas the execution time of the original algorithm exhibits linear growth.

C. Correctness

To verify the accuracy of our test results, we conduct validation on the Fabric blockchain system. The Hammer tool

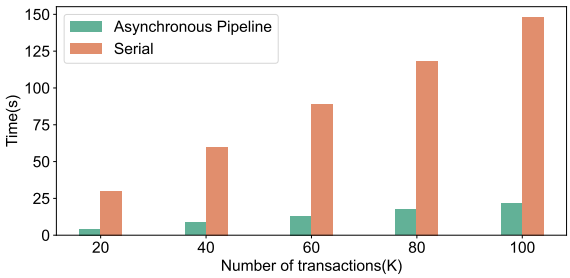


Fig. 8: Comparing asynchronous pipeline and serial workload generation method.

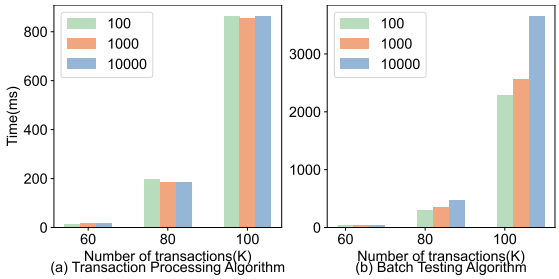


Fig. 9: Comparing task processing algorithm with batch testing algorithm.

is utilized to perform testing on Fabric. It is configured to process a total of 100,000 transactions at a rate of 600 TPS. After completing the tests, we use a Python script to analyze the logs of peer nodes in Fabric and compare the outcomes. The results demonstrate that our statistical data matches the log analysis results, thereby confirming the correctness of our system's test results.

D. Usability

To verify the usability of Hammer, we measure the changes in Hyperledger Fabric latency and throughput as the number of concurrent clients or threads increases or decreases. The experimental setup is the same as mentioned in V-C. Due to Ethereum's use of the Proof of Work (PoW) consensus mechanism, a characteristic feature is the generation of a new block every 15 seconds, leading to a relatively fixed transaction processing rate. Therefore, these limitations of Ethereum may affect the ability to validate the usability of Hammer. Fig. 10 shows that when the number of threads on the client is 2, throughput is highest and latency is lowest. However, throughput begins to decrease and latency increases as the number of threads increases. In fact, the client has 2 vCPUs, and when the number of threads is 2, each CPU core processes one thread, permitting parallel execution without resource contention. However, increasing the number of threads results in competition for CPU cores and increased scheduling overhead, which reduces throughput and increases latency. Fig. 10 shows that as the number of clients increases from 1 to 2, the throughput of the system peaks. However, when the number of clients reaches 3 or

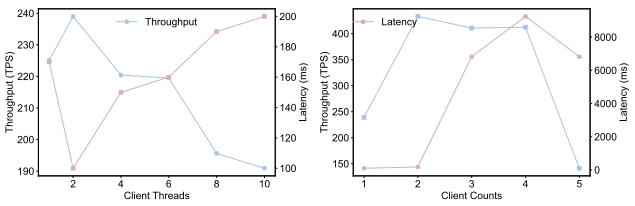


Fig. 10: Comparing the impact of varying thread or client counts on fabric throughput and latency.

4, the system latency increases significantly. Increasing the number of clients further to 5 results in a decrease in system throughput and a reduction in latency. To understand the reasons behind this interesting phenomenon, we examine the system logs and find that as the number of clients increases, the probability of transaction conflicts also increases, leading to a significant increase in latency. When the number of clients exceeds a threshold that exceeds the processing capacity of the nodes, the nodes reject some requests to prevent overload, resulting in a decrease in throughput and a reduction in latency. Our experimental results show that the blockchain system performs best with two clients and two threads per client under Smallbank workload, providing key guidance for developers in selecting and optimizing blockchain platforms.

E. Model Evaluation

TABLE III: Comparison of different methods on three datasets

Dataset	*Method*	*MAE*	*MSE*	*RMSE*	R^2
DeFi	Linear	1.224	2.392	1.547	0.0001
	RNN	1.058	1.574	1.255	-0.1861
	TCN	1.008	1.786	1.336	0.0042
	Transformer	1.395	3.187	1.785	-1.3029
	Ours	**0.378**	**0.259**	**0.508**	**0.8928**
Sandbox	Linear	0.311	6.959	2.638	0.7779
	RNN	0.255	12.530	3.549	0.6763
	TCN	0.233	5.609	2.368	0.7896
	Transformer	1.319	38.710	6.222	-3.304
	Ours	**0.136**	**1.417**	**1.191**	**0.9548**
NFTs	Linear	2.391	34.297	5.856	0.4988
	RNN	2.034	24.639	4.963	0.5357
	TCN	3.124	64.330	8.021	0.1458
	Transformer	3.834	66.141	8.133	-0.0002
	Ours	**0.7267**	**3.087**	**1.757**	**0.9548**

In our research, we construct three datasets in different applications: a decentralized finance dataset comprising 1,791 transactions, a sandbox game dataset containing 22,674 records, and an NFT dataset consisting of 233,014 NFT transactions. We pre-process the datasets by dividing them into hourly intervals and counting the number of transactions in each interval to form time series sequences based on hourly units. Subsequently, we employ the time series model to learn the temporal characteristics of these datasets. After comparing with existing advanced models such as TCN and Transformer, as shown in table III, our method significantly outperforms these models in metrics like MSE, MAE, and RMSE. An R-squared value close to 1 indicates that our model achieves a good fit. Our model does not perform well on the dataset of DeFi, possibly due to the limited amount of data. We use the model to generate and visualize the learned sequences. Fig. 11 shows that our model effectively captures burst events, long-term dependencies, and short-term dependencies, with a particularly notable performance in learning sudden bursts as plotted in Fig. 11b. The model can effectively predict future trends in real loads and extend time series, which is particularly suitable for addressing the issue of extending time series in blockchain real loads with unclear periodicity.

VI. RELATED WORK

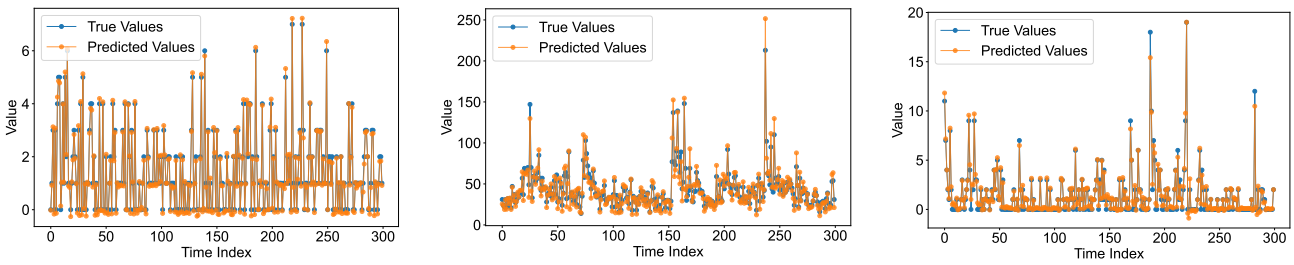
Blockchain-based Evaluation Framework. Besides the works mentioned in Section II, there are other representative works in this field. DAGBENCH [31] is the first performance evaluation framework for the DAG distributed ledger. BBS [32] leverage fuzzy set theory to identify important micro-architecture events after their significance is quantified by a machine learning based approach. BlockEmulator [33] is a tool for evaluating blockchain sharding protocols and mechanisms. Gromit [21] is a blockchain evaluation tool implemented in Go language. It regards the abstract model of blockchain system as a transaction structure, and evaluates seven famous blockchain systems.

Time-Series Workload Prediction. Time series database workload prediction has become a hot research area. It mainly includes query arrival rate prediction and resource utilization prediction. Yuan et al. [34] propose DBAugur, an adversarial-based trend forecasting system designed to predict the trends of diversified workloads. Antony et al. [35] utilize supervised machine learning to identify traits such as reoccurring patterns, shocks and trends that the workloads exhibit and account for those traits in the forecast. Li et al. [36] propose a LSTM model to predict practical storage workload. However, in the field of blockchain, time series prediction is mainly used to forecast cryptocurrency price trends, as shown in the paper [37], [38].

However, existing research has not fully considered the temporal characteristics of workload. To fill this research gap, our work focuses on learning and simulating real-time serial data loads to evaluate the performance of various blockchain platforms, thereby guiding developers to choose the blockchains that best suits their needs.

VII. CONCLUSION

In the paper, we present Hammer, a generic blockchain evaluation framework. To the best of our knowledge, we are the first evaluation framework that is able to support both non-sharding and sharding architectures. Hammer focuses on learning and simulating the time series of real workloads, and then transferring these characteristics to flexible synthetic workloads. Hammer leverages key techniques to accelerate its performance, including pipelined preparation and execution methods that combine asynchronous signature algorithms with



(a) DeFi (b) SandboxGames (c) NFTs

Fig. 11: Real sequence vs generated sequence.

pipelined execution to optimize computation and communication and a task processing algorithm that combines dynamic creation of hash indexes and fast matching of vector lists to achieve efficient task processing. Additionally, we are still working to improve it by opening its source codes with GPL licence on GitHub.[2] However, this research has this limitation, especially the dynamic hash table will lead to an increase in storage consumption. When dealing with large-scale transactions of long duration, the volume of the hash table will continue to expand. Although this expansion has no direct impact on performance evaluation, it does increase the storage pressure. Aiming at this problem, we plan to explore effective solutions in future work.

ACKNOWLEDGMENTS

This work is supported by the National Natural Science Foundation of China (62372097), and the Fundamental Research Funds for the Central Universities (N2416003). This project is also supported by the 2023-2024 Open Project of Key Laboratory Ministry of Industry and Information Technology-Blockchain Technology and Data Security 20242216.

REFERENCES

[1] Z. Peng, Y. Zhang, Q. Xu, H. Liu, Y. Gao, X. Li, and G. Yu, "Neuchain: a fast permissioned blockchain system with deterministic ordering," *Proceedings of the VLDB Endowment*, vol. 15, pp. 2585–2598, 2022.

[2] P. Zheng, Q. Xu, Z. Zheng, Z. Zhou, Y. Yan, and H. Zhang, "Meepo: Sharded consortium blockchain," in *2021 IEEE 37th International Conference on Data Engineering*, 2021, pp. 1847–1852.

[3] C. Fan, S. Ghaemi, H. Khazaei, and P. Musilek, "Performance evaluation of blockchain systems: A systematic survey," *IEEE Access*, vol. 8, pp. 126 927–126 950, 2020.

[4] T. T. A. Dinh, J. Wang, G. Chen, R. Liu, B. C. Ooi, and K.-L. Tan, "Blockbench: A framework for analyzing private blockchains," in *Proceedings of the 2017 ACM international conference on management of data*, 2017, pp. 1085–1100.

[5] M. Rasolroveicy, "Blockcompass: A benchmarking platform for blockchain performance," *Authorea Preprints*, 2023.

[6] V. Gramoli, R. Guerraoui, A. Lebedev, C. Natoli, and G. Voron, "Diablo-v2: A benchmark for blockchain systems," EPFL, Tech. Rep., 2022.

[7] K. Ren, J. F. Van Buskirk, Z. Y. Ang, S. Hou, N. R. Cable, M. Monares, H. F. Korth, and D. Loghin, "Bbsf: Blockchain benchmarking standardized framework," in *Proceedings of the 1st Workshop on Verifiable Database Systems*, 2023, pp. 10–18.

[2]https://github.com/btcly/BlockBenchMark

[8] Q. Wang, R. Li, Q. Wang, and S. Chen, "Non-fungible token (nft): Overview, evaluation, opportunities and challenges," *arXiv preprint arXiv:2105.07447*, 2021.

[9] C. R. Harvey, A. Ramachandran, and J. Santoro, *DeFi and the Future of Finance*. John Wiley & Sons, 2021.

[10] T. Min, H. Wang, Y. Guo, and W. Cai, "Blockchain games: A survey," in *2019 IEEE conference on games*, 2019, pp. 1–8.

[11] A. Klenik and I. Kocsis, "Porting a benchmark with a classic workload to blockchain: Tpc-c on hyperledger fabric," in *Proceedings of the 37th ACM/SIGAPP Symposium on Applied Computing*, 2022, pp. 290–298.

[12] C. Dannen, *Introducing Ethereum and solidity*. Springer, 2017.

[13] S. Nakamoto, "A peer-to-peer electronic cash system," *Decentralized business review*, 2008.

[14] E. Androulaki, A. Barger, V. Bortnikov, Y. Cachin *et al.*, "Hyperledger fabric: a distributed operating system for permissioned blockchains," in *Proceedings of the thirteenth EuroSys conference*, 2018, pp. 1–15.

[15] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears, "Benchmarking cloud serving systems with ycsb," in *Proceedings of the 1st ACM symposium on Cloud computing*, 2010, pp. 143–154.

[16] D. E. Difallah, A. Pavlo, C. Curino, and P. Cudre-Mauroux, "Oltp-bench: An extensible testbed for benchmarking relational databases," *Proceedings of the VLDB Endowment*, vol. 7, pp. 277–288, 2013.

[17] S. T. Leutenegger and D. Dias, "A modeling study of the tpc-c benchmark," *ACM Sigmod Record*, vol. 22, pp. 22–31, 1993.

[18] Huawei. (2021-05) Hyperledger caliper.

[19] D. Saingre and J.-M. Ledoux, "Bctmark: a framework for benchmarking blockchain technologies," in *2020 IEEE/ACS 17th International Conference on Computer Systems and Applications*, 2020, pp. 1–8.

[20] J. A. Chacko, R. Mayer, and H.-A. Jacobsen, "Why do my blockchain transactions fail? a study of hyperledger fabric," in *Proceedings of the 2021 international conference on management of data*, 2021, pp. 221–234.

[21] B. Nasrulin, M. De Vos, G. Ishmaev, and J. Pouwelse, "Gromit: Benchmarking the performance and scalability of blockchain systems," in *2022 IEEE International Conference on Decentralized Applications and Infrastructures*, 2022, pp. 56–63.

[22] J. Sedlmeir, P. Ross, A. Luckow, J. Lockl, D. Miehle, and G. Fridgen, "The dlps: a new framework for benchmarking blockchains," *Hawaii International Conference on System Sciences*, p. 10, 2021.

[23] J. L. Carlson, *Redis in Action*. USA: Manning Publications Co., 2013.

[24] M. Reichardt, M. Gundall, and H. D. Schotten, "Benchmarking the operation times of nosql and mysql databases for python clients," in *IECON 2021–47th Annual Conference of the IEEE Industrial Electronics Society*, 2021, pp. 1–8.

[25] L. Hochstein and R. Moser, *Ansible: Up and Running: Automating configuration management and deployment the easy way*. " O'Reilly Media, Inc.", 2017.

[26] M. Chakraborty and A. P. Kundan, "Grafana," in *Monitoring Cloud-Native Applications: Lead Agile Operations Confidently Using Open Source Software*. Springer, 2021, pp. 187–240.

[27] J. Turnbull, *Monitoring with Prometheus*. Turnbull Press, 2018.

[28] S. Bai, J. Z. Kolter, and V. Koltun, "An empirical evaluation of generic convolutional and recurrent networks for sequence modeling," *arXiv preprint arXiv:1803.01271*, 2018.

[29] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," *arXiv preprint arXiv:1412.3555*, 2014.
[30] R. Dey and F. M. Salem, "Gate-variants of gated recurrent unit (gru) neural networks," in *2017 IEEE 60th international midwest symposium on circuits and systems*, 2017, pp. 1597–1600.
[31] Z. Dong, E. Zheng, Y. Choon, and A. Y. Zomaya, "Dagbench: A performance evaluation framework for dag distributed ledgers," in *2019 IEEE 12th international conference on cloud computing*, 2019, pp. 264–271.
[32] L. Zhu, C. Chen, Z. Su, W. Chen, T. Li, and Z. Yu, "Bbs: Micro-architecture benchmarking blockchain systems through machine learning and fuzzy set," in *2020 IEEE International Symposium on High Performance Computer Architecture*, 2020, pp. 411–423.
[33] H. Huang, G. Ye, Q. Chen, Z. Yin, X. Luo, J. Lin, T. Li, Q. Yang, and Z. Zheng, "Blockemulator: An emulator enabling to test blockchain sharding protocols," *ArXiv*, vol. abs/2311.03612, 2023.
[34] Y. Gao, X. Huang, X. Zhou, X. Gao, G. Li, and G. Chen, "Dbaugur: An adversarial-based trend forecasting system for diversified workloads," in *2023 IEEE 39th International Conference on Data Engineering*, 2023, pp. 27–39.
[35] A. S. Higginson, M. Dediu, O. Arsene, N. W. Paton, and S. M. Embury, "Database workload capacity planning using time series analysis and machine learning," in *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, 2020, pp. 769–783.
[36] L. Ruan, Y. Bai, S. Li, S. He, and L. Xiao, "Workload time series prediction in storage systems: a deep learning based approach," *Cluster Computing*, pp. 1–11, 2021.
[37] I. E. Livieris, E. Pintelas, S. Stavroyiannis, and P. Pintelas, "Ensemble deep learning models for forecasting cryptocurrency time-series," *Algorithms*, vol. 13, p. 121, 2020.
[38] R. K. Malladi and P. L. Dheeriya, "Time series analysis of cryptocurrency returns and volatilities," *Journal of Economics and Finance*, vol. 45, pp. 75–94, 2021.