mathematics

MDPI

Article

BlockLoader: A Comprehensive Evaluation Framework for Blockchain Performance Under Various Workload Patterns

Gang Wang [1], Yanfeng Zhang [1,*], Chenhao Ying [2], Qinnan Zhang [3], Zhiyuan Peng [2], Xiaohua Li [1] and Ge Yu [1]

[1] School of Computer Science and Engineering, Northeastern University, Shenyang 110167, China; 1910636@stu.neu.edu.cn (G.W.); lixiaohua@mail.neu.edu.cn (X.L.); yuge@mail.neu.edu.cn (G.Y.)
[2] Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai 200240, China; yingchenhao@sjtu.edu.cn (C.Y.); pzy2000@sjtu.edu.cn (Z.P.)
[3] Beijing Advanced Innovation Center for Future Blockchain and Privacy Computing, Beihang University, Beijing 100191, China; zhangqn@buaa.edu.cn
* Correspondence: zhangyf@mail.neu.edu.cn

Abstract: Hyperledger Fabric is one of the most popular permissioned blockchain platforms widely adopted in enterprise blockchain solutions. To optimize and fully utilize the platform, it is desired to conduct a thorough performance analysis of Hyperledger Fabric. Although numerous studies have analyzed the performance of Hyperledger Fabric, three significant limitations still exist. First, existing blockchain performance evaluation frameworks rely on fixed workload rates, which fail to accurately reflect the performance of blockchain systems in real-world application scenarios. Second, the impact of extending the breadth and depth of endorsement policies on the performance of blockchain systems has yet to be adequately studied. Finally, the impact of node crashes and recoveries on blockchain system performance has yet to be comprehensively investigated. To address these limitations, we propose a framework called BlockLoader, which offers seven different distributions of load rates, including linear, single-peak, and multi-peak patterns. Next, we employ the BlockLoader framework to analyze the impact of endorsement policy breadth and depth on blockchain performance, both qualitatively and quantitatively. Additionally, we investigate the impact of dynamic node changes on performance. The experimental results demonstrate that different endorsement policies exert distinct effects on performance regarding breadth and depth scalability. In the horizontal expansion of endorsement policies, the OR endorsement policy demonstrates stable performance, fluctuating around 88 TPS, indicating that adding organizations and nodes has minimal impact. In contrast, the AND endorsement policy exhibits a declining trend in performance as the number of organizations and nodes increases, with an average decrease of 10 TPS for each additional organization. Moreover, the dynamic behaviour of nodes exerts varying impacts across these endorsement policies. Specifically, under the AND endorsement policy, dynamic changes in nodes significantly affect system performance. The TPS of the AND endorsement policy shows a notable decline, dropping from 79.6 at 100 s to 41.96 at 500 s, reflecting a reduction of approximately 47% over time. Under the OR endorsement policy, the system performance remains almost unaffected.

Keywords: hyperledger fabric; BlockLoader; endorsement policy; node dynamic changes

MSC: 68M14

check for updates

Citation: Wang, G.; Zhang, Y.; Ying, C.; Zhang, Q.; Peng, Z.; Li, X.; Yu, G. BlockLoader: A Comprehensive Evaluation Framework for Blockchain Performance Under Various Workload Patterns. *Mathematics* **2024**, *12*, 3403. https://doi.org/10.3390/math12213403

Academic Editor: Vincenzo Vespri

Received: 28 September 2024
Revised: 24 October 2024
Accepted: 24 October 2024
Published: 31 October 2024

Copyright: © 2024 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

Blockchain is a form of distributed ledger technology that is decentralized, data-sharing, and tamper-resistant. These characteristics have demonstrated significant potential across various domains, including retail, healthcare, and financial applications [1]. Among many blockchain platforms, Hyperledger Fabric (HLF) is a representative consortium blockchain platform popular due to its highly modular and configurable architecture, making it widely used in enterprise-level solutions. Furthermore, HLF exhibits robust

performance, scalability, and security characteristics [2]. To comprehensively understand the performance of HLF, researchers have utilized popular performance evaluation tools such as Caliper [3], BlockBench [4] and Hammer [5]. These tools facilitate the evaluation of HLF performance under various workloads and configurations. Most of the evaluation frameworks [6–8] provide synthetic workloads, with only a few offering real-world workloads [5,9]. However, these frameworks typically utilize fixed workload rates, which fail to adequately consider the impact of varying load rates on the system's performance under test (SUT). To illustrate this phenomenon, we analyze the YCSB benchmark provided by BlockBench, which is widely used in databases. Understanding the real workload rates of different blockchain-based applications provides a more comprehensive view of their performance in real-world scenarios. As shown in Figure 1, the transaction- sending pattern exhibits pronounced fluctuations within the initial 20-s period, which can be attributed to the system's cold start phase. After this initial phase, the transaction rate achieves a stable equilibrium. The sending pattern of YCSB does not align with the workload patterns observed in real-world scenarios. Consequently, the existing evaluation framework, which has a fixed workload rate, cannot meet the real requirements of blockchain applications.

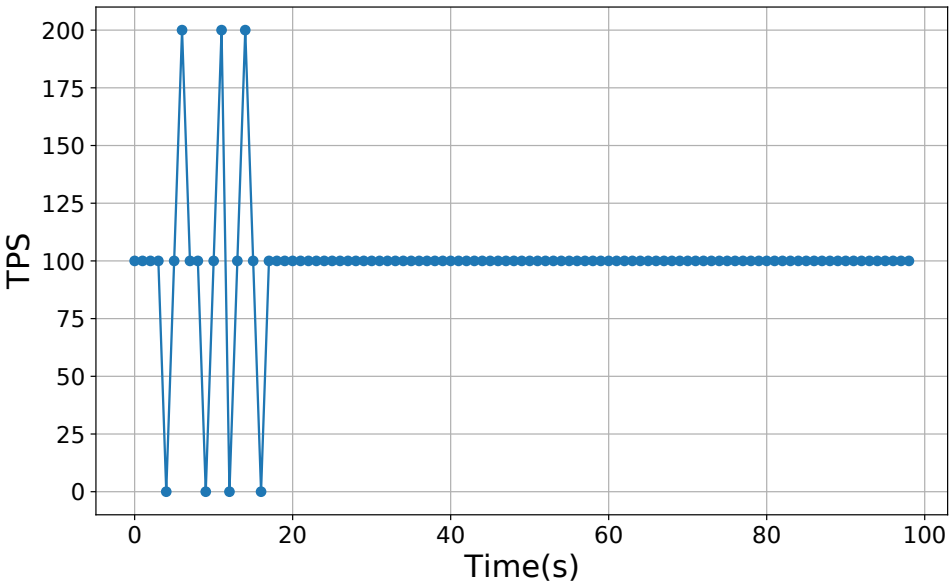


Figure 1. Analyzing workload patterns in YCSB.

Furthermore, while there is extensive research on evaluating the performance of HLF, this work mainly focuses on configuration parameters and consensus algorithms [10–12]. However, there is a significant gap in existing research regarding the impact of endorsement policies on blockchain performance. This gap arises from the complexity of endorsement policies in Hyperledger Fabric, which can be nested, combined, and scaled both horizontally and vertically, making their performance impact challenging to analyze.

Finally, dynamic changes in nodes pose challenges to blockchain performance evaluation. Node crashes and recoveries disrupt the stability of consensus algorithms and complicate load balancing.

To solve these challenges, we propose an evaluation framework called BlockLoader. This framework supports a variety of workload rate distributions and provides insights into the impact of three different endorsement policies on HLF performance. Additionally, we examine the effects of dynamic node changes on HLF performance and system stability.

The main contributions of this paper are as follows:

- We propose the BlockLoader evaluation framework, which offers various workload distribution patterns, including uniform, linear, single-peak, and multi-peak patterns, to simulate sudden changes or periodic workloads. These patterns provide more

realistic performance evaluations, addressing the limitations of existing frameworks that rely solely on fixed workload patterns.

- We perform qualitative and quantitative analyses of the impact of different endorsement policies on blockchain performance. Compared to previous research focusing on the effects of system configuration parameters and consensus algorithms, we fill this gap in the literature and provide new insights for optimizing blockchain systems.
- We propose a dynamic crash fault tolerance evaluation scheme that evaluates the performance of the blockchain system under node dynamics while ensuring that the system continues to operate normally. This scheme addresses the shortcomings of existing frameworks regarding crash fault tolerance evaluation.

The remainder of this paper is structured as follows: Section 2 provides the necessary background. The related work is introduced in Section 3. Section 4 describes the architecture and workflow of BlockLoader. Section 5 presents a comprehensive examination of the design of workload distribution patterns and the generation strategies. Section 6 presents a qualitative analysis that explores the impact of endorsement policies and node variations on the performance of HLF. Section 7 evaluates the performance of HLF under various workload distribution patterns and conducts a quantitative analysis of the impact of endorsement policies and node dynamics on performance. Our research concludes in Section 8. The complete implementation code is available on GitHub (https://github.com/shuai2077/BlockLoader (accessed on 1 October 2024)), allowing readers to delve deeper into our algorithms and experimental procedures.

2. Background

2.1. Hyperledger Fabric

Hyperledger Fabric(HLF) is a popular open-source permissioned blockchain system established under the Linux foundation [13]. It is the first distributed ledger platform that supports the writing of smart contracts in multiple general-purpose programming languages such as Go (https://golang.org (accessed on 1 October 2024)) and Java (https://www.java.com (accessed on 1 October 2024)).

The system comprises several key components: clients, peers, orderers, a gateway (GW), and a certificate authority (CA). Next, we will briefly explain the roles of each of these components. The Certificate Authority (CA) is a fundamental component of a permissioned blockchain. It generates and verifies identity credentials in the blockchain network, including public-private key pairs, and certificates. Additionally, in Hyperledger Fabric V2.x, the gateway (GW) is introduced to simplify client interactions with the Fabric network. It provides a streamlined API for transaction submission and querying while managing node connections and handling errors. In Hyperledger Fabric, peers are network nodes responsible for maintaining the ledger. They typically serve as endorsers and validators within the network. Orderers are network nodes that provide consensus services, implementing protocols like Kafka [14] or PBFT [15] to achieve agreement across the blockchain network.

Transaction Overview. In a Hyperledger Fabric (HLF) network, the transaction execution process begins with clients submitting transactions to the decentralized system, which operates without a central authoritative node and relies on a consensus mechanism to achieve transaction agreement. Note that we will list some concepts that are often ignored. (1) The chaincode implements the application's business logic, with each invocation treated as a transaction. The resulting state updates are managed by a versioned key-value database, such as [16] or CouchDB [17], which ensures consistency and accuracy throughout the execution and recording of transactions. (2) To ensure the consistency of transaction states, each peer node in the network maintains a synchronized copy of the ledger.

As the transaction enters the simulation execution phase, endorsing peers generate read-write sets based on the current world state. The endorsement policy then defines the criteria under which a transaction is considered valid, ensuring the integrity and reliability of the entire transaction execution process. Upon accumulating sufficient transactions, the

orderer service sequences these transactions and packages them into blocks. These blocks are subsequently disseminated to all peers in the network via the gossip protocol. Peer communication occurs through designated channels to reach a consensus on block height quickly, ensuring efficient and secure data exchange.

The transaction execution flow in HLF is divided into three stages: Execution, Ordering, and Validation. This process is known as Execute-Order-Validate (E-O-V), as shown in Figure 2. The following describes each stage in detail.

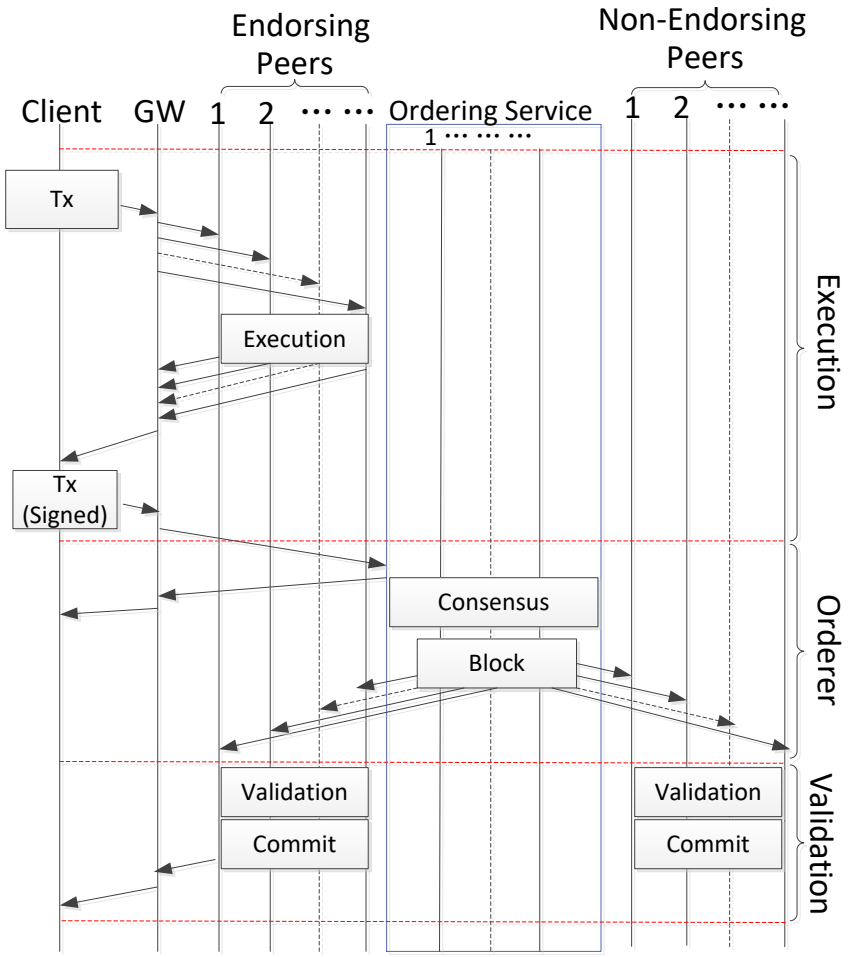


Figure 2. Execution flow of hyperledger fabric.

Execution Phase. In the execution phase, the client creates a transaction proposal and sends it to the GW. In Hyperledger Fabric, the deployment of the GW is adaptable. It can be configured for endorsing peers as well as for non-endorsing peers. The GW receives transaction proposals and forwards them to the endorsing peers specified by the endorsement policy established during channel initialization. Each endorsing peer simulates the execution of transactions based on the current world state, generating read/write sets corresponding to each key in the transaction. After completing the endorsement, the endorsing peers send the endorsement results, including the read-write sets and signatures, to the GW. Once the GW has collected enough endorsements, it packages the endorsed transaction proposal into a transaction envelope and sends it to the client for signature. After the client signs the transaction envelope, it is returned to the GW, which then forwards it to the ordering service.

Ordering Phase. The ordering service uses a consensus protocol to sequence the transactions received from the GW. Factors such as block size, block interval, and the number of transactions control the generation of transaction blocks. Finally, the ordering

service broadcasts the generated blocks to all peers for independent validation during the validation phase.

Validation Phase. Peers receive blocks directly from the ordering service or via the gossip protocol from other peers. The critical validation steps during the validation phase are as follows. First, it verifies that each transaction meets the endorsement requirements, followed by Multi-Version Concurrency Control (MVCC) validation. The valid transactions update the world state if the validations are successful. Finally, one of the peers notifies the client via the GW that the transaction has been successfully committed to the blockchain.

2.2. Hyperledger Caliper

Hyperledger Caliper is a blockchain performance benchmarking tool that evaluates the performance of blockchain systems using custom test cases. Its primary performance metrics include throughput and latency. The architecture of Hyperledger Caliper is shown in Figure 3.

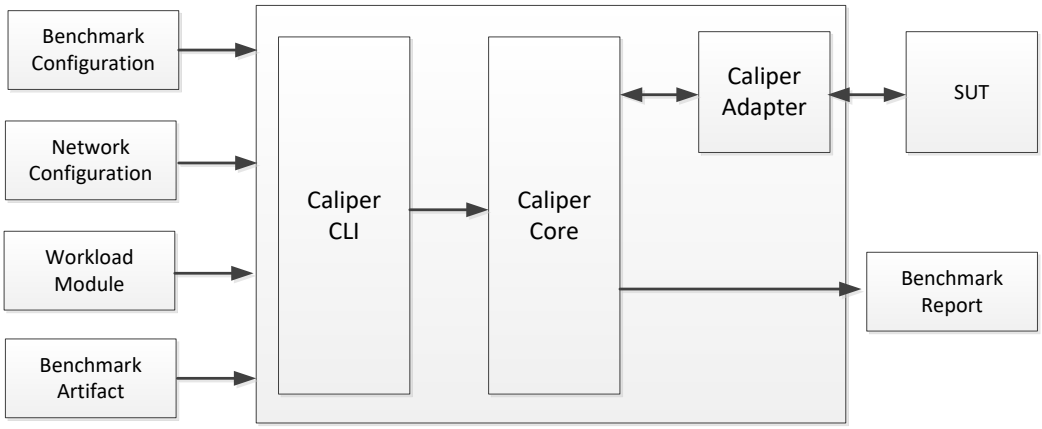


Figure 3. Architecture of Hyperledger Caliper.

Caliper has three main components: Caliper CLI, Core, and Adaptor. We will briefly introduce the functions of these three components. First, Caliper CLI provides a command-line interface for executing benchmark tests. Second, Caliper Core is a continuous monitoring and response interface during system performance evaluation. Finally, the Caliper Adaptor is an adapter connecting the System Under Test (SUT) with the benchmarking tool. In addition, several configuration file modules are also introduced. The Benchmark configuration file defines parameters such as transaction send rate, total transaction count, number of test rounds, and duration. The network configuration file specifies the system topology in testing, including the network location and identity of peers. The Benchmark Artifacts directory stores the public/private key pairs and certificates required for system interactions. The Workload Module is responsible for generating and submitting transactions.

Caliper testing workflow. The workflow of Hyperledger Caliper can be divided into three main phases: the preparation phase, the execution phase, and the result analysis phase. In the preparation phase, it is necessary to ensure the correct configuration of the test environment, including the installation of Caliper, the setting of the working directory, and the configuration files required for setup (e.g., network configuration file, benchmark configuration file, workload module). Additionally, we need to ensure that the blockchain network under test has been deployed and is running. In the HLF network, the appropriate smart contracts are deployed according to the specifications in the $config.yaml$ file, ensuring that they are installed and instantiated before testing begins. In the execution stage, it represents the critical phase of benchmark testing. First, Caliper generates and submits transactions to the network based on the rules specified in the workload module. Second, during the transaction's execution, Caliper monitors the transaction's progress in real-time, including recording the submission time, confirmation time, success and failure status. Upon completion of the test, Caliper automatically aggregates the execution results

of all transactions and compiles them into comprehensive performance evaluation data. In the result analysis phase, Caliper generates a detailed report that includes metrics such as throughput, latency, and success rate, among other performance indicators.

3. Related Work

3.1. Performance Evaluation Tools

The existing evaluation methods for blockchain performance can be broadly categorized into empirical evaluation and analytical modeling. Analytical modeling primarily includes three stochastic models: Markov chains, queuing models, and stochastic Petri nets (SPNs). These models offer a theoretical perspective on system behaviour, but applying them to Distributed Ledger Technologies (DLTs) poses challenges due to the diverse underlying architectures of different DLTs. As a result, these methods often face difficulties in providing accurate and comparable performance metrics across varying DLT systems [18].

Empirical evaluation methods include benchmarking, monitoring, experimental analysis, and simulation. Among these, simulation techniques provide lower accuracy than other empirical methods, while live monitoring combined with experimental analysis has been identified as more effective for evaluating the performance of public DLTs [19]. Our work follows this empirical evaluation approach to analyze Hyperledger Fabric's performance in practical settings.

Benchmarking tools for blockchain performance can be classified into three main categories: private blockchain benchmarks, general-purpose benchmarking frameworks, and domain-specific tools. Private blockchain benchmarks like BlockBench [4] focus on evaluating systems through abstraction layers, adapting distributed database workloads for blockchain environments, though they often struggle with scalability. General-purpose frameworks, such as Hyperledger Caliper [3], BCTMark [6], HyperBench [20], and Diablo [21], provide flexibility in evaluating a range of blockchain solutions. While Caliper and HyperBench are known for their usability, their scalability is limited, and BCTMark remains in the experimental phase. Domain-specific tools, such as DAGBench [22] for DAG-based blockchains, Chainhammer [23], Quorum Profiling [24], and GoHammer [25] for Ethereum-based systems, offer targeted performance insights but often require complex setups. Each tool has strengths and limitations, serving different evaluation needs in blockchain research.

3.2. Fabric Performance Evaluation

Existing research on Hyperledger Fabric has conducted extensive analyses across various performance dimensions, including throughput, latency, and scalability. For instance, Murat Kuzlu et al. [26] performed a comprehensive analysis focusing on key performance metrics such as throughput, latency, and scalability, providing insights into Fabric's behavior under different workload scenarios. Similarly, Ju Won Kim et al. [27] evaluate the performance of an NFT trading platform built on Hyperledger Fabric, analyzing its behavior under real-world transaction conditions, particularly highlighting findings related to throughput and latency. Harris et al. [28] compared different ordering services (Solo, Kafka, Raft) under distinct endorsement policies, adding to the understanding of how consensus mechanisms interact with Fabric's performance. Paper [18] systematically reviewed the state of the art on the blockchain Hyperledger Fabric performance evaluation from several different perspectives.

In addition, several other vital studies [10–12,29–31] contribute to this area. Refs. [10,30] each investigate the impact of real-world workloads on Hyperledger Fabric's performance, specifically in the contexts of IoT and healthcare scenarios. Ref. [11] examines how variations in resource capacity affect performance, while [31] focuses on the effects of high-conflict workloads on the system's performance. These studies collectively provide valuable insights into how different application contexts and system conditions influence the efficiency and scalability of the Hyperledger Fabric platform.

However, the existing research still has some gaps. To address these gaps, we investigated the impact of both horizontal and vertical expansion of endorsement policies on performance. In addition, we explore how dynamic nodes affect system performance, and provide new insights and recommendations for optimizing Hyperledger Fabric performance.

4. System Architecture

4.1. Overview and Key Components

BlockLoader is an evaluation platform with five essential components: Launcher, Configuration Handler, Workload Executor, Caliper Adapter and Performance Report. The system architecture of BlockLoader is shown in Figure 4.

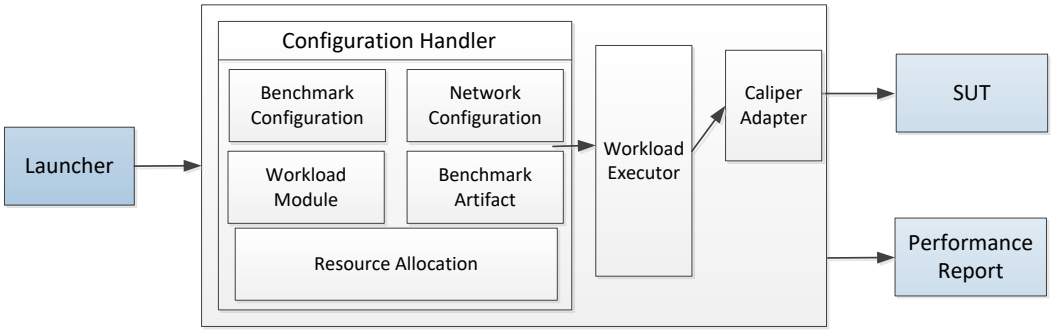


Figure 4. System architecture of BlockLoader.

Launcher. The Launcher serves as the entry point for BlockLoader, which is responsible for initializing the resource configuration of the Hyperledger Fabric (HLF) network, including CPU, memory, and storage resources. When triggered, the Launcher parses the user-provided configuration file, including resource allocation task details, network topology, benchmark configuration, and workload attributes. This process ensures that the HLF network test environment resources are accurately allocated and properly configured.

Configuration Handler. The Launcher transfers configuration data to the Configuration Handler, responsible for processing these configurations. The Configuration Handler's tasks include allocating computing and storage resources and deploying the blockchain test environment. Subsequently, the Configuration Handler parses the benchmark and network configuration files, enabling the definition of test parameters such as the number of test rounds, the transaction volume, and the test duration. Next, the Configuration Handler processes the workload module and transmits the generated workload profile to the Workload Executor. Finally, it utilizes the Benchmark Artifact to configure the identity credentials required by the HLF system.

Workload Executor. The Workload Executor is responsible for generating workloads with time series characteristics. Specifically, the Workload Executor first reads the workload profile and then generates the load data structure for a specific chaincode (smart contract) based on that profile. Subsequently, the actual workload is generated based on the time series distribution, which determines the quantity of workload over time. The content of the workload, including specific functions (e.g., the transferFunds function) and parameters (e.g., the transfer amount and the recipient), is determined by the workload operation. Note that in this paper, we use the SmallBank chaincode. The time series distribution characteristics of the workload consist of one or more stages. Each stage represents a period with a fixed transaction sending rate. The specific workload details are described in Section 5.

Caliper Adapter. Caliper Adaptor serves as a bridge between the evaluation framework and the system under test (SUT). It is responsible for managing the lifecycle of transactions, including sending transactions to the SUT, monitoring transaction status, collecting performance data, and processing responses. The adaptor enables seamless interaction between the evaluation framework and various blockchain platforms by hiding the details specific to each platform.

Performance Report. The Performance Report collects and records performance indicators and resource utilization for each stage and its timestamp. By aggregating the data in chronological order, users can analyze and observe dynamic performance changes and resource utilization fluctuations under the load time series distribution. The report includes key metrics such as resource usage, maximum throughput, average throughput, and latency.

4.2. Execution Workflow

Figure 5 illustrates the BlockLoader's execution workflow, which consists of three phases: preparation, execution, and report generation.

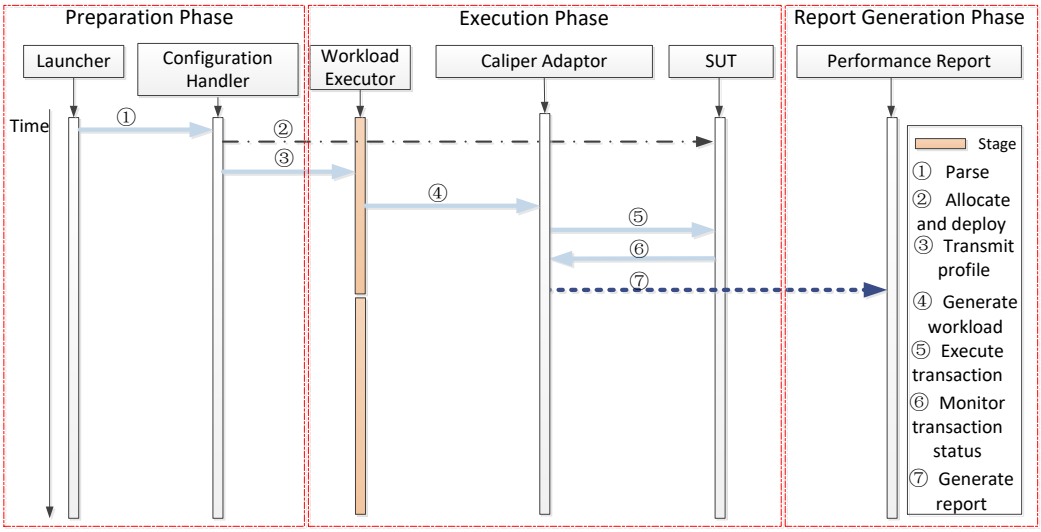


Figure 5. Execution flow of BlockLoader.

4.2.1. Preparation Phase

The launcher and configuration handler modules are integrated into the client in the preparation phase. Launcher performs step ① to parse the user-defined configuration file. These configuration files typically contain the user's definition of benchmarks and System Under Test (SUT) network configuration (e.g., Benchmark Configuration and Network Configuration). The parsing process generates the essential configuration files. It accurately defines the resource allocations for the System Under Test (SUT), ensuring the system is appropriately prepared for subsequent benchmarking tasks. The Configuration Handler receives input from the Launcher. Step ② allocates resources (e.g., compute and storage) based on the SUT resource configuration file and deploys the SUT network according to the network configuration file. In addition, the Configuration Handler configures evaluation parameters for the BlockLoader framework, such as the number of test rounds, transactions, and duration. In step ③, the workload configuration file is transmitted to the Workload Executor. This file specifies critical parameters such as the read-write ratio, operation distribution, and the distribution of transmission rates.

4.2.2. Execution Phase

The execution phase is divided into the load generation module and the transaction execution module. The workload executor is responsible for the load generation module. The workload executor contains two queues: the execution queue, where generated workloads are sequentially added for execution, and the collection queue, which stores the state of initialized transactions. The details of the workload generation process are provided in Section 5. In the execution phase, the key of step ④ is that a thread pushes the generated workload into the execution queue. When the trigger time of the workload arrives, another thread pulls the workload from the queue and pushes it to the Caliper Adaptor. Subsequently, step ⑤ involves sending these workloads to the System Under Test (SUT) via the

Caliper Adaptor. Step ⑥ continuously monitors the execution status of transactions and synchronizes the update of transaction status identifiers in the collection queue.

4.2.3. Report Generation Phase

In the report generation phase, step ⑦ analyzes the transaction status in the collection queue, calculates throughput and latency, merges the captured resource usage data into a report, and then sends the report to the performance report component. Since the report corresponds to each stage of the workload generation, the report execution component collects data reports from each stage and organizes them sequentially by timestamp. This temporal arrangement reflects the dynamic performance changes of the blockchain system throughout the testing process, providing a detailed foundation for further performance analysis.

5. Workload Design

5.1. Workflow of Workload Executor

Figure 6 illustrates the core workflow of the Workload Executor component. This component provides various load rate distributions to simulate different load scenarios.

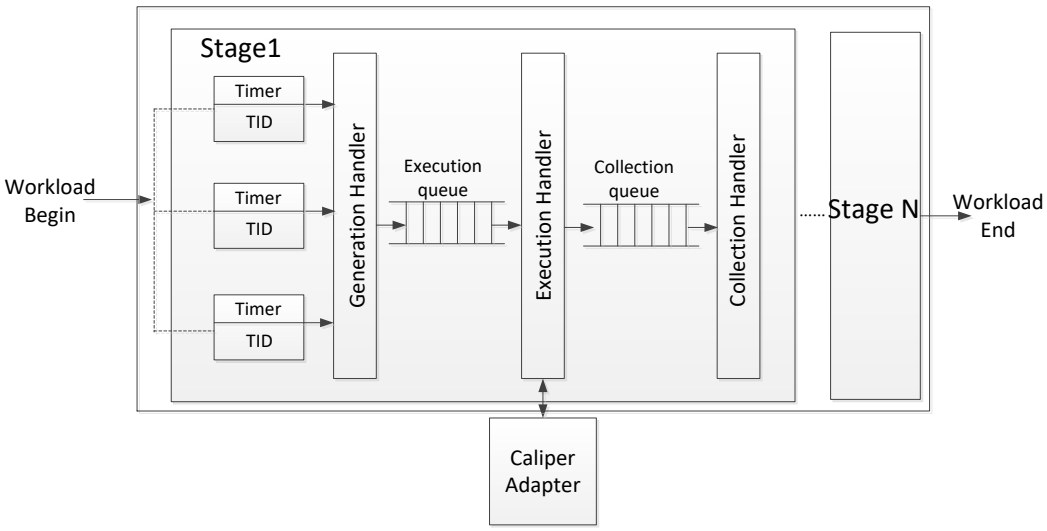


Figure 6. Workflow of the Workload Executor.

The workload executor consists of three primary phases: generation, execution, and status collection. The workload generation phase involves the following key steps. First, the system assigns a monotonically increasing task *id* (e.g., TID) to each workload. Then, based on the specified distribution of load rate, a timestamp is generated for each TID, determining the execution time of the corresponding workload. When the system time reaches the timestamp, a timer triggers the execution of the corresponding workload. The Generation Handler generates the workload and binds the timestamp to the corresponding TID, forming a complete workload item. These prepared workload items are then ordered by their $TIDs$ and sequentially added to the Execution Queue.

In the execution phase of the workload process, the Execution Handler continuously monitors workloads to ensure that they are executed at their scheduled time stamps. When the Execution Handler detects a workload whose timestamp matches the current system time, it immediately retrieves the workload. It engages the Caliper Adapter to initiate testing on the System Under Test (SUT). During the status collection phase, the Caliper Adaptor monitors state changes in transaction workloads on the blockchain. It captures real-time state information and transmits these updates to the Collection Queue for further processing. After completing the first stage (e.g., Stage 1) of testing, the system retrieves all transaction state data from the Collection Queue to calculate throughput and latency metrics. The Caliper Adaptor then compiles these results and forwards them to the Collection Handler for further analysis and reporting. Note that each stage generates a

separate performance report, and the aggregation of all stages reflects the overall workload rate distribution.

5.2. Workload Distribution Pattern

BlockLoader allows users to customize workload distribution patterns and generate diverse workloads. We have provided several general workload distribution patterns in BlockLoader, and their distribution histograms are shown in Figure 7.

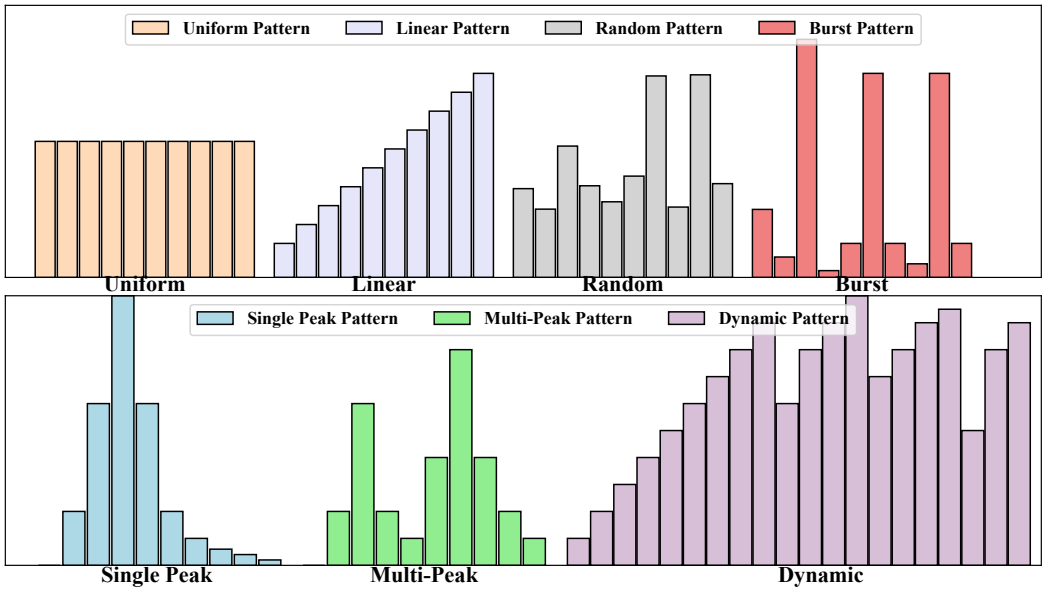


Figure 7. Histograms of various workload distribution patterns.

Uniform Pattern. The Uniform Pattern describes a scenario where workload or request dispatch frequency remains constant over a specified period. In this pattern, the same number of requests or workloads are sent in each time unit, ensuring a consistent transmission rate throughout the specified duration.

Burst Pattern. The Burst Pattern characterizes the phenomenon of a rapid escalation in workload over an extremely short period.

Dynamic Pattern. The Dynamic Pattern utilizes a feedback-based Caliper Adaptor to optimize the send rate for maximum performance without overloading the SUT. This pattern incrementally increases the send rate until a decrease in throughput is detected, indicating system saturation, at which point it reduces the load.

Linear Pattern. The Linear Pattern is a method of progressively adjusting the send rate linearly, either increasing or decreasing between defined start and target values. This pattern systematically adjusts send rates linearly, allowing for controlled observation of how varying load intensities affect system performance.

Random Pattern. The Random Pattern refers to a workload sending rate that follows a random distribution, simulating the system's behaviour under non-determinate conditions. This pattern is designed to evaluate the resilience and stability of the blockchain system under varying random workloads, providing insight into its performance in non-deterministic environments.

Single-Peak Pattern. The single-peak pattern refers to a distribution where the quantity of workloads exhibits a bell-shaped curve, with a distinct peak at the centre and symmetrically decreasing sides. This pattern is closely related to the Gaussian distribution (aka, the normal distribution), as the Gaussian distribution represents the quintessential form of a single-peak symmetric distribution.

Multi-Peak Pattern. The multi-peak pattern refers to multiple local maxima or peaks in workload distribution, indicating significant variations in workload intensity over time. This pattern is often observed in scenarios where different phases of activity or usage

peaks occur at distinct intervals. For instance, in blockchain applications, particularly in decentralized finance (DeFi) platforms, the multi-peak pattern may arise during high user activity periods, such as bursts of transaction volume during large contract executions. In addition, the multi-peak pattern may exhibit periodicity, reflecting cyclical changes in workload intensity corresponding to user behaviour or system events.

6. Endorsement Policies and Dynamic Changes in Nodes

6.1. Endorsement Policies

In Hyperledger Fabric, during the pre-execution phase, endorsement peers sign the results of a simulated execution. The gateway collects sufficient endorsements (i.e., meeting the requirements of the endorsement policy) and submits the endorsement transaction to the ordering service (i.e., Orderer). In the validation phase, the ordering service batches the transaction and distributes it to all peer nodes. The peer nodes then validate the transaction against the endorsement policy to ensure it has received the required endorsements. If the validation is successful, the transaction is committed to the ledger. Fabric provides three types of endorsement policies: AND, OR, and $K\text{-}OutOf$. The AND policy requires that all specified peers within an organization must endorse the transaction. The OR policy allows the transaction to be endorsed by any specified peers. The $K\text{-}OutOf$ policy mandates that at least K of the specified peers must endorse the transaction to satisfy the endorsement requirement.

While it is recognized that varying endorsement policies can influence the performance of Hyperledger Fabric, the precise extent to which these policies affect performance remains an open question. This gap in the literature highlights the need for further research that quantifies the specific impact of different endorsement policies on system latency and throughput. We conduct qualitative and quantitative analyses on the impact of endorsement policies on Fabric's performance. The detailed results of the quantitative analysis are provided in Section 7. Subsequently, we will qualitatively analyze the three endorsement policies using probabilistic distribution models. Assuming there are n endorsement peers, each processing transaction with an average time modeled as an independent and identically distributed random variable T, following a Gaussian distribution with mean μ and variance σ^2. Additionally, we assume that no endorsement peers experience crashes or network interruptions. In the AND policy, all n peers must endorse the transaction. Therefore, the endorsement time T_{AND} is the maximum of the n endorsement times T:

$$T_{\text{AND}} = \max(T_1, T_2, \ldots, T_n) \tag{1}$$

In the OR policy, any peer endorsement is sufficient. Therefore, the endorsement time T_{OR} is the minimum of the n endorsement times T:

$$T_{\text{OR}} = \min(T_1, T_2, \ldots, T_n) \tag{2}$$

In the $K\text{-}OutOf$ policy, the transaction endorsement time is defined as the time required for at least K out of all endorsement peers to complete the endorsement. Specifically, this involves sorting the endorsement times of all peers and selecting the $K\text{-}th$ time as the final transaction endorsement time. This time represents the minimum duration necessary to meet the policy's requirement, ensuring that at least K peers have endorsed the transaction.

The aforementioned three endorsement policies represent a horizontal expansion of the analysis. For the vertical expansion, involving the nested use of endorsement policies, we will further analyze its impact on Fabric's performance.

Assume there is a nested endorsement policy with a depth of d, where the endorsement time at each level is denoted as τ_1, τ_2, ..., τ_d. The endorsement time at each level is calculated as follows:

$$\tau_{\text{total}} = \tau_1 + \tau_2 + \cdots + \tau_d \tag{3}$$

The endorsement time at each level is determined by the endorsement policy applied at that specific level.

6.2. Dynamic Changes in Nodes

In the Hyperledger Fabric (HLF) permissioned chain, all nodes (a.k.a., peers) joining the blockchain must be authorized and declared through a configuration file during network initialization, similar to joining an allowlist. As a result, adding new peers requires restarting and redeploying the network, which limits the scalability of peers in Hyperledger Fabric (HLF). In an open environment, nodes may be temporarily offline due to external factors, such as downtime or network interruption. When these nodes are restarted or recovered, the performance of HLF may be affected.

Currently, there is no literature thoroughly investigating the impact of dynamic node changes on fabric performance. To fill this research gap, we design experiments to analyze the specific impact of dynamic node changes (such as offline and rejoined) on system performance.

In Section 6.1, although the previous analysis assumes that nodes do not fail, the reality in Hyperledger Fabric's production environment is that it operates in an open system, making it vulnerable to node failures. The Raft consensus mechanism ensures transaction consistency, yet different endorsement strategies have significantly different impacts on performance. For instance, the *AND* endorsement strategy may lead to endorsement failures in the event of node failures, whereas the *OR* endorsement strategy is more tolerant to such failures. Therefore, this paper explores the impact of node failures on different endorsement strategies under the Raft mechanism, with a detailed quantitative analysis presented in Section 7.

7. Evaluation

Implementation. We implement a BlockLoader framework based on Caliper V0.4 ([3]) with Node.js v10.x. To accurately simulate the temporal characteristics of the workload, we design specific workload patterns based on our study's requirements and integrate these patterns into the Rate Controllers module. We utilize Prometheus ([32]) to fetch test metrics from the Caliper Adaptor and store all collected data locally. Prometheus executes custom rules to aggregate new time series. Subsequently, we employ Grafana ([33]) to visualize these data, providing an intuitive graphical interface to display and analyze the time series data, effectively monitoring and assessing system performance.

Experiment Setup. We deploy 9 nodes on Aliyun ([34]) in the Zhangjiakou data centre, Hebei. Each node is an ecs.c5.xlarge instance with an Intel(R) Xeon(R) Platinum 3.1 GHz 4-core processor, 8 GB RAM, and 100 Mbps bandwidth. In the HLF network, 4 nodes are orderers, 1 node serves as the Caliper client, and the remaining 4 nodes operate as peers.

Workload. We utilize the popular benchmark smart contract, SmallBank, to simulate basic banking transfer operations. We configure the SmallBank benchmark with 100,000 accounts, and the access pattern to these accounts follows a uniform distribution.

Metrics. We focus on transaction throughput and latency using the definition provided in Hyperledger Fabric's white paper. Additionally, we monitor resource utilization, such as CPU and memory usage. Note that we will explain the definitions for throughput and latency below.

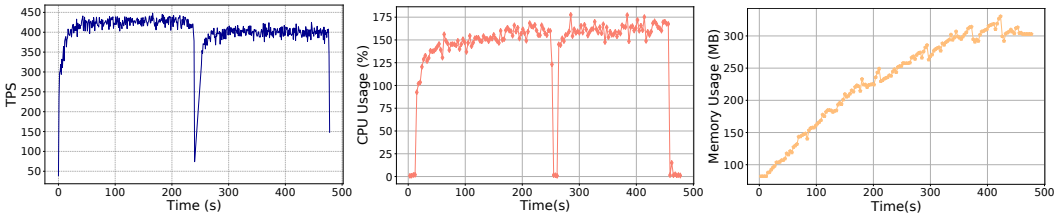
Throughput: The throughput is defined as the number of transactions successfully committed to the blockchain per second.

Latency: Latency is the difference between when a transaction is successfully committed to the blockchain and when it is initially submitted.

7.1. Performance Evaluation of HLF Under Different Workload Patterns

Baseline. Figure 8 shows the evaluation results of HLF in a uniform workload pattern, including throughput, CPU usage, and memory usage. The entire evaluation process is divided into two rounds. As shown in Figure 8a, the distribution of the two test rounds is

quite similar, and the same trend is also observed in Figure 8b. In this study, we chose the uniform workload mode as the baseline.

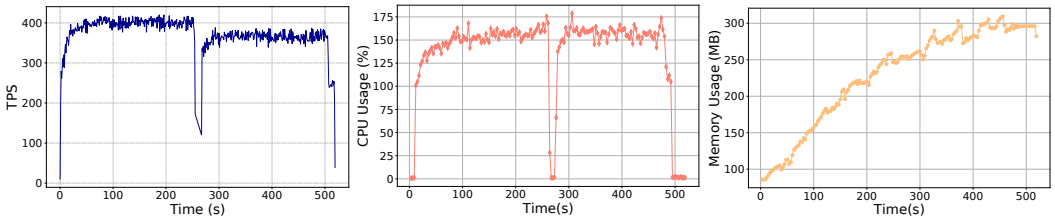


(**a**) Throughput (**b**) CPU (**c**) Memory

Figure 8. Performance and resource utilization under the uniform workload pattern.

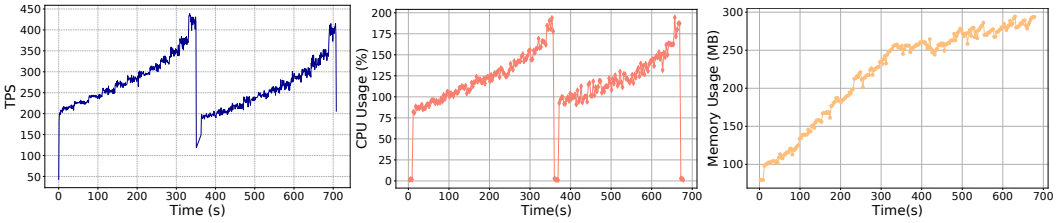
We compare the performance and resource utilization of Hyperledger Fabric under different workload patterns using the SmallBank workload. In our evaluation, all experiments are conducted more than three times in an Alibaba Cloud environment. In this section, we use Blockloader to evaluate the performance of HLF in six different workload patterns. We find that the trend in CPU usage highly correlates with transaction throughput, while memory usage exhibits a continuous growth trend. The increase in CPU usage is due to the increase in the number of transactions processed per unit time, which leads to an increase in computational overhead in the simulation execution, consensus, and validation phases. The continuous increase in memory usage is because, during the entire test process, the memory is not reclaimed immediately after each round but only released when the entire test is complete. We observe an interesting phenomenon. The throughput is significantly lower at the beginning of each test round than in the following stable running phase. This phenomenon can be attributed to the cold start problem, where the system is not fully loaded or initialized at the start of the test, resulting in a temporary reduction in throughput. As the system becomes stable, the necessary resources and caches are ready, and the throughput rapidly rises to an average level.

Figure 8 shows that although the fixed sending rate is set to 1000 TPS in the uniform workload pattern, the peak throughput in the testing is only 448 TPS. This indicates that when using the uniform workload pattern, the preset sending rate must exceed the system's peak throughput to evaluate its performance effectively. From Figure 8b, we observe that the CPU utilization of the blockchain node reaches a maximum of 194.36%, exceeding 100%. This is because the server used in the test is a quad-core architecture, and the CPU usage of each core is up to 100%. Therefore, the total CPU usage can reach 400% when multi-core parallel processing is performed. The figure shows that the current CPU usage does not reach its maximum value of 400%, which shows that the CPU computing resources do not become the performance bottleneck of the system during the test. This means that system performance limitations may arise from other factors. Figure 9 illustrates the system performance and resource utilization of Hyperledger Fabric (HLF) under the Dynamic Workload Pattern. In this pattern, users only need to set the initial value and step size, after which the system automatically adjusts the load to explore the maximum throughput. This pattern is different from the Uniform pattern in that there is no requirement to manually set a high sending rate and avoids excessive load pressure on the client. Figure 10 illustrates the system performance and resource utilization under the Linear Workload Pattern. In this pattern, the initial sending rate is set to 250 TPS, and the terminal rate is 1000 TPS. Figure 10a shows that when the system throughput reaches the peak, it will tend to converge near the peak. It is worth noting that the Linear Workload Pattern can be configured for monotonic increase and monotonic decrease. Figure 11 demonstrates the performance of the single-peak pattern. The single-peak trend can be clearly seen in Figure 11b, but it is difficult to clearly distinguish in Figure 11a because the throughput fluctuates wildly. This fluctuation occurs because the single-peak pattern comprises multiple stages, each with a different number of transactions. If the stage size is too large, the system faces a cold start issue each time it switches to a new stage, resulting in large fluctuations in throughput.



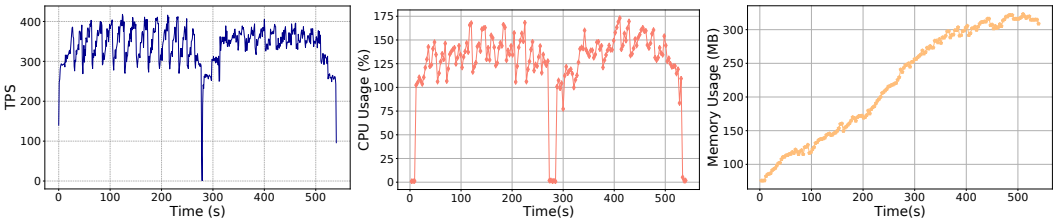
(**a**) Throughput (**b**) CPU (**c**) Memory

Figure 9. Performance and resource utilization under the dynamic workload pattern.



(**a**) Throughput (**b**) CPU (**c**) Memory

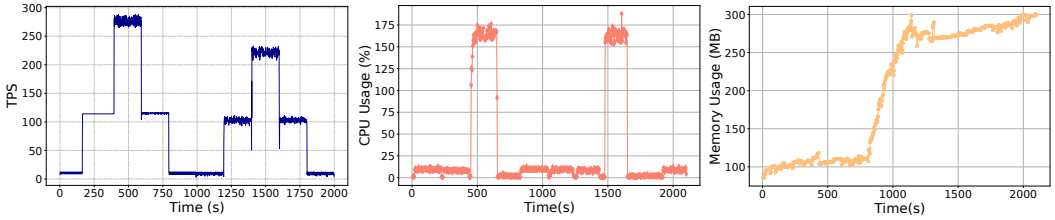
Figure 10. Performance and resource utilization under the linear workload pattern.



(**a**) Throughput (**b**) CPU (**c**) Memory

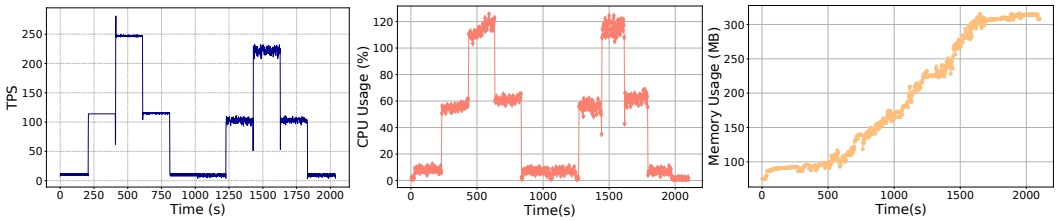
Figure 11. Performance and resource utilization under the single-peak workload pattern.

By observing Figures 12–14, we find an interesting phenomenon, that is, the throughput in certain stages remains consistent. After analysis, we find that this is related to implementing the workload distribution pattern. BlockLoader segments these distributions based on time steps, applying a fixed-rate pattern within each segmented time step. In some stages with a low fixed rate, the BlockLoader can quickly meet the requirements of the sending rate, so the throughput in these stages is consistent. Additionally, Table 1 shows latency statistics under different workload patterns, including maximum, minimum and average latency. A notable observation is that the minimum latency for all workload patterns is consistently 0.04 s. However, it is essential to note that while the minimum latency remains the same, there are significant variations in the maximum and average latency across different workload patterns. For instance, the Burst pattern exhibits a maximum latency of 2.09 s, indicating a substantial increase in response time under peak workload conditions. In contrast, the Dynamic pattern has a much lower maximum latency of 0.31 s, suggesting more stable performance under dynamic workloads.



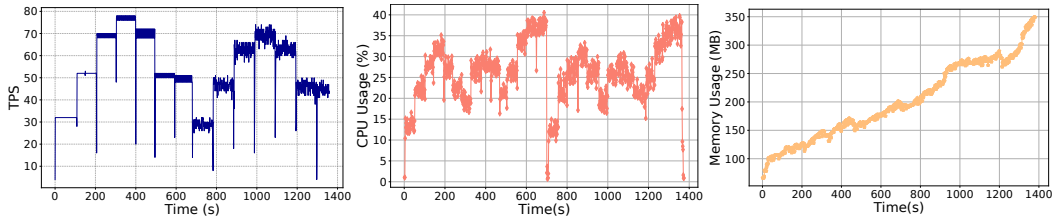
(**a**) Throughput (**b**) CPU (**c**) Memory

Figure 12. Performance and resource utilization under the burst workload pattern.



(**a**) Throughput (**b**) CPU (**c**) Memory

Figure 13. Performance and resource utilization under the random workload pattern.



(**a**) Throughput (**b**) CPU (**c**) Memory

Figure 14. Performance and resource utilization under the multi-peak workload pattern.

Table 1. Latency comparison across different patterns.

Workload Distribution Patterns	Latency (s)		
	Max	**Min**	**Avg**
Uniform (Baseline)	0.45	0.04	0.14
Burst	2.09	0.04	0.41
Linear	0.53	0.04	0.09
Random	2.05	0.04	0.21
Dynamic	0.31	0.04	0.09
Single-Peak	2.08	0.04	0.09
Multi-Peak	2.04	0.04	0.11

7.2. Comparing the Performance Under Different Endorsement Policies

In this section, we examine the performance impact of horizontal and vertical scaling under different endorsement policies. In order to eliminate the delay caused by geographical distribution across nodes, all experiments are carried out on one machine. We investigate the AND, OR and K-OutOf endorsement policies across different organizations. As shown in Figure 15, the OR endorsement policy exhibits minimal performance variation with changes in system scale, maintaining a steady throughput of approximately 86 to 90 TPS. As the number of organizations increases, the throughput of the AND endorsement policy exhibits a downward trend. When five organizations collaborate, the throughput drops to 47.02 TPS, approximately half of the throughput of a single organization. Notably, when the number of organizations is two, the performance reduction is most pronounced, with a decrease of about 20%. In contrast, the K-OutOf endorsement policy shows varying performance depending on the value of K among five organizations. When K = 1, the policy effectively becomes an OR endorsement policy; conversely, when K = 5, it reverts to an AND endorsement policy. For K = 2, K = 3, and K = 4, the throughput exceeds that of the AND endorsement policy. This is because the K-OutOf strategy requires only a certain number of endorsements to satisfy the endorsement condition without waiting for responses from specific organizations. In general, in the horizontal expansion of endorsement policies, the AND and K-OutOf policies significantly impact performance. In contrast, the OR policy has an almost negligible impact on performance.

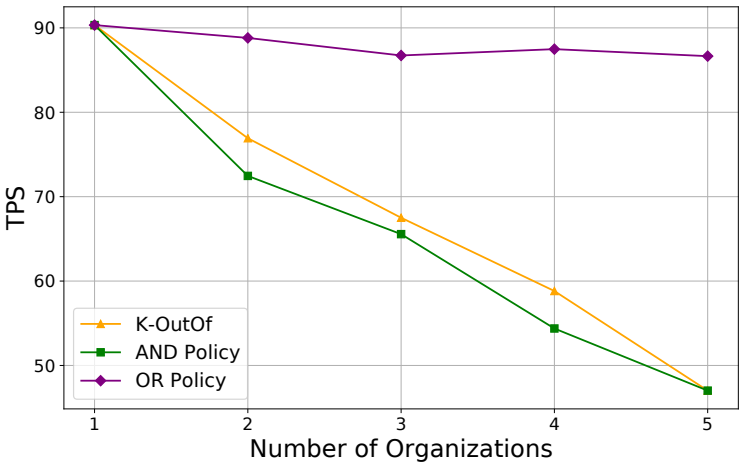


Figure 15. Performance impact of horizontal scaling in endorsement policies.

We model the endorsement strategies, where OR operations are sibling nodes and AND operations are child nodes. As shown in Figure 16, the modeling representations of the three endorsement policies. Figure 16a represents the $OR\ Org1$ endorsement policy, with Org1 as the tree's root node, resulting in a tree depth of 1. Figure 16b represents the policy $Org1\ AND\ (Org2\ OR\ Org3)$, and the depth of the tree is 2. Figure 16c illustrates a strategy with a depth of 3. We evaluate the impact of the depth of endorsement policies on performance. The experimental results demonstrate that the throughput is 90.33 TPS at a depth of 1. At a depth of 2, the throughput decreases to 79.83 TPS, and at a depth of 3, it further declines to 57.58 TPS. These findings suggest that the depth of endorsement policies within our tree structure significantly affects performance.

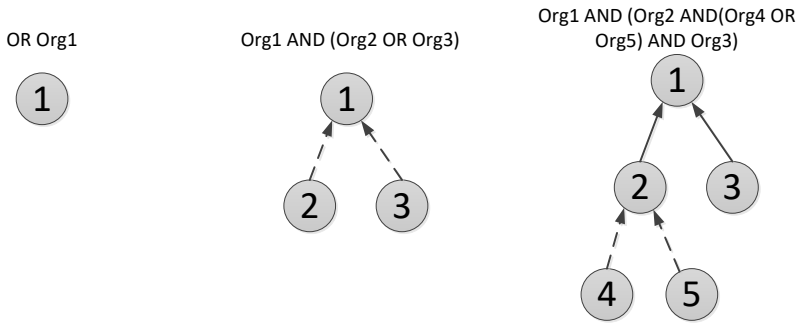


Figure 16. Hierarchical Representation of Endorsement Policies in Blockchain Systems. This figure depicts three different endorsement policies modeled by a binary tree, where the solid arrows represent the AND endorsement policy and the dotted arrows represent the OR endorsement policy.

7.3. Analyzing the Impact of Node Dynamics on Hyperledger Fabric Performance

In this section, we deploy a Hyperledger Fabric (HLF) network on a personal computer. The network consists of two organizations and four nodes. Specifically, one organization consists of three nodes, while the other organization consists of only one node. BlockLoader is bound to the peer0 node of Organization 1 and sends 100,000 transactions at a fixed rate of 1000 transactions per second (TPS). The experiments employ two different endorsement policies: OR and AND. In this experiment, we examine the effects of node dynamics by simulating node failures and recoveries. Specifically, we intentionally shut down nodes for durations of 100, 200, 300, 400, and 500 s. As illustrated in Figure 17, HLF maintains a throughput of approximately 87 TPS under the OR endorsement policy. As the duration of node failures or node leaves increases to 100, 200, 300, 400, and 500 s, the throughput

remains stable, exhibiting no significant variation. These findings indicate that node dynamics do not affect the throughput of the HLF under the OR endorsement policy.

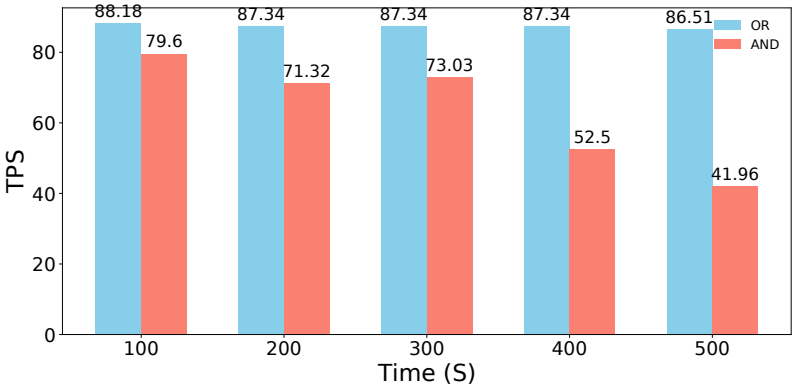


Figure 17. Performance of HLF node dynamics under different endorsement policies.

Additionally, we conduct evaluation experiments under the AND endorsement policy, primarily simulating node failures or leaves within Organization 2. As the duration of node failures or node leaves increases to 100, 200, 300, 400, and 500 s, the throughput of Hyperledger Fabric (HLF) significantly decreases. When the node failure or leave duration reaches 500 s, the throughput decreases to 41.96 TPS, equivalent to half of Hyperledger Fabric (HLF) performance under normal node conditions. We observe an interesting phenomenon: the throughput at 300 s of node failure or leaving is higher than 200 s. A deeper analysis of the transaction logs shows a higher number of aborted transactions at the 300 s mark. These aborted transactions do not execute the consensus process, thereby reducing the overall test time. The reason behind this may be that when a node fails or leaves, some transactions are aborted, reducing the number of transactions that need to reach consensus, thereby shortening the overall execution time. An appropriate number of aborted transactions can reduce the system load and allow valid transactions to be completed more effectively, thereby increasing throughput. However, when the number of aborted transactions exceeds a certain threshold, the reduction in valid transactions will be too large, resulting in the system being unable to fully utilize resources, ultimately leading to a decrease in throughput. In summary, node dynamics have varying impacts under different endorsement policies.

8. Conclusions

In this study, we introduce BlockLoader, a novel framework designed to simulate varying load distributions, including linear, single-peak, and multi-peak patterns, to evaluate the performance of blockchain systems. Using BlockLoader, we conduct a comprehensive analysis of the impact of endorsement policy breadth and depth on the performance of Hyperledger Fabric. Our results reveal that endorsement policies significantly influence the scalability and performance dynamics of the system. Specifically, while the OR endorsement policy maintains stable performance, averaging around 88 TPS even with the addition of organizations and nodes, the AND endorsement policy demonstrates an apparent decline in performance, with an average decrease of 10 TPS per additional organization. Furthermore, the impact of dynamic node changes is markedly different across the two endorsement policies. The AND policy is particularly sensitive to such changes, as evidenced by a TPS reduction from 79.6 at 100 s to 41.96 at 500 s, representing a 47% decrease over time. In contrast, the OR policy resists dynamic node variations with minimal impact on system performance. These insights facilitate blockchain developers in optimizing platform selection and formulating strategic endorsement policies.

Author Contributions: Conceptualization, X.L.; Data curation, Y.Z.; Investigation, G.W.; Methodology, Y.Z.; Software, G.W.; Supervision, G.Y.; Validation, C.Y.; Visualization, G.W. and Z.P.; Writing—original draft, G.W. and Q.Z.; Writing—review and editing, C.Y. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by the National Natural Science Foundation of China (NSFC) (Grant numbers 62141605, 62402313, and 62372097), the Beijing Natural Science Foundation (Grant Z230001), and the Special Funds for Basic Scientific Research of Central Universities (Grant numbers N2416003).

Data Availability Statement: The original contributions presented in the study are included in the article, further inquiries can be directed to the corresponding author.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Wang, C.; Chu, X. Performance characterization and bottleneck analysis of hyperledger fabric. In Proceedings of the 2020 IEEE 40th International Conference on Distributed Computing Systems (ICDCS), Singapore, 29 November–1 December 2020; pp. 1281–1286.
2. Fabric, H. *Hyperledger Fabric Documentation*; The Linux Foundation: San Francisco, LA, USA, 2023.
3. Huawei. Hyperledger Caliper. 2017. Available online: https://www.hyperledger.org/projects/caliper (accessed on 2 August 2023).
4. Dinh, T.T.A.; Wang, J.; Chen, G.; Liu, R.; Ooi, B.C.; Tan, K.L. Blockbench: A framework for analyzing private blockchains. In Proceedings of the 2017 ACM International Conference on Management of Data, Chicago, IL, USA, 14–19 May 2017; pp. 1085–1100.
5. Wang, G.; Zhang, Y.; Ying, C.; Li, X.; Yu, G. Hammer: A General Blockchain Evaluation Framework. In Proceedings of the 44th IEEE International Conference on Distributed Computing Systems, ICDCS 2024, Jersey City, NJ, USA, 23–26 July 2024; pp. 391–402.
6. Saingre, D.; Ledoux, T.; Menaud, J.M. BCTMark: A framework for benchmarking blockchain technologies. In Proceedings of the 2020 IEEE/ACS 17th International Conference on Computer Systems and Applications (AICCSA), Antalya, Turkey, 2–5 November 2020; pp. 1–8.
7. Nasrulin, B.; De Vos, M.; Ishmaev, G.; Pouwelse, J. Gromit: Benchmarking the performance and scalability of blockchain systems. In Proceedings of the 2022 IEEE International Conference on Decentralized Applications and Infrastructures (DAPPS), Newark, CA, USA, 15–18 August 2022; pp. 56–63.
8. Sedlmeir, J.; Ross, P.; Luckow, A.; Lockl, J.; Miehle, D.; Fridgen, G. The DLPS: A new framework for benchmarking blockchains. In Proceedings of the 54th Hawaii International Conference on System Sciences, Kauai, HI, USA, 5–8 January 2021.
9. Chacko, J.A.; Mayer, R.; Jacobsen, H.A. Why do my blockchain transactions fail? A study of hyperledger fabric. In Proceedings of the 2021 International Conference on Management of Data, Xi'an, China, 20–25 June 2021; pp. 221–234.
10. Enare Abang, J.; Takruri, H.; Al-Zaidi, R.; Al-Khalidi, M. Latency performance modelling in hyperledger fabric blockchain: Challenges and directions with an IoT perspective. *Internet Things* **2024**, *26*, 101217. [CrossRef]
11. Piao, X.; Ding, H.; Song, H. Performance Analysis of Endorsement in Hyperledger Fabric Concerning Endorsement Policies. *Electronics* **2023**, *12*, 4322. [CrossRef]
12. Melo, C.; Gonçalves, G.; Silva, F.A.; Soares, A. A comprehensive hyperledger fabric performance evaluation based on resources capacity planning. *Clust. Comput.* **2024**, *27*, 12395–12410. [CrossRef]
13. Androulaki, E.; Barger, A.; Bortnikov, V.; Cachin, C.; Christidis, K.; De Caro, A.; Enyeart, D.; Ferris, C.; Laventman, G.; Manevich, Y.; et al. Hyperledger fabric: A distributed operating system for permissioned blockchains. In Proceedings of the Thirteenth EuroSys Conference, Porto, Portugal, 23–26 April 2018; pp. 1–15.
14. Kreps, J.; Narkhede, N.; Rao, J. Kafka: A distributed messaging system for log processing. In Proceedings of the NetDB, Athens, Greece, 12–16 June 2011; pp. 1–7.
15. Castro, M.; Liskov, B. Practical Byzantine fault tolerance. In Proceedings of the Third Symposium on Operating Systems Design and Implementation (OSDI), New Orleans, LA, USA, 22–25 February 1999; pp. 173–186.
16. Dean, J.; Ghemawat, S. LevelDB. 2020. Available online: https://github.com/google/leveldb (accessed on 24 February 2021).
17. Apache CouchDB. CouchDB. 2020. Available online: https://couchdb.apache.org/ (accessed on 24 February 2021).
18. Fan, C.; Ghaemi, S.; Khazaei, H.; Musilek, P. Performance evaluation of blockchain systems: A systematic survey. *IEEE Access* **2020**, *8*, 126927–126950. [CrossRef]
19. Shah, J.; Sharma, D. Performance Benchmarking Frameworks for Distributed Ledger Technologies. In Proceedings of the 2021 IEEE International Conference on Electronics, Computing and Communication Technologies (CONECCT), Bangalore, India, 9–11 July 2021; pp. 1–5.
20. Technologies, H.Q. HyperBench: Blockchain Performance Benchmarking Tool. Available online: https://github.com/meshplus/hyperbench (accessed on 19 October 2024).
21. Gramoli, V.; Guerraoui, R.; Lebedev, A.; Natoli, C.; Voron, G. *Diablo-v2: A Benchmark for Blockchain Systems*; Technical Report; EPFL: Lausanne, Switzerland, 2022.

22. Dong, Z.; Zheng, E.; Choon, Y.; Zomaya, A.Y. Dagbench: A performance evaluation framework for dag distributed ledgers. In Proceedings of the 2019 IEEE 12th International Conference on Cloud Computing (CLOUD), Milan, Italy, 8–13 July 2019; pp. 264–271.
23. Krüger, A. Chainhammer: Ethereum Benchmarking. 2017. Available online: https://github.com/drandreaskrueger/chainhammer (accessed on 4 September 2024).
24. ConsenSys. Quorum Profiling: Performance Analysis of Quorum. Available online: https://github.com/ConsenSys/quorum-profiling (accessed on 4 September 2024).
25. Birim, M.; Ari, H.E.; Karaarslan, E. GoHammer Blockchain Performance Test Tool. *J. Emerg. Comput. Technol.* **2021**, *1*, 31–33.
26. Kuzlu, M.; Pipattanasomporn, M.; Gurses, L.; Rahman, S. Performance analysis of a hyperledger fabric blockchain framework: Throughput, latency and scalability. In Proceedings of the 2019 IEEE International Conference on Blockchain (Blockchain), Atlanta, GA, USA, 14–17 July 2019; pp. 536–540.
27. Kim, J.W.; Song, J.G.; Lee, T.R.; Jang, J.W. Performance evaluation of NFT trading platform based on hyperledger fabric blockchain. In Proceedings of the 2022 8th International Conference on Computing and Data Engineering, Bangkok, Thailand, 11–13 January 2022; pp. 65–70.
28. Harris, C. Performance Evaluation of Ordering Services and Endorsement Policies in Hyperledger Fabric. In Proceedings of the 2023 33rd Conference of Open Innovations Association (FRUCT), Zilina, Slovakia, 24–26 May 2023; pp. 63–69.
29. Al-Sumaidaee, G.; Alkhudary, R.; Zilic, Z.; Swidan, A. Performance analysis of a private blockchain network built on Hyperledger Fabric for healthcare. *Inf. Process. Manag.* **2023**, *60*, 103160. [CrossRef]
30. Ke, Z.; Park, N. Performance modeling and analysis of Hyperledger Fabric. *Clust. Comput.* **2023**, *26*, 2681–2699. [CrossRef]
31. Stoltidis, A.; Choumas, K.; Korakis, T. Performance Optimization of High-Conflict Transactions within the Hyperledger Fabric Blockchain. *arXiv* **2024**, arXiv:2407.19732.
32. Volz, J.; Brian B.; Conor, B.; Matt, L.; Steve, D. Prometheus: Monitoring System and Time Series Database. 2012. Available online: https://prometheus.io/ (accessed on 6 June 2024).
33. Ödegaard, T. Grafana: The Open Platform for Analytics and Monitoring. Available online: https://grafana.com/ (accessed on 6 June 2024).
34. Alibaba Cloud. Cloud Computing Services. 2024. Available online: https://www.alibabacloud.com (accessed on 1 October 2024).

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.