Journal of Internet Services and Applications, 2026, 17:1, doi: 10.5753/jisa.2026.6262

Performance Analysis and Metric Correlation in Blockchain Networks: Geth vs. Besu with and without Load Balancer

Isaac de Abreu Gaspar [**Universidade Federal Fluminense** | ***iagaspar@id.uff.br***]
Antonio A. de A. Rocha [**Universidade Federal Fluminense** | ***arocha@ic.uff.br***]

Institute of Computing, Universidade Federal Fluminense, Av. Gal. Milton Tavares de Souza, s/n, São Domingos, Niterói, RJ, 24210-590, Brazil.

Received: 25 July 2025 • **Accepted:** 27 November 2025 • **Published:** 30 March 2026

Abstract

Blockchain technology has emerged as a disruptive innovation, facilitating decentralized and secure transactions across various domains. However, performance and scalability are critical challenges, especially for enterprise applications. This study evaluates the performance of two Ethereum-based blockchain clients: Geth (Go Ethereum) and Hyperledger Besu, under different network conditions, including the use of a Load Balancer. We conducted an extensive benchmarking analysis using the Hyperledger Caliper tool to measure key performance metrics such as Requests per Second (Req/s), Transactions per Second (TPS), Latency, and Resource Utilization (CPU and Memory). Additionally, we performed statistical analyses, including Percentage Difference, Coefficient of Variation, and Correlation between Metrics, to provide deeper insights. Our results indicate that Geth consistently outperforms Besu in terms of TPS and latency under high workloads, while Besu demonstrates greater stability and scalability as the number of nodes increases. The impact of the Load Balancer varies across metrics, improving throughput in some cases but increasing latency in others. These findings offer valuable insights for organizations selecting blockchain platforms for enterprise solutions, highlighting scenarios in which each client performs best.

Keywords: Blockchain Performance, Ethereum, Geth, Hyperledger Besu, Load Balancer, Benchmarking, Scalability.

1 Introduction

Blockchain technology has consolidated itself as a promising tool for building decentralized, secure, and transparent systems. Permissioned platforms such as Geth (Go Ethereum) and Hyperledger Besu have been explored in different corporate and financial scenarios, mainly due to the need to ensure transaction integrity, auditability, and reliability [Hafza, 2025]. However, despite their theoretical advantages, the practical adoption of these platforms is still limited by challenges related to performance and scalability, which are critical factors for real-world applications [Yli-Huumo *et al*., 2016].

The growing interest in blockchain for enterprise and market applications highlights the need to understand not only the technology itself but also its performance, scalability, and reliability in realistic deployment scenarios. Permissioned blockchain networks hold great potential to support secure transaction systems but still face significant challenges related to efficiency, latency, and processing capacity under different workloads and node configurations [Adulla *et al*., 2018].

In this work, we focus on private Ethereum networks that employ the Proof-of-Authority (PoA) consensus protocol, rather than on public Ethereum mainnet configurations based on Proof-of-Work (PoW) or Proof-of-Stake (PoS). PoA was originally proposed for private and consortium Ethereum deployments and is widely used in industrial and public-sector applications [Samuel *et al*., 2021; Islam *et al*., 2022]. Examples include the Energy Web Chain, which leverages PoA-based Ethereum technology for energy-sector use cases [Energy Web Foundation, 2023], and the Ethereum Proof-of-Authority consortium templates offered on Microsoft Azure for enterprise blockchain consortia [Microsoft, 2020]. These deployments, together with several governmental pilots based on permissioned Ethereum networks reported in the literature, illustrate that PoA is a practical choice when participants are known and authorized and when predictable performance and governance are prioritized over open participation [Yli-Huumo *et al*., 2016; Adulla *et al*., 2018]

In this context, the comparative evaluation of the performance of permissioned Ethereum platforms becomes essential. Performance analysis not only allows for the identification of bottlenecks and limitations of each client, but also provides a foundation for developing strategies to optimize transaction execution and the utilization of computational resources [Fan *et al*., 2022].

The core focus of this work is the investigation of the impact of load balancing in private Ethereum PoA networks. Different scalability limitations suggest that the implementation of load-balancing middleware can mitigate bottlenecks, improve transaction processing efficiency, and increase network resilience [Fan *et al*., 2022]. The experiments conducted in this study assess how load distribution affects performance metrics such as request rate (Req/s), latency, throughput (TPS), and completion rate, in addition to its impact on CPU and memory consumption.

To investigate these limitations, we propose a comparative study of the performance of Geth and Besu networks. In this study, we focus on two major Ethereum clients: Geth and Hyperledger Besu. This choice is motivated by their practical relevance and representativeness within the Ethereum ecosystem. Geth serves as the reference and most widely adopted

client in both public and private deployments, while Besu, maintained under the Hyperledger Foundation, is widely used in enterprise and permissioned blockchain environments. Although other implementations such as Nethermind, Erigon, and Reth are available, their inclusion would broaden the scope of this work beyond its main objective — to provide a comparative and practical evaluation of load balancing effects on two representative Ethereum clients with distinct architectural and operational profiles.

To guide the investigation, we formulated the following main research question:

- RQ. How do different configurations of permissioned Ethereum PoA networks affect their performance and scalability, and how can strategies such as load balancers mitigate performance bottlenecks?

This main question is subdivided into complementary sub-research questions:

- SRQ1. How do Geth and Besu differ in performance under different workload scenarios and node configurations?
- SRQ2. How does the application of a middleware load balancer influence the performance, scalability, and resource utilization of Geth and Besu networks?
- SRQ3. What performance relationships emerge among TPS, latency, and Req/s, and how do these correlations support the interpretation of results in SRQ1 and SRQ2?

To answer these questions and identify performance bottlenecks, we conducted extensive performance testing on both networks using the Hyperledger Caliper benchmarking tool [Hyperledger, 2025]. We evaluated performance metrics such as request rate (Req/s), average latency, throughput (TPS), and completion rate, along with additional metrics including percentage differences, coefficients of variation, and correlations between metrics. We also evaluated performance in terms of computational resources, monitoring CPU and memory usage during the tests. All experiments were performed both with and without an Nginx-based load balancer in private PoA networks using the Clique consensus protocol and a fixed 1-second block time.

This paper makes the following contributions:

- Experimental framework for private Ethereum PoA networks. We design and implement a private Ethereum testbed based on the Clique PoA consensus, using Geth and Besu clients under multiple node configurations (2–10 nodes) and realistic smart contract workloads derived from a decentralized marketplace prototype.
- Systematic evaluation of Nginx load balancing. We evaluate the impact of introducing an Nginx L7 load balancer on key performance indicators (Req/s, TPS, latency, completion rate) and on CPU and memory usage, under a wide range of transaction volumes and network sizes.
- Comparative analysis of Geth vs. Besu under PoA. We compare Geth and Besu in identical PoA settings, showing that Geth achieves higher throughput and lower latency under high loads, while Besu benefits more from load balancing in terms of latency control but at a higher CPU cost.
- Statistical analysis and metric correlations. We compute percentage differences, coefficients of variation, and Pearson correlations between metrics to quantify stability and linearity. Our results show that Geth exhibits an almost linear Req/s–TPS relationship (correlations above 0.96) even without load balancing, whereas Besu requires a load balancer to achieve similar predictability.
- Practical guidelines for enterprise PoA deployments. From the experiments, we derive concrete guidelines for architects of private Ethereum-based systems, indicating when and how to employ Nginx load balancing, and under which conditions Geth or Besu is more appropriate (e.g., Geth for resource-constrained high-throughput scenarios; Besu for latency-sensitive deployments with ample CPU capacity).

The lessons learned from these experiments indicate that: (i) naive single-endpoint configurations tend to overload a primary node and degrade latency and completion rate; (ii) introducing an Nginx load balancer reduces latency and stabilizes throughput, but at the cost of increased CPU usage on backend nodes; and (iii) architectural choices such as client selection and number of validators significantly affect how effectively a PoA network converts additional request load (Req/s) into committed transactions (TPS).

This article is organized as follows: Section 2 contextualizes the topics discussed in this work. Section 3 outlines the methodology used for conducting the experiments. Section 4 presents the results obtained from those experiments. Finally, Section 5 concludes the work.

2 Background

2.1 Ethereum Fundamentals and Ethereum Clients

Ethereum is one of the leading blockchain platforms for developing enterprise applications. In an Ethereum network, nodes are referred to as clients, which are responsible for executing transactions and verifying and creating blocks. All nodes in the network maintain a state machine known as the Ethereum Virtual Machine (EVM), which executes transactions and smart contracts while calculating the network's state in each block. A smart contract consists of code that automatically applies predefined rules and executes as programmed [Fan *et al*., 2022].

Ethereum clients are powerful software applications that connect users to the blockchain. By implementing the Ethereum protocol, these clients become nodes that help maintain the decentralization and security of the network. Notable examples include the Geth (Go Ethereum) and Hyperledger Besu platforms. These platforms enable users to run an Ethereum node, validate transactions, deploy smart contracts, and interact with the blockchain [Hafza, 2025].

The Geth platform is designed to be efficient and robust, taking advantage of the strengths of the Go programming language to provide a reliable client for interacting with the Ethereum network. Geth maintains a copy of all historical

transaction and smart contract data, ensuring synchronization with the rest of the network. Additionally, Geth verifies the validity of transactions, checking that the sender has sufficient funds, that the transaction is correctly signed, and that it complies with the network's consensus rules. The platform also executes smart contracts using the EVM, allowing decentralized applications (dApps) to run securely and immutably on the blockchain [Hafza, 2025].

The Hyperledger Besu platform is an Ethereum client developed by Pegasys, a subsidiary of ConsenSys. Besu is designed for building and managing decentralized applications (dApps). The platform supports both public and private Ethereum networks and includes features for smart contract management, scalability, privacy protection, and integration with other blockchain networks. Additionally, Besu enables secure transactions and allows users to create their own dApps using its development toolkit. This toolkit facilitates the development of applications that can interact with smart contracts on the Ethereum network as well as other compatible blockchains, such as Bitcoin or Quorum. As a result, dApps built on Besu can operate across multiple platforms, providing greater access and flexibility for users [Carneiro, 2023].

The Geth platform employs the Proof-of-Authority (PoA) consensus protocol. Besu, in addition to PoA, also supports the IBFT 2.0 and QBFT consensus mechanisms. In a blockchain network, consensus defines the process by which participating nodes agree on the final state of the distributed ledger [Samuel *et al*., 2021].

The PoA protocol was originally proposed for private Ethereum networks and is mainly implemented in two variants: Aura and Clique. In this study, the Clique consensus was adopted because it is natively supported by both Geth and Besu, allowing a direct and unbiased comparison between the two platforms under identical consensus conditions [Islam *et al*., 2022].

PoA is a fault-tolerant algorithm designed to optimize the performance of private and consortium blockchains. In this model, a group of trusted validators, known as signers, is responsible for approving and appending new blocks to the chain. The protocol operates with N authorized nodes, each identified by a unique public key. To achieve consensus, at least $N/2 + 1$ signers must agree on the network state. Block creation occurs in a round-robin scheme at fixed intervals, with the responsibility for proposing new blocks rotating cyclically among the authorities [Islam *et al*., 2022].

The IBFT 2.0 (Istanbul Byzantine Fault Tolerance) and QBFT (Quorum Byzantine Fault Tolerance) mechanisms are based on Byzantine Fault Tolerance (BFT). Both ensure consensus even in the presence of malicious nodes through multiple rounds of voting among validators before a block is finalized. IBFT 2.0 is widely used in permissioned Ethereum networks, while QBFT, its successor, improves efficiency and stability by reducing communication complexity among nodes [Samuel *et al*., 2021].

2.2 Performance Evaluation and Benchmarking in Blockchain Networks

Performance evaluation is the process of measuring how well a System Under Test (SUT) performs. This evaluation can encompass broad system analyses, such as assessing latency and response time, as well as specific metrics, like the time taken to write a block of data to persistent memory. The main goal of this evaluation is to understand and document how the system or its subsystems behave under varying conditions. This often involves analyzing how independent variables impact system performance, such as the relationship between concurrent interference rates and processing speeds [Adulla *et al*., 2018].

Benchmarking, on the other hand, involves applying standardized methods to compare different systems or various versions of the same system over time. Performance benchmarks are a specific type of performance evaluation that carefully define the workloads used during testing and specify the metrics that will be collected. While benchmarking can be time-consuming and is typically conducted in controlled environments to ensure reproducibility, it allows for reliable comparisons between different configurations. Without standardized benchmarks, comparative analysis across different systems becomes less meaningful [Adulla *et al*., 2018].

2.3 Definition of Relevant Terms in Blockchain Network Performance Assessment

In order to establish a common understanding of the concepts employed in the benchmarking and evaluation process, this subsection summarizes the main terms used in the performance assessment of blockchain networks. The following definitions are consistent with prior work on blockchain benchmarking and performance evaluation [Kanai, 2022; Raab, 2019; Adulla *et al*., 2018; Becher, 2024; Chen *et al*., 2022; Fan *et al*., 2022]:

- **Test Harness.** Collection of hardware and software components used to conduct performance evaluations, including multiple clients that generate workloads and monitor the System Under Test (SUT) [Kanai, 2022]. In this work, Hyperledger Caliper plays the role of a test harness.
- **System Under Test (SUT).** Set of all components that compose the blockchain network under evaluation, such as hardware, software (blockchain clients), network infrastructure, and configuration parameters [Raab, 2019]. In our case, the SUT corresponds to the private Ethereum PoA networks built with Geth and Besu.
- **Client.** Entity that interacts with the SUT by submitting requests or monitoring its behavior. Two main types are typically distinguished [Adulla *et al*., 2018]: (i) *load-generating clients*, which send transactions using automated test scripts; and (ii) *observer clients*, which monitor and query the SUT but do not submit transactions.
- **Node.** Independent computational unit that participates in the blockchain network by processing transactions and contributing to network integrity. Nodes propagate transactions, maintain local copies of the ledger, generate and validate blocks, and may assume leader/validator roles depending on the consensus rules [Becher, 2024].
- **Transaction.** Operation that triggers a state change in the blockchain (for example, a financial transfer or the recording of IoT data). Transactions are initiated by clients,

validated by the SUT, and permanently stored on the ledger once the consensus rules are satisfied [Adulla *et al*., 2018].
- **Latency.** Time interval between transaction submission by a client and its confirmation on the blockchain. This metric captures propagation delays, queuing effects, and the time required for consensus settlement [Chen *et al*., 2022].
- **Throughput.** Rate at which valid transactions are confirmed by the blockchain, commonly expressed in transactions per second (TPS). Throughput is typically computed as the number of successfully committed transactions divided by the observation time window, discounting invalid or failed submissions [Chen *et al*., 2022].
- **Scalability.** Ability of the blockchain network to sustain higher workloads or a larger number of participants while keeping performance at acceptable levels. In performance studies, scalability is usually assessed by observing how throughput and latency evolve as the number of nodes or the transaction volume increases [Fan *et al*., 2022].

2.3.1 Nginx and Round-Robin Load Balancing

NGINX is one of the most widely adopted load-balancing solutions in modern distributed systems. Originally introduced as a high-performance web server, it evolved into a reverse proxy and load balancer, known for its scalability, low resource usage, and reliability [Elaurichenickson, 2024]. Its event-driven, asynchronous architecture enables handling thousands of concurrent connections efficiently, making it a common choice for fronting APIs and microservices in production environments.

In addition to static content delivery, NGINX is frequently deployed as an application-layer (Layer 7) load balancer. In this role, it receives client requests on a single endpoint and forwards them to a pool of backend servers according to a configurable policy. NGINX supports multiple load-balancing strategies, such as *round robin*, *least connections*, and *IP hash*, and provides advanced features including health checks, session persistence, SSL/TLS termination, and caching [Holcombe, 2018]. By enabling horizontal scalability and high availability, NGINX improves user-perceived latency and guarantees service continuity under failures or peak loads, making it a cornerstone for resilient large-scale applications [Alesh, 2023].

In the context of load balancing, the *round-robin* strategy is a simple, stateless algorithm that forwards each incoming request to the next backend server in a cyclic order. Unlike CPU scheduling, where round robin relies on fixed time quanta to share processor time among processes, here it is applied at the request-routing level: each individual request (for example, an HTTP or JSON-RPC call) is sent to a different backend node in turn. This tends to distribute the load evenly across all available servers, without requiring per-connection state or complex heuristics [Alesh, 2023].

Due to its simplicity, wide adoption, and neutrality with respect to client identity, round-robin load balancing is frequently used as a baseline strategy in empirical studies of distributed systems and middleware performance.

2.4 Related Work

Several studies have explored the application of load balancing techniques to mitigate bottlenecks and enhance the performance of blockchain networks, as the Table 1 shows.

Fan *et al*. [2022] conducted a comprehensive analysis of Hyperledger Besu in private networks, evaluating metrics such as throughput, latency, resource usage, and scalability under different parameters, including the use of load-balancing middleware. The authors identified that factors such as block time and the computational power of nodes have a decisive impact on performance, revealing bottlenecks related to transaction execution and state updates.

Tareen *et al*. [2023] proposed a load balancing scheme for healthcare systems in smart cities, aiming to reduce delays in the transmission of critical patient data. The method dynamically distributes mining tasks among computational nodes and prioritizes sensitive information, outperforming traditional algorithms such as round-robin in terms of efficiency and response time.

Li *et al*. [2023] introduced LB-Chain, a sharding system with dynamic load balancing among blockchain shards through the periodic migration of active accounts. Experiments demonstrated significant improvements in throughput (over 10%) and up to 90% reduction in transaction confirmation delays, as well as greater fairness in processing time across users.

In a similar line, Li *et al*. [2022] addressed the challenges of state sharding by proposing the Transformers protocol, which applies a community-aware account partitioning algorithm. Results showed substantial improvements in throughput, latency, and load distribution among shards, even under high rates of cross-shard transactions.

Harish and Sridevi [2024] investigated the impact of load balancing in split-join blockchains, which aim to achieve scalability through parallel block processing. Their empirical study demonstrated that load balancing is crucial to sustaining throughput and reducing processing time under increasing transaction loads.

Samuel *et al*. [2021] conducted a comprehensive performance evaluation of three major Ethereum clients — Geth, OpenEthereum (Parity), and Hyperledger Besu — operating under Proof of Authority consensus mechanisms, including Clique, Aura, and IBFT 2.0. The study introduced a novel testing methodology aimed at overcoming limitations of prior benchmarking approaches and implemented the experiments in a Microsoft Azure Cloud environment. To ensure fair transaction distribution, the authors proposed a dedicated load-balancer middleware specifically designed for Ethereum-based private networks. Their analysis identified key performance bottlenecks and client-specific issues, such as node stalling in Geth's EVM layer and fork-related deadlocks in Clique, highlighting critical aspects affecting reliability and efficiency in private blockchain deployments.

In contrast to these proposals, which focus on specific architectures (such as sharding, split-join, or sectoral applications) or isolated platforms like Besu, this work adopts a comparative and practical approach by evaluating the impact of a traditional load balancer on two major Ethereum implementations — Geth and Hyperledger Besu. Beyond analyzing

classical performance metrics such as throughput and latency, this study incorporates additional statistics (Percentage Difference, Coefficient of Variation, Correlations between Metrics) and measures computational resource consumption, providing concrete guidelines for the application of load balancers in private blockchain networks.

3 Methodology

This section details the methodology employed to evaluate the performance of the Geth and Besu blockchain networks under different load scenarios, with and without a Load Balancer (LB). We used Hyperledger Caliper as a benchmark tool to measure key performance metrics. The main objective of this study is to analyze the impact of LB on the networks, as well as to compare their operational efficiency.

3.1 Testbed Setup

The experiments were conducted in a controlled environment on an infrastructure composed of 10 Geth nodes and 10 Besu nodes, hosted on virtual machines (VMs) running on a server equipped with 40 Intel Xeon Silver 4114 2.20 GHz cores (distributed across two sockets) and 125.58 GB of RAM. Each VM was configured with Geth v1.13.13-stable and Hyperledger Besu v24.1.0, both installed on Ubuntu 22.04.4 LTS.

To simulate a realistic usage scenario, we deployed a smart contract derived from a decentralized marketplace prototype, the Ânima Marketplace. This prototype was designed to emulate operations typical of blockchain-based commercial environments, such as product registration, listing, and transactions between users. In this study, we employed the createProduct function of the contract, where each request sent by Caliper represented a new product creation transaction on the marketplace.

The createProduct function is responsible for registering new products in the Ânima Marketplace smart contract. It receives the product name and price as parameters, performs basic validation (ensuring that the name is not empty and the price is greater than zero), and then increments the product counter (productCount). After that, it creates a new Product object, associating it with the address of the user who initiated the transaction, and emits the ProductCreated event, which notifies the network of the new product creation.

This process represents a typical blockchain write operation, involving validation, state update, and event generation for traceability. The code snippet showing the createProduct function can be seen in Figure 1.

```
function createProduct(string memory _name, uint _price) public {
    require(bytes(_name).length > 0, "Invalid name");
    require(_price > 0, "Invalid price");
    productCount++;
    products[productCount] = Product(productCount, _name, _price,
    payable(msg.sender), false);
    emit ProductCreated(productCount, _name, _price,
    payable(msg.sender), false);
}
```

Figure 1. CreateProduct function of the Anima Marketplace contract

Figure 2 illustrates the two architectural scenarios considered in this study. In the non-load-balanced configuration (Figure 2(1)), the Hyperledger Caliper client sends all requests directly to Node 1. This node acts simultaneously as the main entry point for incoming transactions and as a full validator participating in the Clique PoA consensus, propagating the received transactions to the other validator nodes (Node 2, Node 3, …, Node 10). As a consequence, Node 1 tends to become a computational and networking hotspot, since it concentrates both client-facing communication and block validation duties.

In the load balanced configuration (Figure 2(2)), an Nginx instance is introduced between Caliper and the blockchain network. Caliper now targets a single virtual endpoint exposed by Nginx, which forwards each incoming request to one of the backend Ethereum nodes (Node 1, Node 2, …, Node 10) according to the round□robin policy. In this scenario, all nodes remain full validators in the PoA consensus, but the client traffic is no longer concentrated on Node 1: requests are distributed more evenly across the validator set, reducing overload on any single node and allowing us to isolate the impact of the load□balancing layer on performance and resource utilization.

In both scenarios, all nodes (including Node 1) are configured as authorized signers in the Clique PoA network.

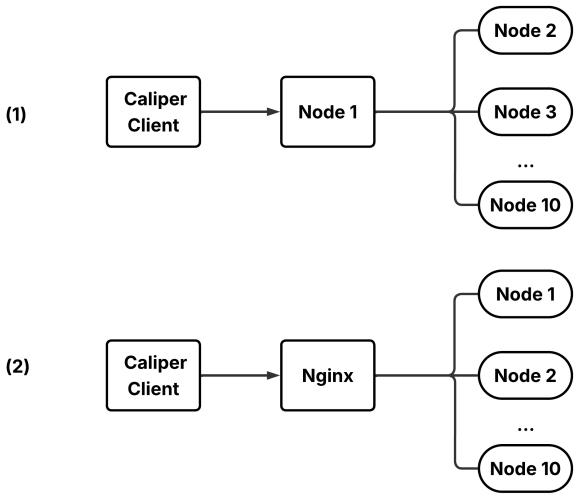


Figure 2. (1) Load distribution architecture without LB. (2) Load distribution architecture with LB.

Extensive tests were conducted under different conditions, varying transaction volume and network size, while maintaining the Clique consensus algorithm (Proof of Authority) and a fixed block creation time of one second. Each node was configured as a signatory within the network. Using the control variable method, we systematically analyzed the impact of each factor on network performance.

3.2 Experimental Procedure

The tests followed a standardized format, evaluating each configuration under nine fixed transaction volumes (TV): 100, 250, 500, 750, 1000, 1250, 1500, 1750, and 2000 transactions. To ensure greater statistical robustness, each experiment was repeated 20 times per volume, and the average of these executions was used as the representative value for each metric.

Table 1. Comparative Table of Related Work

Work	Main Focus	Methodology/Approach	Key Results/Contributions	Difference from This Work
Fan *et al*. [2022]	Performance analysis of Hyperledger Besu in private networks with LB middleware.	Evaluation of throughput, latency, resource usage, and scalability under different LB parameters.	Identified bottlenecks in transaction execution and state updates.	Focused on a single platform (Besu) and LB middleware, while our work compares Geth and Besu with a traditional LB.
Tareen *et al*. [2023]	LB scheme for healthcare systems in smart cities.	Dynamic distribution of mining tasks and prioritization of sensitive data.	Reduced delays in transmitting critical patient data, outperforming traditional algorithms.	Sector-specific application (healthcare) with customized LB, while our work is more general and focuses on traditional LB for Ethereum.
Li *et al*. [2023]	Sharding system with dynamic load balancing among blockchain shards.	Periodic migration of active accounts across shards.	Significant improvements in throughput (>10%) and reduction of confirmation delays (up to 90%).	Focused on sharding and inter-shard LB, while our work evaluates traditional LB in full Ethereum implementations.
Li *et al*. [2022]	Challenges of state sharding in blockchain.	Transformers protocol with a community-aware account partitioning algorithm.	Substantial improvements in throughput, latency, and load distribution across shards.	Focused on state sharding and account partitioning, while our work evaluates traditional LB in full Ethereum implementations.
Harish and Sridevi [2024]	Impact of LB on split-join blockchains.	Empirical study of LB impact on throughput and processing time.	LB is crucial to sustain throughput and reduce processing times under increasing transaction loads.	Focused on split-join architectures, while our work compares Geth and Besu with a traditional LB.
Samuel *et al*. [2021]	Performance evaluation of Geth, OpenEthereum (Parity), and Hyperledger Besu in private PoA networks.	Proposed a novel testing methodology and an Ethereum-specific LB middleware; experiments conducted in Microsoft Azure.	Identified bottlenecks and client-specific issues such as Geth EVM stalls and Clique fork deadlocks.	Evaluates three clients and a custom LB middleware, while our work focuses on two major clients using a traditional Nginx-based LB.
This Work	Comparative and practical evaluation of the impact of a traditional LB on Geth and Hyperledger Besu.	Analysis of throughput, latency, resource usage, Percentage Difference, Coefficient of Variation, and Metric Correlations.	Provides concrete guidelines for applying LBs in private blockchain networks.	Comparative approach of traditional LBs on two major Ethereum implementations, with additional metrics and computational resource analysis.

Transaction submission was performed using Hyperledger Caliper, configured to stress the smart contract under different workloads. The txNumber parameter defined the total number of transactions per round, while the rateControl component was set to fixed-rate mode, regulating the rate of request generation. This mechanism ensures that transactions are sent progressively until the specified total is reached, avoiding instant or batch submission.

In practice, the configuration combined three dimensions: (i) total transaction volume (txNumber), (ii) sending rate (TPS), and (iii) number of concurrent workers. This setup ensured that the load was distributed in a controlled and reproducible manner, allowing the observation of the impact of increasing transaction volume on critical metrics such as throughput, latency, CPU usage, and memory consumption.

Four main configurations were evaluated:

- Geth without Load Balancer (NLB)
- Geth with Load Balancer (LB)
- Besu without Load Balancer (NLB)
- Besu with Load Balancer (LB)

3.3 Performance Metrics

In this study, we distinguish between *workload parameters* configured in the benchmarking tool and *performance metrics* that characterize the behavior of the blockchain network under load.

Hyperledger Caliper is configured with two main workload parameters: the *transaction volume* (TV) and the average *request rate* (Req/s). The transaction volume (TV) corresponds to the total number of transactions that the client will attempt to submit during a given test run (e.g., TV = 10,000). The request rate (Req/s) specifies the average number of requests per second that Caliper tries to send until the target TV is reached (e.g., Req/s = 100 means that the client attempts to submit approximately 100 transactions per second).

It is important to emphasize Req/s is a *workload configuration parameter* of the benchmarking tool, not a performance outcome of the blockchain network itself. In other words, Req/s characterizes the intensity of the load offered to the system, whereas the actual performance of the network is captured by output metrics such as throughput (TPS), latency, and completion rate. Throughput (TPS) may be lower than the configured Req/s when the network becomes saturated.

Based on these definitions, the performance metrics analyzed in this work are:

- **Latency:** average end-to-end response time for transactions, measured from submission by the client until confirmation on the blockchain;
- **Throughput (TPS):** number of transactions effectively

committed to the blockchain per second, after validation and consensus;
- **Completion Rate:** percentage of submitted transactions that are successfully confirmed by the network;

Additional analyses were performed based on the results obtained, including:

- **Percentage differences;**
- **Coefficients of variation;**
- **Correlations between variables (Req/s, Latency, TPS);**

The percentage difference is a quantitative statistical metric used to express the relative variation between two distinct values in percentage terms [Cuemath, 2025]. The average percentage difference (Δ%) between two configurations A and B is computed as

$$\Delta\% = \frac{X_A - X_B}{X_B} \times 100$$

where $|X|$ represents the metric of interest (TPS, latency, or Req/s). In the analysis of load balancer impact (Figures 11 and 12), we set A as the configuration without load balancer (NLB) and B as the configuration with load balancer (LB), so that positive values indicate higher metric values without LB compared to with LB. For client comparisons (Figures 13 and 14), A is Geth and B is Besu. The interpretation of desirable signs varies by metric: for TPS and Req/s, higher values are preferable; for latency, lower values are preferable.

The coefficient of variation (CV) is a statistical measure that indicates the relative dispersion of a data distribution with respect to its mean. It is calculated as:

$$\mathrm{CV} = \frac{\sigma}{\mu}$$

where $|\sigma|$ is the standard deviation and $|\mu|$ is the mean of the distribution. The CV is dimensionless and is often expressed as a percentage, which allows comparing variability across different datasets regardless of their units of measurement. A lower CV indicates lower relative variability (values more concentrated around the mean), whereas a higher CV denotes greater relative dispersion [Frost, 2020]. In this study, we use the CV to assess the stability of performance metrics (such as TPS and latency) under different workloads and network configurations.

Pearson's correlation coefficient (or Pearson's $|r|$) is a statistical measure that evaluates the strength and direction of the linear relationship between two continuous variables. It is computed as:

$$r = \frac{\sum(X_i - \bar{X})(Y_i - \bar{Y})}{\sqrt{\sum(X_i - \bar{X})^2 \sum(Y_i - \bar{Y})^2}}$$

where $|X_i|$ and $|Y_i|$ are the observed values of variables $|X|$ and $|Y|$, and $|\bar{X}|$ and $|\bar{Y}|$ are their respective means. The coefficient $|r|$ ranges from -1 to +1, where +1 indicates a perfect positive linear correlation, -1 a perfect negative linear correlation, and 0 the absence of a linear relationship. Values closer to +1 or -1 denote stronger linear relationships, whereas values near 0 indicate weak or no linear relationship [Stewart, 2025]. Here, we employ Pearson's correlation to quantify how strongly metrics such as Req/s, TPS, and latency are linearly related across different scenarios.

3.4 Experiments

The following experiments were conducted:

- **Experiment 1: Horizontal Scalability —** Variation in the number of nodes (N) (2N, 4N, 6N, 8N, and 10N) with fixed hardware specifications of 4 CPU cores and 16 GB of RAM, comparing scenarios without Load Balancer (NLB) and with Load Balancer (LB). The evaluation included Req/s, Latency, TPS, and Completion Rate, as well as percentage differences, coefficients of variation, and correlations.
- **Experiment 2: Impact of the Load Balancer on Computational Resources —** CPU and memory usage data were collected to correlate resource consumption with variations in performance metrics such as TPS and latency, enabling the assessment of how the Load Balancer influences the operational efficiency of the Geth and Besu networks.

4 Analysis of Results

This section presents the results obtained in this study, which investigates the scalability of Geth and Besu networks under different node configurations, both with and without the use of a load balancer. The analyses in this section address the research questions defined in Section 1, particularly **SRQ1** — concerning the comparative performance of Geth and Besu — and **SRQ2**, which examines the influence of the load balancer on performance and scalability.

4.1 Experiment 1: Horizontal Scalability

4.1.1 Individual Network Performance

To address **SRQ1**, this subsubsection analyzes the individual performance of each network (Geth and Besu) under different workloads and node configurations, both with and without load balancing. The goal is to identify performance patterns and scalability behavior specific to each implementation.

In Figures 5, 7, 9 and 11, transaction volume (TV), request rate (Req/s) and throughput (TPS) are presented together to show, for each experimental point, both the *load offered* to the network and the *load effectively processed*. For each combination of TV and number of nodes, the yellow bar represents the effective send rate (Req/s) sustained by the Caliper client, while the blue bar inside it shows the corresponding throughput (TPS) and completion rate achieved by the blockchain network under the same workload. In this way, the figures allow the reader to observe how much of the submitted load (Req/s) is actually converted into committed transactions (TPS) as TV and the number of validators increase.

Geth NLB - The results show that the performance of Geth NLB does not increase linearly as the number of nodes grows, as shown in **Figure 3**. Latency, presented in **Figure 4**, rises with the increase in transaction volume (TV), but the behavior varies depending on the number of nodes. In the 2-node configuration, latency increases from 1.6 s under loads of 100 transactions to 3.9 s under loads of 2000 transactions—an expected behavior due to the higher processing demand per node. With 4 nodes, the response time rises sharply, reaching 6.3 s under 2000-transaction loads, likely resulting from overload in the validation and synchronization processes among validators. In the 6- and 8-node configurations, latency stabilizes between 4.5 s and 5 s, indicating that the addition of nodes contributes to better load distribution, although coordination limitations persist. With 10 nodes, latency drops to 3.0 s under 2000 transactions, showing that this configuration balances processing across validators and reduces communication bottlenecks. This behavior demonstrates that intermediate configurations tend to introduce greater coordination overhead, while a 10-node setup achieves better parallelism and network efficiency.

TPS increases with the number of nodes, but the relationship is not proportional. With 2 nodes, TPS reaches 256.2 under 2000-transaction loads. Expanding to 4 nodes, TPS decreases to 176.0, indicating that the added inter-node communication introduces additional latency and reduces effective throughput. With 6 and 8 nodes, TPS rises again, peaking at 289.4 under 2000 transactions with 6 nodes, due to improved internal load balancing among validators. With 10 nodes, TPS stabilizes at 268.9 under 2000 transactions, reflecting greater consistency and balanced utilization of network resources.

The coefficient of variation (CV) of latency with 4 nodes, as shown in **Table 2**, is the highest (49.17%), indicating instability and unpredictable behavior. Even with 6 and 8 nodes, dispersion remains high, suggesting sensitivity to load variations. In contrast, the 10-node configuration exhibits the lowest latency CV and the highest TPS, confirming that the network becomes more stable and predictable at this scale.

These results contribute to answering **SRQ1** by characterizing Geth's baseline performance and its scalability limitations in the absence of load balancing.

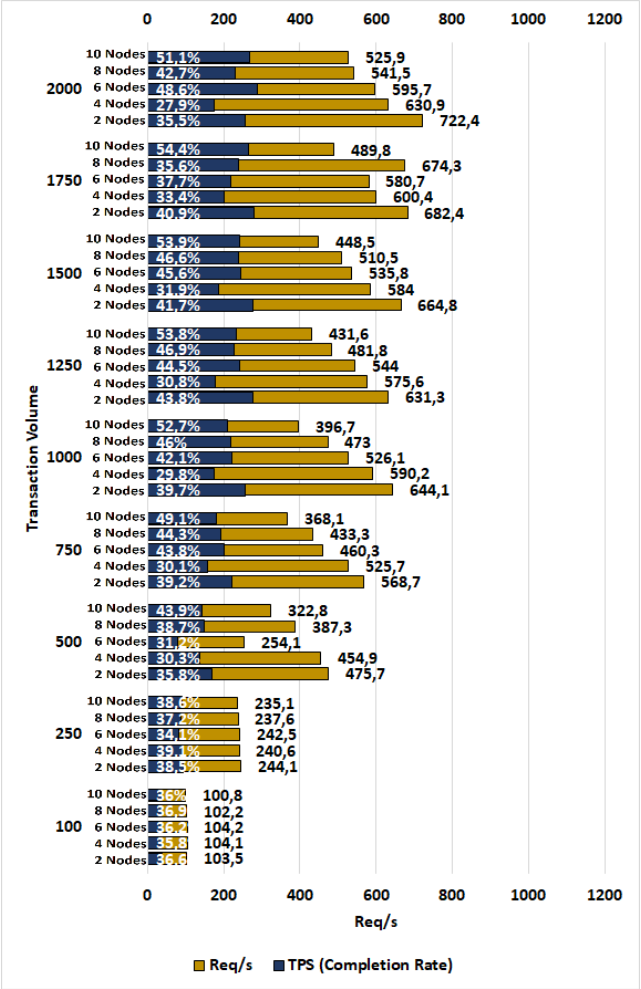


Figure 3. Geth NLB

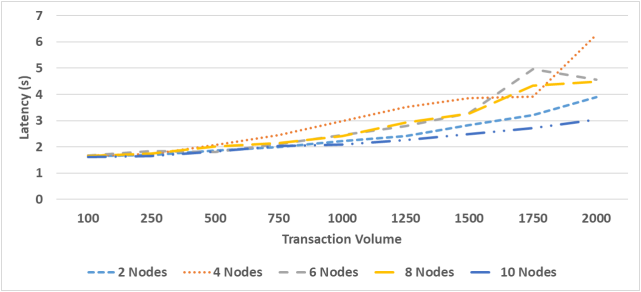


Figure 4. Geth NLB Latency

Geth LB - The introduction of the load balancer (LB) significantly altered Geth's behavior, although the improvements were not uniform across all configurations. As shown in **Figure 5**, the effective request rate (Req/s) observed by the client remained almost constant, especially under lower loads, indicating that the LB does not substantially change the intensity of the workload imposed by Caliper, but primarily influences how the incoming requests are distributed and coordinated among the validator nodes.

The impact on TPS was positive but varied. In the 2-node configuration, TPS slightly increased from 37.9 with Geth NLB to 39.3 with Geth LB under loads of 100 transactions. In the 4- and 6-node configurations, TPS fluctuated as the transaction volume increased, reflecting instability introduced by dynamic load balancing in smaller topologies. In contrast, for 8 and 10 nodes, the growth became more linear, showing that the LB operates more efficiently in networks with a larger number of validators—particularly with 10 nodes, where TPS remained consistent and close to the network's throughput limit.

Latency decreased in almost all configurations, with more pronounced variations at 6 and 8 nodes, as shown in **Figure 6**. With 10 nodes, latency reached its lowest value—2.1 s under loads of 2000 transactions—resulting from improved request distribution and reduced queuing time at each node. This demonstrates that the LB enhances communication efficiency and reduces waiting times when the network has enough validators to absorb the balancing effect.

The completion rate also improved significantly with the LB. In the 10-node configuration, it reached 71%, compared to 51.1% with Geth NLB under loads of 2000 transactions. This increase results from the more homogeneous request distribution provided by the LB, which reduces localized

Table 2. Coefficient of Variation - Geth NLB

TV	Req/s	Latency	TPS
2N			
100	2.97%	2.66%	10.77%
250	2.43%	4.05%	8.41%
500	3.03%	3.85%	3.47%
750	7.07%	6.60%	5.30%
1000	7.04%	5.18%	5.20%
1250	6.90%	5.37%	5.58%
1500	5.47%	6.84%	5.93%
1750	4.41%	7.41%	5.79%
2000	3.64%	9.65%	13.48%
4N			
100	2.62%	4.02%	7.01%
250	4.64%	3.85%	7.17%
500	6.40%	6.25%	5.53%
750	4.98%	5.57%	7.24%
1000	9.19%	6.83%	3.88%
1250	7.94%	28.18%	16.57%
1500	8.14%	12.26%	6.70%
1750	6.33%	3.71%	2.37%
2000	9.68%	49.17%	25.77%
6N			
100	3.82%	10.42%	6.53%
250	3.28%	9.94%	9.98%
500	0.00%	0.00%	0.00%
750	7.40%	8.30%	6.17%
1000	8.84%	8.86%	9.36%
1250	7.29%	11.75%	10.40%
1500	7.02%	16.25%	10.64%
1750	9.30%	74.98%	22.58%
2000	12.66%	15.98%	59.80%
8N			
100	4.09%	3.98%	10.93%
250	3.88%	9.30%	9.09%
500	4.57%	8.29%	7.39%
750	4.77%	10.82%	4.91%
1000	6.95%	8.44%	6.39%
1250	8.35%	7.16%	6.65%
1500	4.96%	7.37%	6.89%
1750	5.76%	21.36%	14.47%
2000	7.11%	11.64%	8.93%
10N			
100	2.08%	5.64%	8.29%
250	3.32%	6.28%	7.69%
500	4.73%	4.34%	6.76%
750	5.95%	7.43%	11.10%
1000	6.51%	5.37%	10.92%
1250	5.81%	5.49%	9.31%
1500	6.31%	6.09%	8.36%
1750	5.48%	6.85%	6.50%
2000	5.05%	7.68%	7.65%

overloads and minimizes confirmation failures. However, the 6- and 8-node configurations still showed variability, indicating that while the LB enhances efficiency, its effectiveness depends on the network's ability to absorb the redistributed load.

Regarding the latency CV, the 2-node configuration showed a consistent reduction compared to Geth NLB, as reported in **Table 3**, except under loads of 1000 transactions, where the CV was 24.17% (LB) versus 5.18% (NLB), reflecting a temporary instability caused by low concurrency. In the 4-node setup, variance sharply decreased from 49.17% (NLB) to 2.55% (LB) under loads of 2000 transactions, representing significant stability. Moreover, the TPS CV dropped from 25.77% (NLB) to 2.24% (LB), reinforcing the system's predictability.

The most consistent LB effect occurred in the 6-node configuration, where latency CV was reduced from 74.98% (NLB) to 16.65% (LB) under loads of 1750 transactions, and TPS CV decreased from 59.8% (NLB) to 28.6% (LB) at 2000 transactions. In the 10-node setup, the LB stabilized both latency and TPS, resulting in predictable behavior and optimized performance across all metrics.

The analysis of Geth with LB further complements **SRQ1**, revealing how the introduction of a balancing layer modifies the network's throughput, latency, and stability behavior.

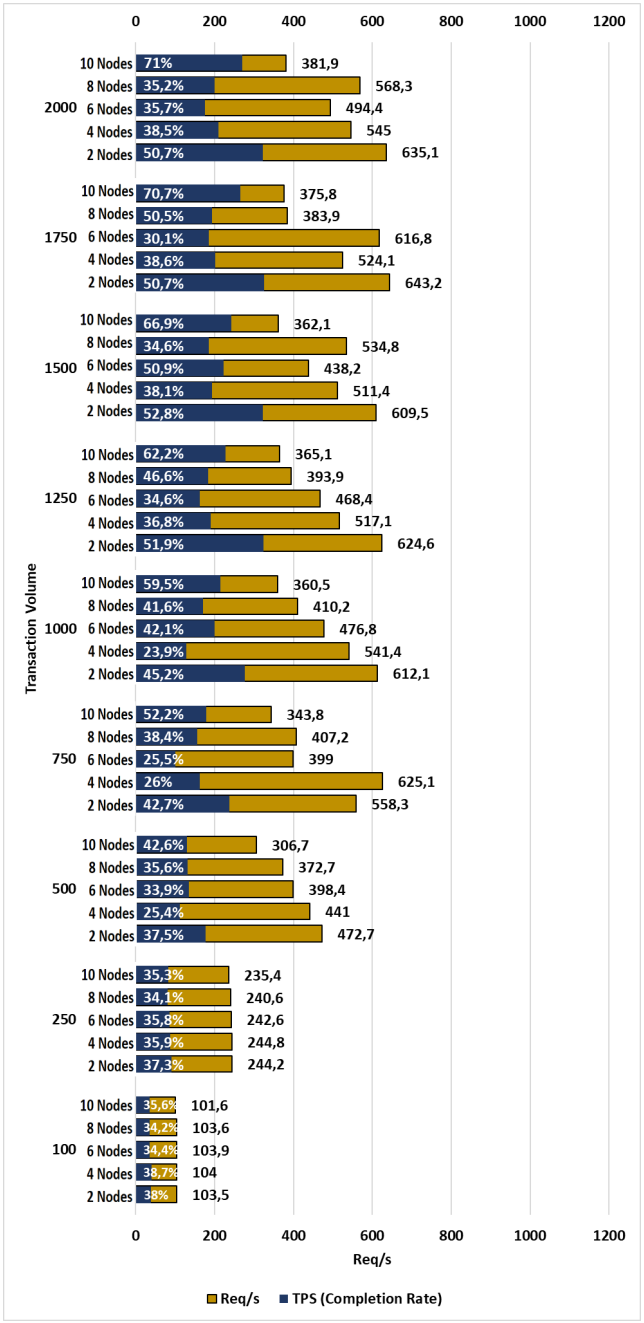


Figure 5. Geth LB

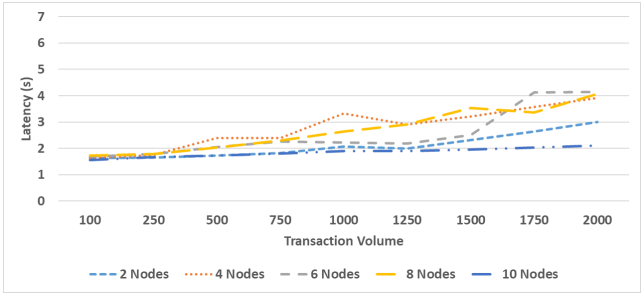


Figure 6. Geth LB Latency

Table 3. Coefficient of Variation - Geth LB

TV	Req/s	Latency	TPS
2N			
100	2.21%	1.95%	10.58%
250	1.90%	2.28%	9.89%
500	2.64%	2.78%	4.35%
750	5.96%	3.97%	2.99%
1000	6.99%	24.17%	10.80%
1250	7.04%	3.30%	4.20%
1500	6.09%	7.03%	4.00%
1750	6.15%	7.55%	5.27%
2000	6.36%	8.96%	5.59%
4N			
100	2.46%	2.09%	5.80%
250	2.50%	7.89%	6.68%
500	5.86%	17.89%	13.28%
750	8.89%	23.47%	16.01%
1000	8.15%	12.81%	25.48%
1250	11.03%	5.59%	5.50%
1500	7.23%	4.86%	3.26%
1750	6.39%	3.77%	2.72%
2000	7.42%	2.55%	2.24%
6N			
100	1.62%	7.84%	10.22%
250	4.36%	6.37%	8.72%
500	6.27%	8.92%	11.16%
750	0.00%	0.00%	0.00%
1000	11.66%	7.70%	17.33%
1250	0.00%	0.00%	0.00%
1500	8.52%	9.83%	19.02%
1750	20.78%	16.65%	9.92%
2000	7.83%	18.48%	28.60%
8N			
100	2.32%	7.18%	13.91%
250	3.59%	5.55%	7.62%
500	5.50%	3.75%	5.76%
750	9.97%	2.64%	4.75%
1000	7.05%	9.64%	5.09%
1250	13.41%	39.62%	25.16%
1500	11.39%	19.03%	10.85%
1750	12.65%	10.64%	6.85%
2000	7.75%	11.62%	6.94%
10N			
100	3.04%	9.11%	19.02%
250	3.31%	3.32%	11.77%
500	10.46%	5.71%	15.34%
750	4.94%	4.88%	9.01%
1000	4.20%	3.92%	7.76%
1250	6.33%	3.67%	8.17%
1500	5.45%	3.83%	7.19%
1750	4.85%	2.86%	4.77%
2000	7.78%	9.45%	7.12%

Besu NLB - The results shown in **Figure 7** indicate that the effective request rate (Req/s) observed by the client increases sharply as the transaction volume grows. In a two-node configuration, Req/s rises from 104.2 under loads of 100 transactions to 1084.15 under loads of 2000 transactions, which reveals that the load offered by Caliper becomes significantly more intense under higher TV values. This behavior is consistent with Besu's architecture, which favors parallel handling of incoming requests and employs dynamic execution queues, allowing the client to sustain higher submission rates as the volume increases, up to the saturation limit of the nodes.

Latency, as shown in **Figure 8**, exhibits non-linear behavior. Under low loads (between 100 and 250 transactions), latency remains low, ranging from 0.8 to 1.0 seconds, due to limited resource contention and a smaller number of simultaneous validations. As the load increases, latency grows significantly. In a two-node configuration, latency rises from 0.79 seconds under 100-transaction loads to 10.12 seconds under 2000-transaction loads. In ten nodes, latency reaches 24.68 seconds under the same conditions. This continuous increase in latency across all configurations results from the buildup of pending transactions, leading to longer waiting times in both the execution queue and the consensus process.

TPS reaches its maximum value under intermediate loads and decreases as the load intensifies. In a two-node configuration, TPS increases from 59.59 under 100-transaction loads to 222.51 under 750-transaction loads, then drops to 142.09 under 2000 transactions. This decline occurs because, under high volumes, the Clique consensus mechanism and Besu's memory management become limiting factors, preventing new transactions from being processed until previous blocks are finalized, thus reducing effective throughput.

The completion rate decreases significantly as the load increases. In a two-node setup, it drops from 52.5% under 100-transaction loads to only 9.04% under 2000-transaction loads. This reduction results from the accumulation of unconfirmed transactions and a rise in submission failures, caused by saturation of the pending transaction pool (mempool) and the validators' limited processing capacity. Consequently, Besu NLB becomes increasingly inefficient under high transaction loads.

Across all configurations, the CV of both latency and TPS, as presented in **Table 4**, shows high variability as transaction volume increases. Latency reaches a CV of 135.73% with four nodes under 1000-transaction loads, while TPS records a CV of 51.29% with ten nodes under 2000 transactions. This variation reflects performance instability under high concurrency, stemming from how Besu handles block scheduling and thread control. In contrast, Req/s variation is less pronounced, peaking at 13.2% with eight nodes under 1750 transactions, which indicates that the rate of request submission imposed by Caliper remains relatively stable even when the network's internal processing is heavily affected.

Compared with Geth NLB, Besu NLB exhibits lower latency under light loads due to its parallel execution architecture and more efficient queue management under low demand. However, at higher loads, latency increases rapidly, surpassing that of Geth. Similarly, Besu NLB achieves higher TPS under light loads but experiences a sharp decline at higher transaction volumes, peaking under intermediate loads before decreasing. This behavior highlights the early saturation of Besu's consensus mechanism and internal processing pipeline, which constrains performance gains under high transaction throughput conditions.

Besu LB - The TPS for Besu with LB and two nodes exhibits high variability as the transaction volume increases, as shown in **Figure 9**. Other node configurations display similar behavior, with TPS peaks under intermediate loads and reductions under heavier loads. Overall, the use of LB contributes to greater TPS stability, although the absolute gains remain limited. This effect occurs because the load balancer distributes incoming requests more evenly across nodes, reducing localized overload peaks without fully eliminating the internal bottlenecks of Besu's Clique consensus mechanism and memory management.

The results with the introduction of LB show that, in the two-node configuration, latency values remain similar to those observed in Besu without LB. However, as shown in **Figure 10**, other configurations exhibit significant latency reductions, especially under high transaction loads. In four nodes, the maximum latency decreases from 19.53 seconds (under 2000 transactions) to 6.08 seconds with LB. Similarly, in ten nodes, the maximum latency is reduced to 8.34 se-

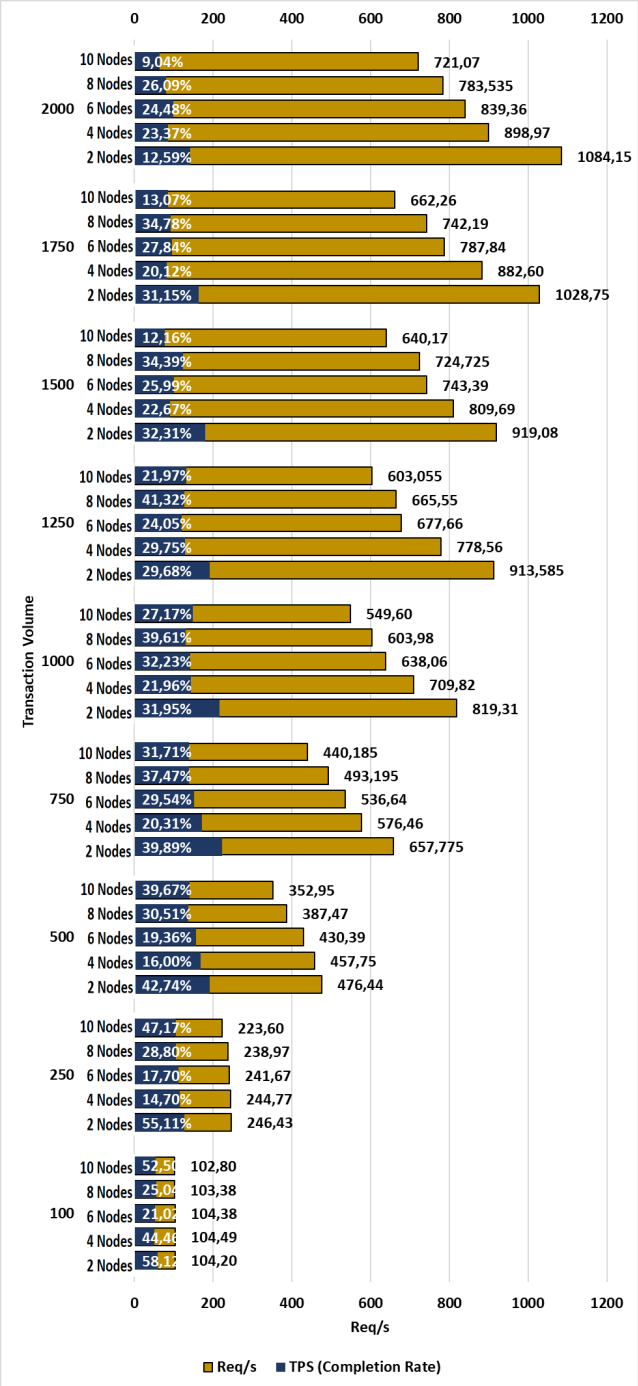


Figure 7. Besu NLB

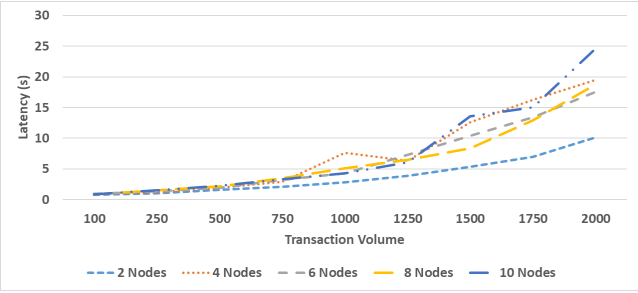


Figure 8. Besu NLB Latency

conds, contrasting with much higher values recorded in the non-LB setup. This improvement results from the redistribution of incoming requests, which alleviates transaction queue buildup on validator nodes and reduces the average waiting time before block inclusion.

Req/s values remain similar between the LB and non-LB scenarios, increasing proportionally with the transaction load. This indicates that load balancing does not restrict the request submission rate but rather alters how internal processing occurs—making it more evenly distributed across the available nodes.

Table 4. Coefficient of Variation - Besu NLB

TV	Req/s	Latency	TPS
2N			
100	2.22%	19.88%	8.97%
250	2.28%	23.38%	9.57%
500	3.92%	25.14%	14.69%
750	6.53%	25.46%	13.25%
1000	8.19%	30.35%	17.69%
1250	8.20%	29.04%	15.21%
1500	10.15%	38.79%	27.21%
1750	6.12%	31.78%	23.45%
2000	6.98%	60.93%	26.80%
4N			
100	4.56%	23.63%	17.09%
250	3.28%	39.87%	18.51%
500	7.82%	38.11%	19.58%
750	9.34%	43.56%	23.49%
1000	9.97%	135.73%	36.68%
1250	12.12%	29.30%	23.40%
1500	7.18%	40.72%	32.26%
1750	9.36%	33.47%	32.06%
2000	6.86%	57.37%	41.01%
6N			
100	3.39%	31.47%	17.13%
250	4.93%	32.90%	21.41%
500	6.66%	33.44%	19.52%
750	12.26%	40.18%	23.74%
1000	12.28%	29.34%	20.15%
1250	12.53%	34.40%	28.55%
1500	8.09%	32.53%	28.21%
1750	11.15%	46.73%	32.17%
2000	10.78%	66.64%	48.05%
8N			
100	3.07%	36.04%	17.29%
250	4.44%	45.83%	28.01%
500	12.87%	39.44%	22.04%
750	11.00%	33.96%	23.20%
1000	14.94%	38.66%	26.98%
1250	9.38%	2.36%	3.65%
1500	11.57%	46.62%	33.42%
1750	13.20%	32.64%	25.04%
2000	11.96%	41.59%	37.11%
10N			
100	3.30%	25.04%	16.01%
250	10.69%	41.98%	24.10%
500	11.80%	47.87%	22.38%
750	12.18%	44.01%	20.67%
1000	12.99%	47.06%	21.99%
1250	11.55%	40.43%	28.25%
1500	12.68%	33.55%	27.94%
1750	13.06%	60.54%	37.56%
2000	12.60%	54.12%	51.29%

Besu LB - The TPS for Besu with LB and two nodes exhibits high variability as the transaction volume increases, as shown in **Figure 9**. Other node configurations display similar behavior, with TPS peaks under intermediate loads and reductions under heavier loads. Overall, the use of LB contributes to greater TPS stability, although the absolute gains remain limited. This effect occurs because the load balancer distributes incoming requests more evenly across nodes, reducing localized overload peaks without fully eliminating the internal bottlenecks of Besu's Clique consensus mechanism and memory management.

The results with the introduction of LB show that, in the two-node configuration, latency values remain similar to those observed in Besu without LB. However, as shown in

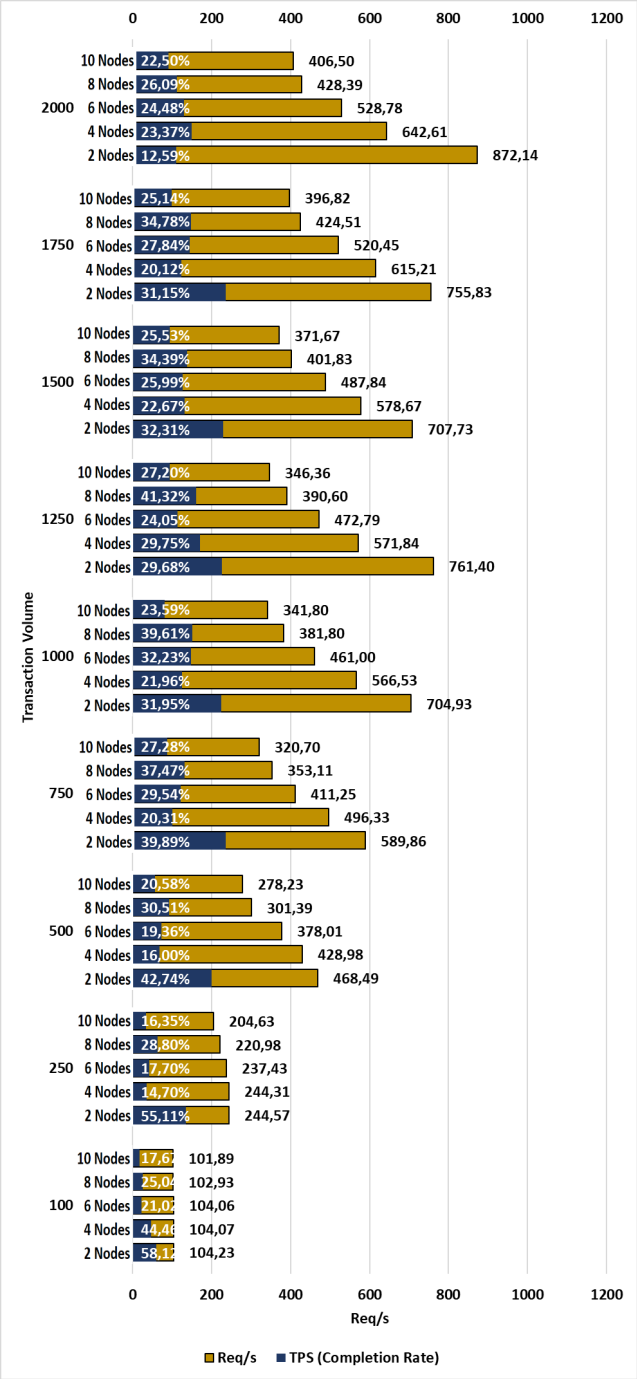


Figure 9. Besu LB

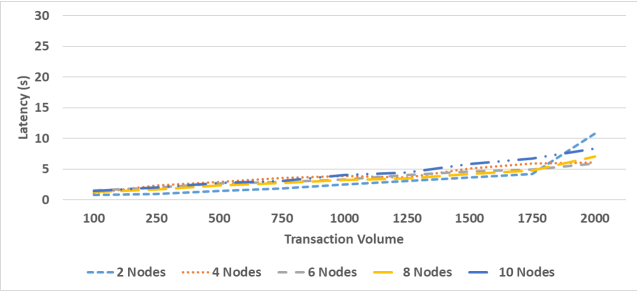


Figure 10. Besu LB Latency

Figure 10, other configurations exhibit significant latency reductions, especially under high transaction loads. In four

Table 5. Coefficient of Variation - Besu LB

TV	Req/s	Latency	TPS
2N			
100	2.27%	10.97%	13.08%
250	2.41%	16.60%	8.93%
500	5.74%	24.78%	13.04%
750	9.37%	26.91%	14.73%
1000	10.05%	22.43%	13.53%
1250	11.49%	21.06%	14.30%
1500	11.64%	21.87%	16.21%
1750	10.96%	18.02%	11.38%
2000	7.80%	45.50%	34.38%
4N			
100	1.76%	35.00%	21.41%
250	2.34%	19.23%	17.04%
500	7.17%	25.08%	20.15%
750	11.52%	19.82%	15.79%
1000	9.04%	19.52%	13.29%
1250	8.30%	18.49%	13.27%
1500	12.19%	18.19%	29.99%
1750	7.80%	13.31%	21.57%
2000	9.34%	16.43%	15.22%
6N			
100	2.26%	34.25%	23.62%
250	4.69%	22.17%	24.85%
500	12.99%	20.33%	29.36%
750	11.53%	26.39%	20.67%
1000	12.22%	28.58%	20.74%
1250	10.06%	23.00%	25.48%
1500	12.22%	22.71%	19.06%
1750	11.12%	19.19%	23.00%
2000	13.34%	23.08%	25.14%
8N			
100	3.56%	19.65%	14.81%
250	10.61%	23.25%	18.85%
500	13.17%	26.94%	14.67%
750	13.67%	30.79%	19.27%
1000	18.62%	25.88%	17.87%
1250	16.07%	27.56%	17.12%
1500	10.91%	20.07%	21.07%
1750	16.04%	22.69%	23.02%
2000	17.19%	23.26%	28.84%
10N			
100	3.13%	32.61%	19.78%
250	14.48%	26.69%	32.80%
500	13.24%	30.85%	33.97%
750	15.05%	32.59%	19.34%
1000	15.82%	34.73%	27.23%
1250	14.13%	26.63%	24.48%
1500	15.64%	27.83%	34.36%
1750	16.13%	19.44%	21.76%
2000	17.51%	33.80%	24.32%

nodes, the maximum latency decreases from 19.53 seconds (under 2000 transactions) to 6.08 seconds with LB. Similarly, in ten nodes, the maximum latency is reduced to 8.34 seconds, contrasting with much higher values recorded in the non-LB setup. This improvement results from the redistribution of incoming requests, which alleviates transaction queue buildup on validator nodes and reduces the average waiting time before block inclusion.

Completion rates tend to be higher in LB configurations under intermediate and high loads, while Besu without LB performs slightly better under light loads. This difference arises because the load balancer becomes more effective as concurrency increases, reducing the number of transactions discarded due to queue saturation or timeouts. Furthermore, the latency coefficient of variation (CV), as presented in **Table 5**, which in Besu without LB reaches peaks between 40% and 135%, is substantially reduced with the use of LB. For instance, with four nodes, the latency CV drops to values between 13% and 19%, demonstrating more predictable performance. Although the TPS CV shows minor fluctuations, the overall trend indicates greater stability in configurations with more nodes, confirming the LB's contribution to more consistent performance metrics in larger-scale scenarios.

Although Geth with LB consistently achieves lower la-

tency, Besu with LB maintains higher values due to its queue-based execution architecture and greater dependence on thread synchronization. This difference reflects distinct approaches to transaction propagation and validation: Geth prioritizes optimized asynchronous execution and efficient backlog control, whereas Besu performs stricter block-based validations, which limit performance gains under heavy load. Consequently, Geth with LB achieves superior TPS performance, particularly in configurations with a larger number of nodes, demonstrating an architecture better suited to scalability under load balancing. In contrast, Besu with LB shows moderate gains under intermediate loads but experiences sharp declines under very high transaction volumes, highlighting limitations in the interaction between the load balancer and the network's internal processing logic.

Similarly to Geth, the evaluation of Besu under both configurations contributes to **SRQ1**, highlighting client-specific performance differences and the effect of load distribution mechanisms.

Req/s values remain similar between the LB and NLB scenarios and increase proportionally with the configured transaction volume. This indicates that load balancing does not restrict the request submission rate imposed by Caliper (i.e., the offered load), but rather alters how internal processing occurs, making it more evenly distributed across the available nodes.

4.2 Load Balancer Impact on Each Network

This subsection specifically addresses **SRQ2**, which explores how the application of a load balancer affects the performance and scalability of each network. By comparing configurations with and without LB for both Geth and Besu, the results quantify the impact of the middleware on throughput, latency, and stability.

Geth NLB vs. Geth LB - The comparison between Geth with and without LB provides direct insights into **SRQ2**, revealing how load distribution alters performance trends under varying transaction volumes.

Figure 11 summarizes these effects using the average percentage difference of each metric (TPS, latency, Req/s) for Geth without LB relative to Geth with LB, by number of nodes. Positive values indicate that the metric is higher without LB than with LB, while negative values indicate that it is higher with LB. Thus, negative values are desirable for TPS and Req/s (higher throughput or send rate with LB), while positive values are desirable for latency (lower response times with LB).

The results show that, when analyzing averages, the use of LB consistently reduces Req/s across all configurations, as evidenced in Figure 11. This reduction occurs because the LB introduces an additional routing step, resulting in a small overhead that decreases the total number of requests processed per second.

Average latency is consistently lower when LB is implemented in all configurations, indicating that the LB distributes loads more effectively among nodes, preventing bottlenecks and improving the average response time. This improvement is more pronounced in the 2- and 10-node configurations, due to the greater difference in load distribution in these scenarios.

With loads ranging from 100 to 2000 transactions, LB increases TPS in the 2-node configuration, as it distributes requests more efficiently across a small number of nodes, reducing congestion. Under intermediate transaction loads in 4-node configurations, the TPS difference is minimal, as load balancing is already naturally efficient. In intermediate configurations (6 and 8 nodes), averages show higher TPS in networks without LB, indicating that the overhead introduced by LB at these scales outweighs the balancing gains. With 100 to 2000 transactions in the 10-node configuration, averages show little difference in TPS between networks with and without LB, indicating that load balancing does not significantly impact throughput at this level. Nevertheless, the consistent reduction in latency confirms that LB remains advantageous in larger networks, improving scalability by reducing response times even with a slight impact on throughput.

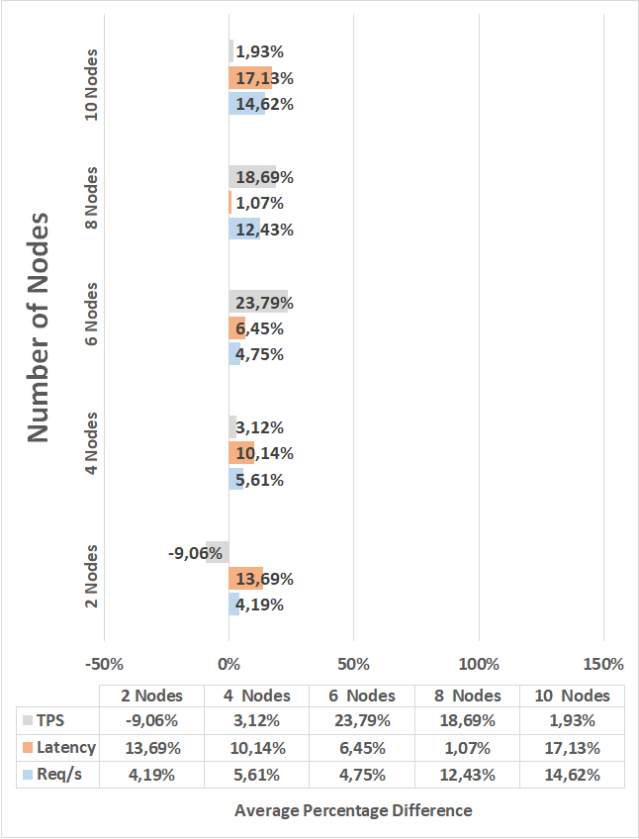


Figure 11. Average percentage difference of TPS, latency, and Req/s for Geth without LB relative to Geth with LB, by number of nodes. Positive values indicate higher values without LB than with LB; thus, negative values are desirable for TPS and Req/s (higher throughput or send rate with LB), while positive values are desirable for latency (lower response times with LB).

Besu NLB vs. Besu LB - Similarly, the Besu results offer additional evidence for **SRQ2**, showing that the balancing overhead becomes more pronounced as network size and workload increase.

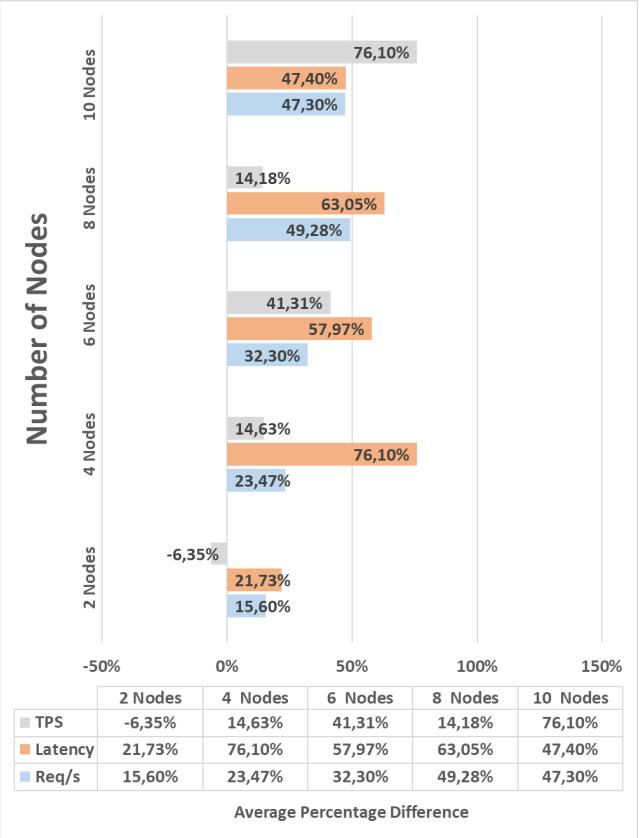
Figure 12 presents the average percentage difference of TPS, latency, and Req/s for Besu without LB relative to Besu with LB, by number of nodes. Positive values indicate that

the metric is higher without LB than with LB, while negative values indicate that it is higher with LB. As before, negative values are desirable for TPS and Req/s (higher throughput or send rate with LB), while positive values are desirable for latency (lower response times with LB).

The average results show that Besu without LB (Besu NLB) consistently achieves higher request throughput (Req/s) across nearly all configurations, as shown in Figure 12. This indicates that the absence of a load balancer avoids the additional processing overhead introduced by LB, which becomes more evident as the network size increases. However, this higher throughput comes at the cost of increased latency.

Under high-load conditions, Besu NLB exhibits greater latency compared to Besu LB. This means that, while the network without LB can process more requests overall, individual transactions take longer to complete. Conversely, the presence of LB reduces latency by distributing requests more evenly across nodes, mitigating local overloads and improving response times. This latency improvement is particularly significant under heavy workloads, where balanced resource utilization prevents bottlenecks.

Therefore, Besu NLB favors throughput efficiency (higher Req/s), while Besu LB enhances responsiveness (lower latency). The trade-off between these metrics reflects the classic balance between raw processing capacity and coordination overhead introduced by load balancing. The impact on TPS is non-linear and configuration-dependent: LB improves TPS only in the 2□node configuration. In all larger networks (4, 6, 8 and 10 nodes), Besu without LB achieves higher TPS, indicating that the coordination overhead introduced by the load balancer progressively outweighs potential gains in parallelism.



	2 Nodes	4 Nodes	6 Nodes	8 Nodes	10 Nodes
TPS	-6,35%	14,63%	41,31%	14,18%	76,10%
Latency	21,73%	76,10%	57,97%	63,05%	47,40%
Req/s	15,60%	23,47%	32,30%	49,28%	47,30%

Figure 12. Average percentage difference of TPS, latency, and Req/s for Besu without LB relative to Besu with LB, by number of nodes. Positive values indicate higher values without LB than with LB; thus, negative values are desirable for TPS and Req/s (higher throughput or send rate with LB), while positive values are desirable for latency (lower response times with LB).

4.3 Comparison Between Geth and Besu

This subsection provides a direct comparison between Geth and Besu to fully address **SRQ1**. The results highlight how architectural differences between the clients lead to distinct performance behaviors under similar network and workload conditions.

Geth NLB vs. Besu NLB - The comparison between Geth and Besu without LB establishes the baseline for **SRQ1**, revealing intrinsic performance contrasts.

Figure 13 summarizes these contrasts using the average percentage difference of TPS, latency, and Req/s for Geth NLB relative to Besu NLB, by number of nodes. Positive values indicate that the corresponding metric is higher for Geth than for Besu, while negative values indicate that Besu yields higher values. Thus, positive values are desirable for TPS and Req/s (higher throughput or send rate for Geth), whereas negative values are desirable for latency (lower response times for Geth compared to Besu).

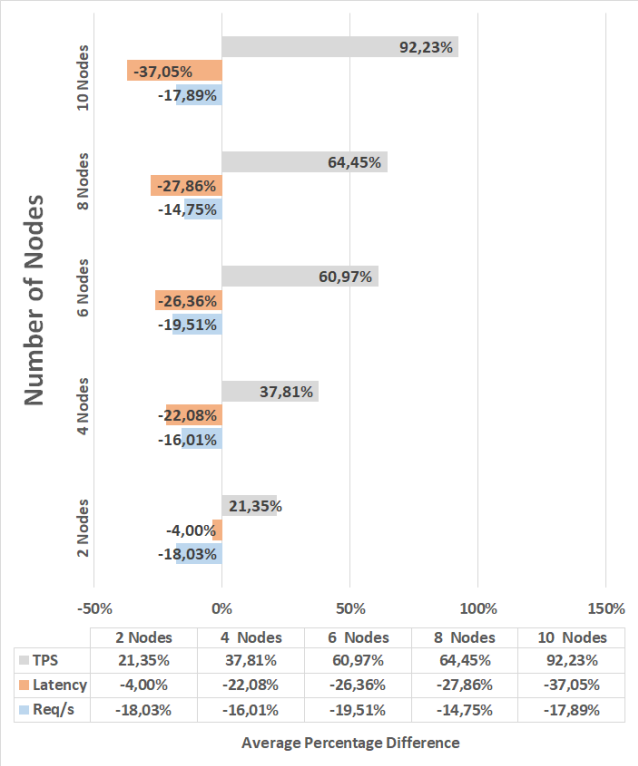
When analyzing the results, we observe that, in all configurations, the average percentage difference in Req/s for Geth NLB is negative relative to Besu NLB, as shown in Figure 13. This means that, for the same workloads, the Caliper client typically sustains a higher send rate when interacting with Besu than with Geth, reflecting how each client handles incoming requests. However, this does not directly translate into better end-to-end performance for Besu, which must be assessed through TPS and latency.

In terms of latency, the results show that Geth NLB consistently achieves lower latency than Besu NLB across all configurations, with the advantage becoming more pronounced as the number of nodes increases. Even under low-load conditions (2 nodes), Geth exhibits slightly lower latency (-4%), and this gap widens substantially in larger networks, reaching -37% at 10 nodes. This indicates that Geth's architecture provides better response times from the outset, with superior scalability under growing demand.

The analysis of TPS results reveals that Geth NLB consistently outperforms Besu NLB across all configurations, even under low loads. At 2 nodes, Geth already achieves 21% higher TPS, and this advantage escalates in high-load scenarios, with percentage differences reaching up to 92% at 10 nodes. These substantial gains highlight Geth's superior scalability in terms of throughput, particularly as network size and workload intensify.

Geth LB vs. Besu LB - The comparison with LB further extends **SRQ2**, showing how each client reacts to the introduction of an external balancing layer.

Figure 14 summarizes this comparison using the average percentage difference of TPS, latency, and Req/s for Geth LB relative to Besu LB, by number of nodes. Positive values indicate that the corresponding metric is higher for Geth than



	2 Nodes	4 Nodes	6 Nodes	8 Nodes	10 Nodes
TPS	21,35%	37,81%	60,97%	64,45%	92,23%
Latency	-4,00%	-22,08%	-26,36%	-27,86%	-37,05%
Req/s	-18,03%	-16,01%	-19,51%	-14,75%	-17,89%

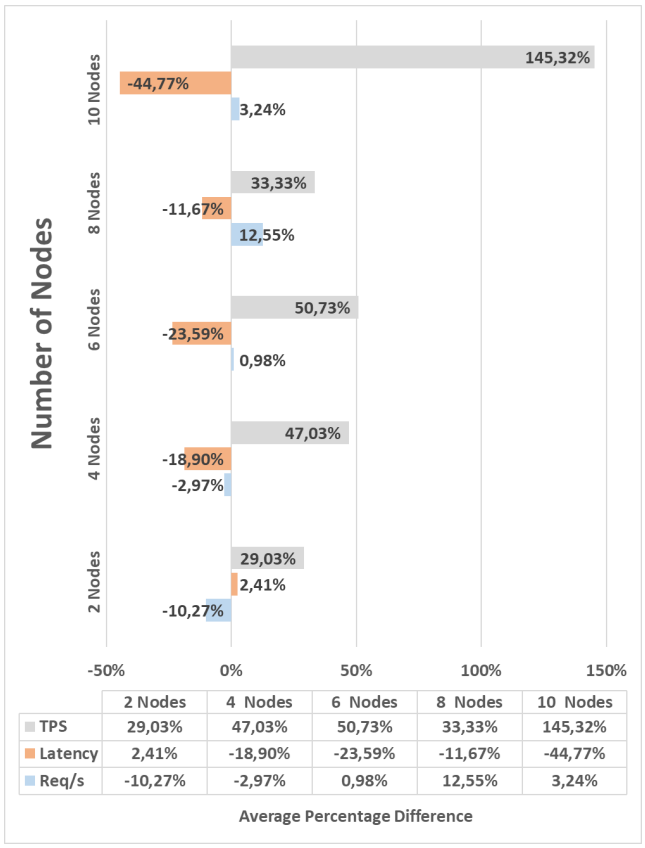
Figure 13. Average percentage difference of TPS, latency, and Req/s for Geth NLB relative to Besu NLB, by number of nodes. Positive values indicate higher values for Geth than for Besu; thus, positive values are desirable for TPS and Req/s (higher throughput or send rate for Geth), while negative values are desirable for latency (lower response times for Geth).

for Besu, while negative values indicate that Besu yields higher values. As before, positive values are desirable for TPS and Req/s (higher throughput or send rate for Geth), whereas negative values are desirable for latency (lower response times for Geth compared to Besu).

When analyzing the results for the Req/s rate in the LB configurations, we observe that the average percentage differences are relatively small across all numbers of nodes, indicating that the send rate sustained by Caliper is similar for both clients when the load balancer is present, as shown in Figure 14. Besu LB maintains a slight advantage in Req/s in the 2 and 4 nodes configurations, while in larger topologies (6, 8, and 10 nodes) the difference becomes small or shifts slightly in favor of Geth LB. Overall, these variations remain modest and do not determine performance.

In terms of latency, the averages indicate that under low load conditions, Geth LB may exhibit slightly higher latency than Besu LB. However, as the volume of transactions increases—especially in configurations with 4, 6, 8, and 10 nodes—the average difference reverses, with Geth LB providing significantly lower response times than Besu LB under high demand. This demonstrates a notable improvement in latency, particularly in networks with 10 nodes.

Geth LB consistently achieves higher TPS across all configurations, with performance gaps increasing sharply in larger networks—reaching gains above 100% under high-load conditions (e.g., 10 nodes). This finding suggests that Geth LB architecture scales more efficiently in terms of throughput.



	2 Nodes	4 Nodes	6 Nodes	8 Nodes	10 Nodes
TPS	29,03%	47,03%	50,73%	33,33%	145,32%
Latency	2,41%	-18,90%	-23,59%	-11,67%	-44,77%
Req/s	-10,27%	-2,97%	0,98%	12,55%	3,24%

Figure 14. Average percentage difference of TPS, latency, and Req/s for Geth LB relative to Besu LB, by number of nodes. Positive values indicate higher values for Geth than for Besu; thus, positive values are desirable for TPS and Req/s (higher throughput or send rate for Geth), while negative values are desirable for latency (lower response times for Geth).

4.4 Correlation Between Metrics

This section addresses **SRQ3**, which investigates how the relationships among TPS, latency, and Req/s help explain the performance behaviors observed in **SRQ1** (Geth vs. Besu without LB) and **SRQ2** (impact of LB).

We analyzed the correlation among three key relationships between the metrics: Req/s vs TPS, Latency vs TPS, and Req/s vs Latency. This analysis was conducted across various node configurations (2, 4, 6, 8, and 10 nodes). Each correlation value reflects the degree of linear dependence between the metrics, ranging from -1, indicating a perfectly negative correlation, to +1, indicating a perfectly positive correlation. The goal is to understand how efficiently each client converts load into throughput, how latency evolves with increasing demand, and how these relationships differ across configurations.

Correlation for Geth NLB - Across all node configurations (Table 6), the correlation between Req/s and TPS is extremely high (≥0.96), reaching as high as 0.99 with 10 nodes. This shows that Geth NLB converts incoming requests into transactions almost linearly, explaining why Geth achieves consistently higher TPS than Besu in SRQ1.

The correlation between latency and TPS ranges from 0.66 (with 4 nodes) to 0.91 (with 10 nodes), demonstrating a moderate to strong positive relationship. This pattern matches the results in Section 4.1: although absolute latency decreases as

nodes scale, its variation aligns closely with throughput variation. A Pearson correlation of 0.91 in the case of 10 nodes suggests that as TPS varies, latency also varies consistently in the same direction, indicating a strong linear relationship between the two metrics.

For the correlation between Req/s and latency, we observe positive correlations ranging from 0.72 to 0.90. This means that as the number of requests sent per second increases, latency tends to rise in a correlated manner as well. This reflects expected queue-growth behavior: as load rises, so does response time.

In summary: Geth NLB behaves as a highly linear system: more load → more TPS → higher latency in predictable proportions.

Table 6. Correlation Geth NLB

	2N	4N	6N	8N	10N
Req/s x TPS	0.98	0.98	0.98	0.96	0.99
Latency x TPS	0.70	0.66	0.73	0.76	0.91
Req/s - Latency	0.76	0.72	0.77	0.83	0.90

Correlation for Geth LB - The Table 7 shows the correlation results between metrics for Geth LB.

For Geth LB, Req/s × TPS correlations remain high but slightly more volatile (0.84–0.97), reflecting the additional routing and coordination overhead introduced by the load balancer. This explains why TPS differences between LB and NLB vary more in SRQ2, especially in mid-scale networks (4–6 nodes).

Latency × TPS correlations range from weak (0.54) to nearly perfect (0.98). The low value in 6 nodes aligns with the irregular latency spikes observed in this configuration, whereas the near-perfect correlation in 10 nodes explains Geth LB's strong scalability under heavy load.

Req/s × latency correlations (0.68–0.88) show that latency rises consistently—but not perfectly linearly—with increasing requests, again reflecting load-balancer overhead.

In summary: With LB, Geth remains linear but exhibits more variance in mid-range configurations, consistent with the performance fluctuations observed in Section 4.2.

Table 7. Correlation Geth LB

	2N	4N	6N	8N	10N
Req/s x TPS	0.97	0.84	0.85	0.91	0.93
Latency x TPS	0.76	0.86	0.54	0.85	0.98
Req/s - Latency	0.68	0.75	0.73	0.85	0.88

Correlation for Besu NLB - Besu NLB presents a contrasting profile (Table 8). The correlation coefficients between Req/s and TPS are very low (0.04–0.55), meaning increased request volume does not translate predictably into TPS. This explains the behavior discussed in SRQ1, where Besu frequently sustains higher Req/s but achieves lower TPS than Geth.

The correlation between latency and TPS is near zero or even negative, especially in configurations with 4, 8, and 10 nodes, indicating irregular behavior: changes in throughput do not consistently affect latency. Combined with the strong Req/s × latency correlations (0.80–0.85), this suggests that Besu NLB increases latency under load without proportionally increasing TPS.

In summary: Besu NLB exhibits weak coupling between load and throughput, which explains its lower scalability compared to Geth NLB.

Table 8. Correlation Besu NLB

	2N	4N	6N	8N	10N
Req/s x TPS	0.55	0.05	0.22	0.27	0.04
Latency x TPS	0.07	-0.41	-0.24	-0.26	-0.50
Req/s - Latency	0.84	0.85	0.85	0.82	0.80

Correlation for Besu LB - With load balancing (Table 9), Besu shows significantly stronger Req/s × TPS correlations (0.59–0.96). This improvement explains why, in SRQ2, Besu LB achieves TPS gains in some configurations: the load balancer helps convert additional requests into confirmed transactions more efficiently.

Latency × TPS correlations become positive and stronger (0.75–0.79) in medium and large networks, indicating more predictable latency growth with increasing throughput. Req/s × latency remains consistently high (≥0.70), reinforcing the intuitive behavior of latency increasing with load.

In summary: Besu LB becomes more predictable and efficient than Besu NLB, but still less linear than Geth LB overall.

Table 9. Correlation Besu LB

	2N	4N	6N	8N	10N
Req/s x TPS	0.59	0.88	0.94	0.91	0.96
Latency x TPS	-0.14	0.75	0.78	0.53	0.79
Req/s - Latency	0.70	0.90	0.88	0.80	0.86

Overall Interpretation (Answer to SRQ3) - The correlation analysis reveals the structural reasons behind the behaviors observed in **SRQ1** and **SRQ2**:

- **Geth (NLB and LB):** Strong Req/s × TPS and Latency × TPS correlations show highly linear and scalable behavior, explaining its consistently higher TPS and lower latency trends.
- **Besu NLB:** Weak Req/s × TPS and inconsistent Latency × TPS correlations show inefficient conversion of load into throughput, explaining its lower scalability despite higher Req/s.
- **Besu LB:** Improved correlations illustrate why LB benefits Besu more than Geth in some scenarios, but not enough to match Geth's scalability.

Thus, SRQ3 confirms that the correlation structure of each client directly explains: (1) why Geth outperforms Besu intrinsically (SRQ1), and (2) why load balancing impacts each client differently (SRQ2).

4.5 Experiment 2: Impact of the Load Balancer on Computational Resources

This experiment investigates how the load balancer affects the computational footprint of the blockchain networks, focusing on CPU and memory usage. The goal is to understand how variations in load distribution influence overall efficiency when comparing scenarios with and without LB. This analysis directly complements **SRQ2**, extending the performance discussion by examining whether the balancing strategy introduces resource overhead or alleviates localized pressure on specific nodes.

The architecture used in Experiment 2 is the same described in Section 3 (Figure 2), composed of ten validator nodes participating in the Clique PoA consensus. Without LB, Node 1 receives all incoming requests from the Hyperledger Caliper client before propagating them to the remaining validators. With LB enabled, Nginx acts as an intermediary, forwarding requests through round□robin distribution across all nodes. This configuration isolates the effect of the LB layer on both performance and resource consumption while keeping all nodes as authorized signers.

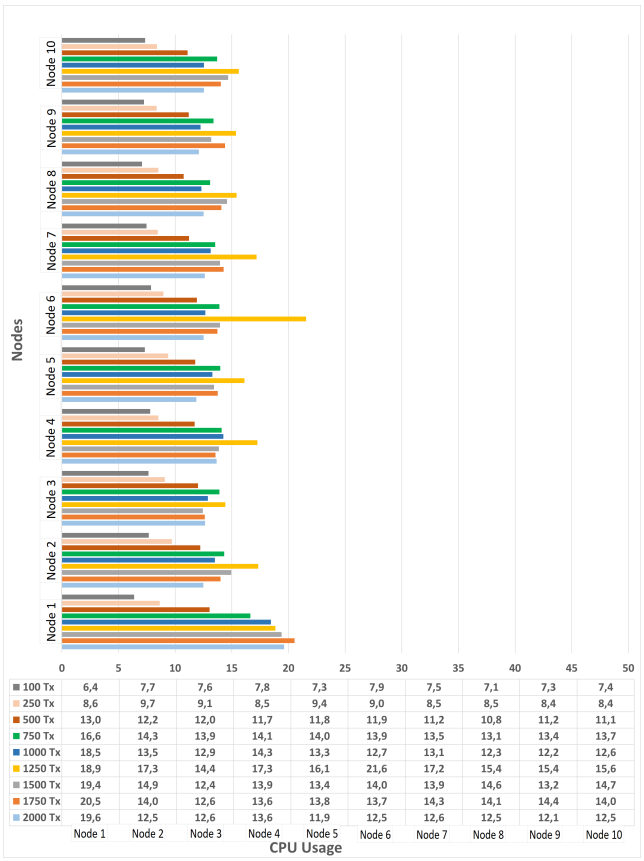
The following figures present CPU and memory utilization across the ten nodes for Geth and Besu, with and without LB. Each chart summarizes the average resource usage observed during nine workload levels (100 to 2000 transactions). The horizontal bars indicate the mean utilization per node, and the accompanying tables provide the corresponding numerical values. This visualization facilitates a detailed comparison of how computational load was distributed under each configuration, revealing whether specific nodes became bottlenecks, how the LB redistributed pressure, and how resource usage evolved with increasing transaction volume.

4.5.1 CPU and Memory Usage - Geth NLB and Geth LB

When analyzing the results of load distribution and CPU usage without a load balancer, as shown in **Figure 15**, we note that the first node—which typically receives incoming requests first—exhibits higher CPU usage. This indicates that it is processing a larger volume of transactions. The reason for this is that the first node is the default address configured in the Caliper client, resulting in it handling the majority of requests. Metrics such as Req/s and TPS reflect this scenario, where processing can become a bottleneck, leading to increased latency.

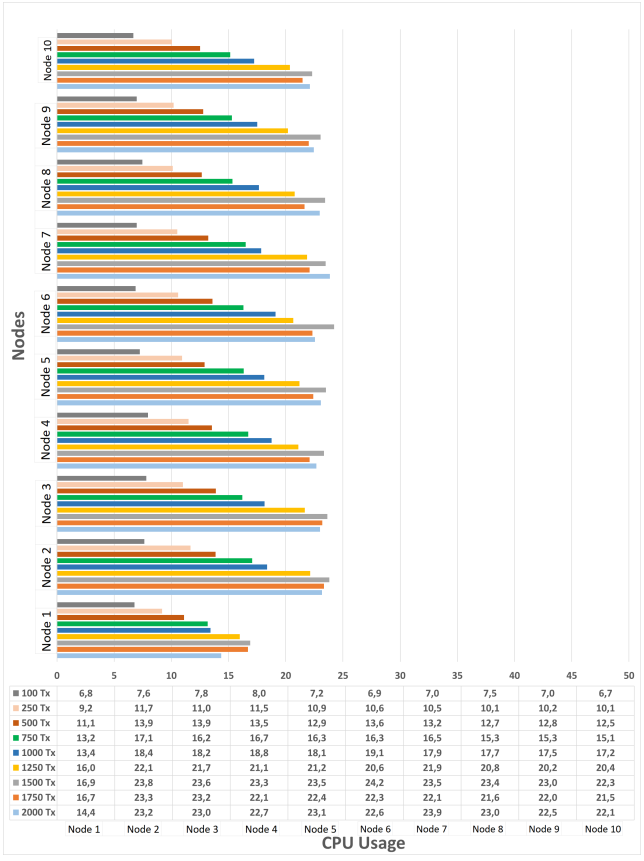
By employing nginx as a load balancer installed on the first node, the request load is redistributed among the other nodes. As a result, the first node, now functioning as the entry point and distributor, shows lower CPU usage, while the backend nodes experience an increase in their CPU usage, as shown in **Figure 16**. This redistribution fosters greater uniformity in processing and can yield improvements in latency (lower response times) and, in some cases, increased or stabilized TPS.

With the load balancer in place, the average RAM usage is significantly reduced compared to scenarios without it, as illustrated in **Figure 17** and **Figure 18**. This reduction suggests that the balancer helps prevent local cache or buffer overloads on each node, leading to more efficient resource utilization. The decreased memory usage also contributes to



	Node 1	Node 2	Node 3	Node 4	Node 5	Node 6	Node 7	Node 8	Node 9	Node 10
100 Tx	6,4	7,7	7,6	7,8	7,3	7,9	7,5	7,1	7,3	7,4
250 Tx	8,6	9,7	9,1	8,5	9,4	9,0	8,5	8,5	8,4	8,4
500 Tx	13,0	12,2	12,0	11,7	11,8	11,9	11,2	10,8	11,2	11,1
750 Tx	16,6	14,3	13,9	14,1	14,0	13,9	13,5	13,1	13,4	13,7
1000 Tx	18,5	13,5	12,9	14,3	13,3	12,7	13,1	12,3	12,2	12,6
1250 Tx	18,9	17,3	14,4	17,3	16,1	21,6	17,2	15,4	15,4	15,6
1500 Tx	19,4	14,9	12,4	13,9	13,4	14,0	13,9	14,6	13,2	14,7
1750 Tx	20,5	14,0	12,6	13,6	13,8	13,7	14,3	14,1	14,4	14,0
2000 Tx	19,6	12,5	12,6	13,6	11,9	12,5	12,6	12,5	12,1	12,5

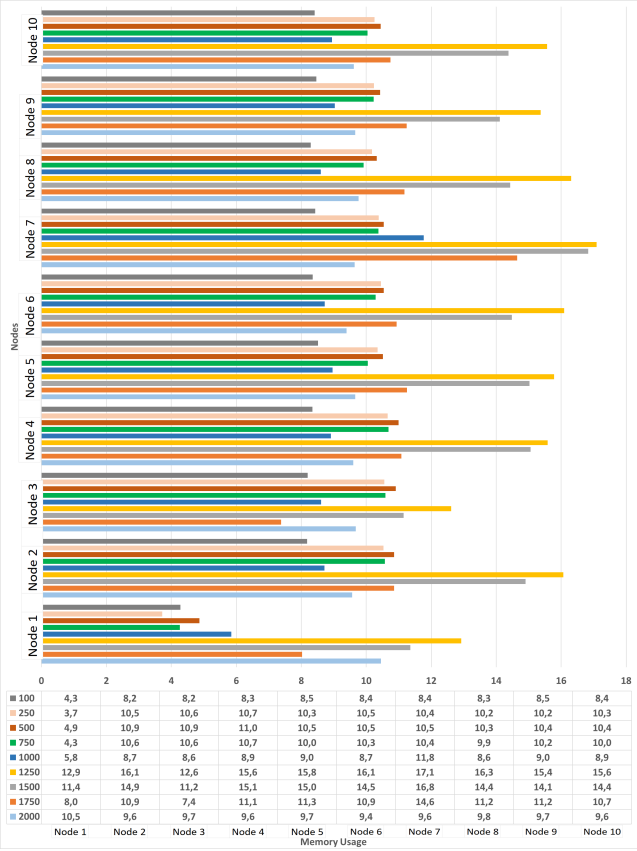
Figure 15. Average CPU usage for Geth NLB



	Node 1	Node 2	Node 3	Node 4	Node 5	Node 6	Node 7	Node 8	Node 9	Node 10
100 Tx	6,8	7,6	7,8	8,0	7,2	6,9	7,0	7,5	7,0	6,7
250 Tx	9,2	11,7	11,0	11,5	10,9	10,6	10,5	10,1	10,2	10,1
500 Tx	11,1	13,9	13,9	13,5	12,9	13,6	13,2	12,7	12,8	12,5
750 Tx	13,2	17,1	16,2	16,7	16,3	16,3	16,5	15,3	15,3	15,1
1000 Tx	13,4	18,4	18,2	18,8	18,1	19,1	17,9	17,7	17,5	17,2
1250 Tx	16,0	22,1	21,7	21,1	21,2	20,6	21,9	20,8	20,2	20,4
1500 Tx	16,9	23,8	23,6	23,3	23,5	24,2	23,5	23,4	23,0	22,3
1750 Tx	16,7	23,3	23,2	22,1	22,4	22,3	22,1	21,6	22,0	21,5
2000 Tx	14,4	23,2	23,0	22,7	23,1	22,6	23,9	23,0	22,5	22,1

Figure 16. Average CPU usage for Geth LB

lower latency, as the nodes are less burdened when accessing the data needed to process transactions.



	Node 1	Node 2	Node 3	Node 4	Node 5	Node 6	Node 7	Node 8	Node 9	Node 10
100	4,3	8,2	8,2	8,3	8,5	8,4	8,4	8,3	8,5	8,4
250	3,7	10,5	10,6	10,7	10,3	10,5	10,4	10,2	10,2	10,3
500	4,9	10,9	10,9	11,0	10,5	10,5	10,5	10,3	10,4	10,4
750	4,3	10,6	10,6	10,7	10,0	10,3	10,4	9,9	10,2	10,0
1000	5,8	8,7	8,6	8,9	9,0	8,7	11,8	8,6	9,0	8,9
1250	12,9	16,1	12,6	15,6	15,8	16,1	17,1	16,3	15,4	15,6
1500	11,4	14,9	11,2	15,1	15,0	14,5	16,8	14,4	14,1	14,4
1750	8,0	10,9	7,4	11,1	11,3	10,9	14,6	11,2	11,2	10,7
2000	10,5	9,6	9,7	9,6	9,7	9,4	9,6	9,8	9,7	9,6

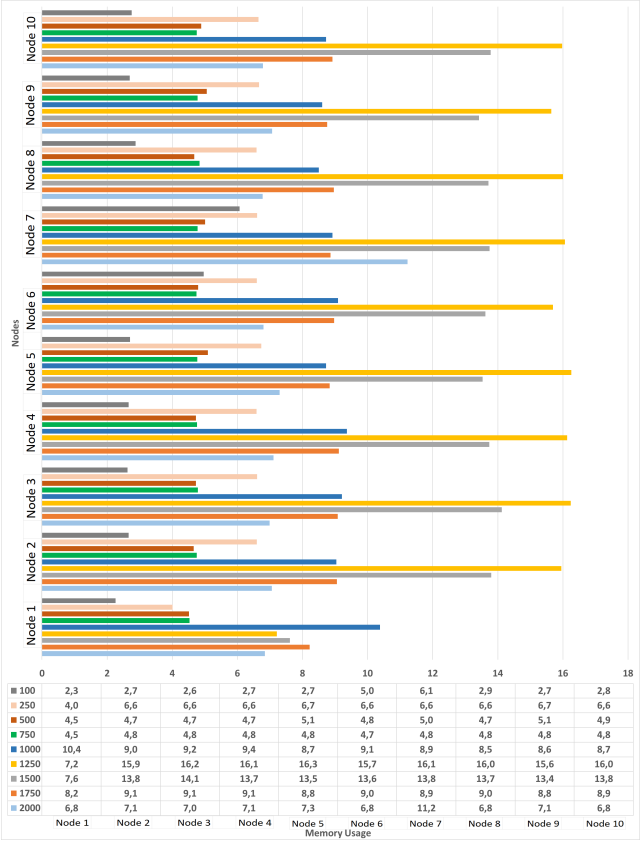
Figure 17. Average RAM usage for Geth NLB

When correlating performance metrics with resource utilization, we find that as the network processes more requests, the CPU demands increase. In scenarios without a load balancer, the first node becomes overloaded, which can limit throughput. In contrast, with a load balancer in place, the increased CPU usage in the backend nodes—responsible for processing transactions—can be viewed as a "cost" associated with achieving more consistent TPS and improved load distribution.

The enhancement in response times, indicated by lower latency when the load balancer is active, can be attributed to the fact that balancing prevents any single machine from becoming overloaded. Although this results in increased CPU usage on the backend nodes, the overall system responds more quickly, thereby reducing latency and improving transaction completion rates. The higher or more stable completion rate observed with the load balancer suggests a reduction in bottlenecks, as nodes experience less saturation, leading to a greater likelihood of successful transaction processing.

4.5.2 CPU and Memory Usage - Besu NLB and Besu LB

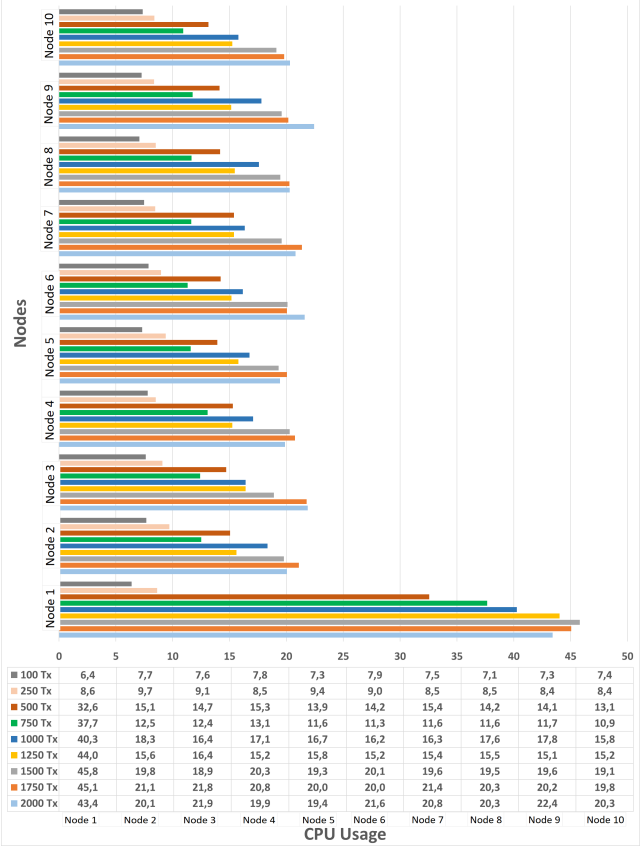
The analysis of the results from the Besu NLB network, as illustrated in the **Figure 19**, shows that for all transaction volumes, node 1 has a significantly higher average CPU usage compared to the other nodes. For instance, in TV2000, the average CPU usage for node 1 is 43.42%, whereas the average for the other nodes ranges from 19% to 22%. This suggests



	Node 1	Node 2	Node 3	Node 4	Node 5	Node 6	Node 7	Node 8	Node 9	Node 10
100	2,3	2,7	2,6	2,7	2,7	5,0	6,1	2,9	2,7	2,8
250	4,0	6,6	6,6	6,6	6,7	6,6	6,6	6,6	6,7	6,6
500	4,5	4,7	4,7	4,7	5,1	4,8	5,0	4,7	5,1	4,9
750	4,5	4,8	4,8	4,8	4,8	4,7	4,8	4,8	4,8	4,8
1000	10,4	9,0	9,2	9,4	8,7	9,1	8,9	8,5	8,6	8,7
1250	7,2	15,9	16,2	16,1	16,3	15,7	16,1	16,0	15,6	16,0
1500	7,6	13,8	14,1	13,7	13,5	13,6	13,8	13,7	13,4	13,8
1750	8,2	9,1	9,1	9,1	8,8	9,0	8,9	9,0	8,8	8,9
2000	6,8	7,1	7,0	7,1	7,3	6,8	11,2	6,8	7,1	6,8

Figure 18. Average RAM usage for Geth LB

that node 1 is handling the majority of requests, leading to overload and a potential bottleneck.



	Node 1	Node 2	Node 3	Node 4	Node 5	Node 6	Node 7	Node 8	Node 9	Node 10
100 Tx	6,4	7,7	7,6	7,8	7,3	7,9	7,5	7,1	7,3	7,4
250 Tx	8,6	9,7	9,1	8,5	9,4	9,0	8,5	8,5	8,4	8,4
500 Tx	32,6	15,1	14,7	15,3	13,9	14,2	15,4	14,2	14,1	13,1
750 Tx	37,7	12,5	12,4	13,1	11,6	11,3	11,6	11,6	11,7	10,9
1000 Tx	40,3	18,3	16,4	17,1	16,7	16,2	16,3	17,6	17,8	15,8
1250 Tx	44,0	15,6	16,4	15,2	15,8	15,2	15,4	15,5	15,1	15,2
1500 Tx	45,8	19,8	18,9	20,3	19,3	20,1	19,6	19,5	19,6	19,1
1750 Tx	45,1	21,1	21,8	20,8	20,0	20,0	21,4	20,3	20,2	19,8
2000 Tx	43,4	20,1	21,9	19,9	19,4	21,6	20,8	20,3	22,4	20,3

Figure 19. Average CPU usage for Besu NLB

With the implementation of load balancing, the traffic is distributed more evenly among all nodes, as shown in **Figure 20**. As a result, the average CPU usage of node 1 decreases significantly, while the other nodes experience an increase in usage. In TV2000, for example, node 1's average CPU drops to 25.75%, while the other nodes range from 29.29% to 34.97%. This indicates a more balanced load across the nodes.

In the Besu NLB setup, the average memory usage can reach high levels across several nodes, especially under high loads, as seen in the **Figure 21**. Although with LB the average memory values remain elevated, as shown in **Figure 22**, these nodes are capable of processing a greater number of transactions.

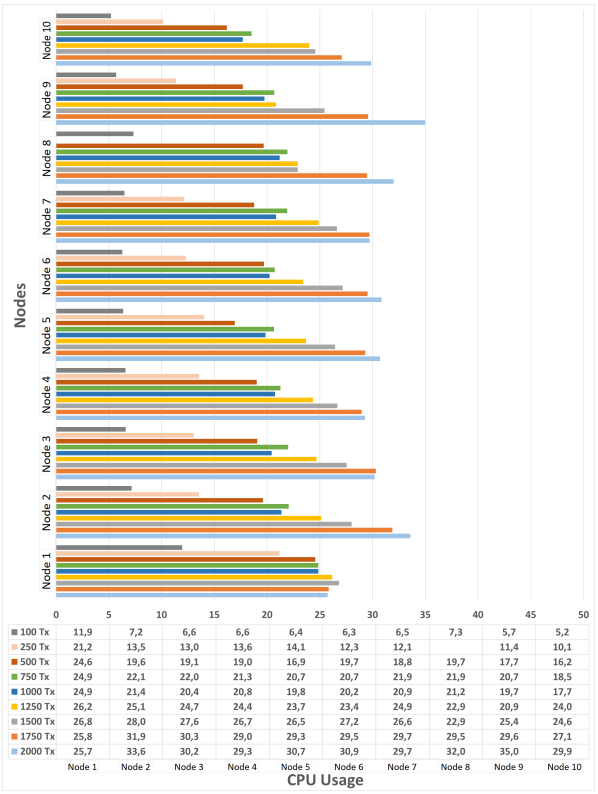


Figure 20. Average CPU usage for Besu LB

When we connect resource utilization to performance metrics, we observe that when node 1 is no longer a bottleneck, the network can handle more requests in a distributed manner. This is reflected in higher or stable TPS, as there is no single point of saturation. The additional CPU capacity available on the backend nodes suggests that they are processing more transactions, which could lead to increased or stabilized throughput. A better-distributed load helps avoid memory spikes on a single node, allowing each node to respond more quickly to the requests it receives, thereby reducing overall latency. Furthermore, without overloading a single node, the likelihood of transactions failing or timing out decreases, which typically leads to an improved completion rate.

4.5.3 Comparative Conclusion

Comparing Geth and Besu by analyzing performance metrics with resource usage reveals important differences and

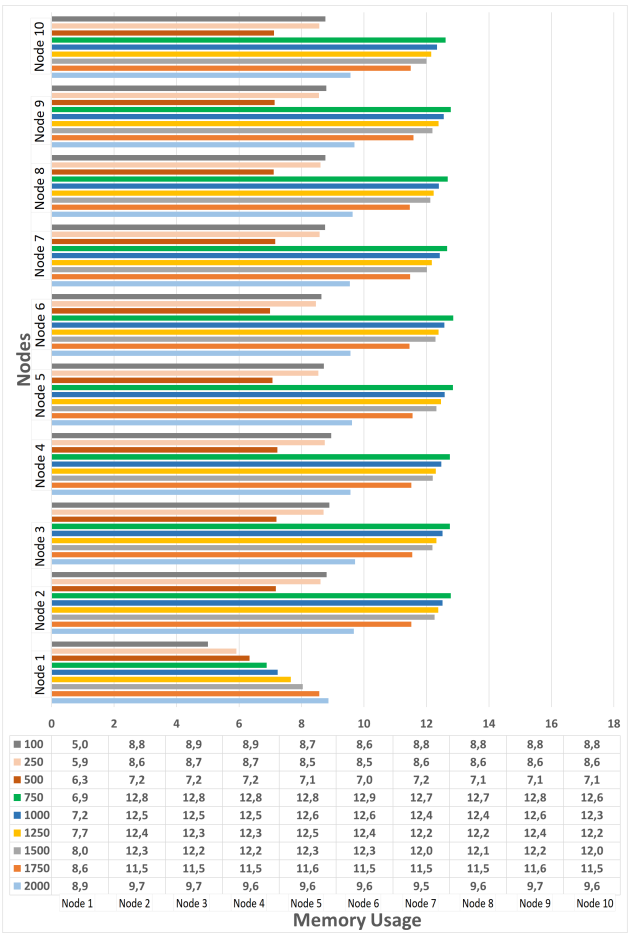


Figure 21. Average RAM usage for Besu NLB

similarities.

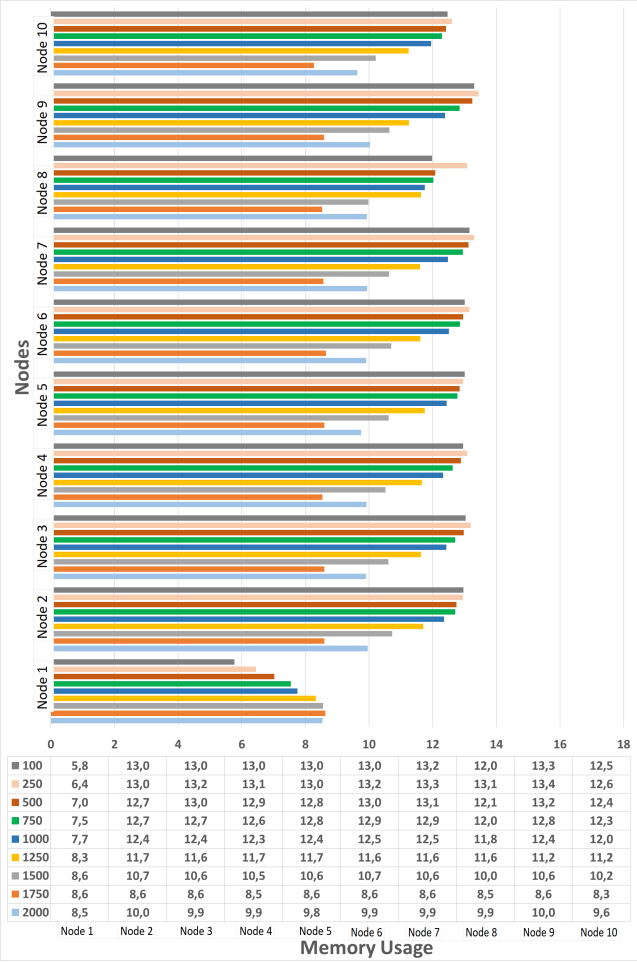
Both systems experience some overhead on the first node, but the average CPU utilization in the Besu NLBtends to be higher than in the Geth NLB. Load Balancing improves resource distribution in both cases, but Besu LB results in higher average CPU usage (around 30%) compared to Geth LB (around 22-23%). This suggests that while LB redistributes the load, the computational overhead is more pronounced in Besu.

When it comes to average RAM utilization, both systems are similar, and the implementation of LB does not significantly increase memory overhead. This efficient memory usage with LB helps maintain low latency despite an increase in TPS.

In summary, Geth shows a notable improvement with LB, featuring lower CPU usage on the first node, stable TPS, and reduced latency with moderate overhead. Although Besu also benefits from LB, it exhibits higher average CPU consumption, which may reflect differences in their architectures. In both cases, memory is utilized optimally, contributing to reduced latency and higher completion rates.

5 Conclusion

This study conducted a systematic performance evaluation of the Geth and Besu blockchain clients, focusing on the influence of load balancing on key performance metrics, including requests per second (Req/s), transactions per second (TPS),



	Node 1	Node 2	Node 3	Node 4	Node 5	Node 6	Node 7	Node 8	Node 9	Node 10
100	5,8	13,0	13,0	13,0	13,0	13,0	13,2	12,0	13,3	12,5
250	6,4	13,0	13,2	13,1	13,0	13,2	13,3	13,1	13,4	12,6
500	7,0	12,7	13,0	12,9	12,8	13,0	13,1	12,1	13,2	12,4
750	7,5	12,7	12,7	12,6	12,8	12,9	12,9	12,0	12,8	12,3
1000	7,7	12,4	12,4	12,3	12,4	12,5	12,5	11,8	12,4	12,0
1250	8,3	11,7	11,6	11,7	11,7	11,6	11,6	11,6	11,2	11,2
1500	8,6	10,7	10,6	10,5	10,6	10,7	10,6	10,0	10,6	10,2
1750	8,6	8,6	8,6	8,5	8,6	8,6	8,6	8,5	8,6	8,3
2000	8,5	10,0	9,9	9,9	9,8	9,9	9,9	9,9	10,0	9,6

Figure 22. Average RAM usage for Besu LB

and latency. The experiments were performed under multiple node configurations to enable a comprehensive comparison between both clients and to assess the specific impact of the Load Balancer (LB).

The findings indicate that Geth achieved superior overall performance, exhibiting higher efficiency in converting Req/s into TPS and lower variability in response times. These results show that Geth is more suitable for scenarios demanding high scalability and throughput. Conversely, Besu demonstrated more effective latency control in configurations that employed a Load Balancer, making it potentially more advantageous for applications where rapid response is a primary requirement.

The inclusion of the Load Balancer contributed to improved load distribution and reduced latency. However, it also increased CPU utilization in transaction-processing nodes. Consequently, while the adoption of load balancing is beneficial in high-demand environments, it is essential to ensure that computational resources are adequately provisioned.

In terms of scalability, both clients displayed distinct behavioral patterns as the number of nodes increased. Geth maintained a consistent growth trend in TPS, whereas Besu exhibited greater sensitivity to workload variations. Moreover, a negative correlation between TPS and latency was observed in both clients, although Geth was less affected by latency increases.

In summary, the selection between Geth and Besu should be guided by the operational requirements of the target application. Geth is particularly well suited for payment systems, decentralized marketplaces, and large-scale environments that demand efficient transaction processing. Besu, on the other hand, may be preferable in contexts where low latency is critical, provided that sufficient computational capacity is available to accommodate its higher resource consumption.

The adoption of a Load Balancer is recommended for multi-node deployments, particularly in networks where transaction distribution is essential to mitigate bottlenecks.

Future work should extend this investigation by exploring alternative load balancing architectures, assessing the influence of different consensus algorithms, and examining additional optimization strategies for transaction execution and network scalability.

Acknowledgements

This project was sponsored by CAPES, CNPq (312687/2025-7, and FAPERJ ($E_31/2025LIN$).

Authors' Contributions

IA and AR contributed to the conception of this study. IA performed the experiments and evaluations. IA and AR are the manuscript's writers; both read and approved the final manuscript.

Competing interests

The authors declare that they have no competing interests.

References

Adulla, M., Baset, S., Bharathan, V., Graham, G., Hochstetler, G., Kocsis, I., Little, T., Middleton, D., Simpson, M., Sukhwani, H., and Wagner, M. (2018). Hyperledger blockchain performance metrics white paper. Available at:`https://www.lfdecentraltrust.org/learn/publications/blockchain-performance-metrics`. Accessed: 2025-03-26.

Alesh, S. (2023). What is nginx? Available at:`https://medium.com/@sami.alesh/what-is-nginx-7db76b2e79f8`. Accessed: 2025-09-24.

Becher, B. (2024). What are blockchain nodes and how do they work? Available at:`https://builtin.com/blockchain/blockchain-node`. Accessed: 2025-03-27.

Carneiro, A. (2023). Introducing hyperledger besu: Your guide. Available at:`https://shre.ink/qA5r`. Accessed: 2025-03-25.

Chen, X., Nguyen, K., and Sekiya, H. (2022). On the latency performance in private blockchain networks. *IEEE Internet of Things Journal*, 9(19):18047–18058. DOI: 10.1109/JIOT.2022.3165666.

Cuemath (2025). Percentage Difference Formula - What is Percentage Difference Formula? Examples. Available at:`https://www.cuemath.com/percent-difference-formula/`.

Elaurichenickson (2024). How to use nginx as a load balancer for your application. Available at: `https://medium.com/`

@elaurichetoho/how-to-use-nginx-as-a-load-balancer-for-your-application-d80ca40f28d8. Accessed: 2025-09-24.

Energy Web Foundation (2023). Energy web chain: Public blockchain for the energy sector. Available at: https://docs.energyweb.org/ewc-ecosystem/energy-web-chain/system-architecture. Accessed: 2025-11-30.

Fan, C., Lin, C., Khazaei, H., and Musilek, P. (2022). Performance analysis of hyperledger besu in private blockchain. In *2022 IEEE International Conference on Decentralized Applications and Infrastructures (DAPPS)*, pages 64–73. IEEE. DOI: 10.1109/DAPPS55202.2022.00016.

Frost, J. (2020). Coefficient of variation in statistics. Available at:https://statisticsbyjim.com/basics/coefficient-variation/. Accessed: 2025-08-14.

Hafza, Y. (2025). What is geth? Available at: https://www.risein.com/blog/what-is-geth. Accessed: 2025-03-25.

Harish, V. and Sridevi, R. (2024). Enhancing split-join blockchain performance through load balancing. *International Journal of Electrical and Electronics Engineering*, 11. DOI: 10.14445/23488379/IJEEE-V11I7P110.

Holcombe, J. (2018). What is nginx? a basic look at what it is and how it works. Available at:https://kinsta.com/blog/what-is-nginx/. Accessed: 2025-09-24.

Hyperledger (2025). Introduction - Hyperledger Caliper. Available at:https://hyperledger-caliper.github.io/caliper/0.6.0/.

Islam, M. M., Merlec, M. M., and In, H. P. (2022). A comparative analysis of proof-of-authority consensus algorithms: Aura vs clique. In *2022 IEEE International Conference on Services Computing (SCC)*, pages 327–332. IEEE. DOI: 10.1109/SCC55611.2022.00054.

Kanai, S. (2022). What is test harness in software testing? Available at:https://www.headspin.io/blog/fundamentals-of-test-harness. Accessed: 2025-03-27.

Li, C., Huang, H., Zhao, Y., Peng, X., Yang, R., Zheng, Z., and Guo, S. (2022). Achieving scalability and load balance across blockchain shards for state sharding. In *2022 41st International Symposium on Reliable Distributed Systems (SRDS)*, pages 284–294. IEEE. DOI: 10.1109/SRDS55811.2022.00034.

Li, M., Wang, W., and Zhang, J. (2023). Lb-chain: Load-balanced and low-latency blockchain sharding via account migration. *IEEE Transactions on Parallel and Distributed Systems*, 34(10):2797–2810. DOI: 10.1109/TPDS.2023.3238343.

Microsoft (2020). Azure blockchain: Ethereum proof-of-authority consortium. Available at: https://azure.microsoft.com/en-us/blog/ethereum-proof-of-authority-on-azure/. Accessed: 2025-11-30.

Raab, F. (2019). System under test. In Sakr, S. and Zomaya, A. Y., editors, *Encyclopedia of Big Data Technologies*, pages 1663–1665. Springer International Publishing, Cham. DOI: 10.1007/978-3-319-77525-8$_1$24.

Samuel, C., Glock, S., Verdier, F., and Guitton-Ouhamou, P. (2021). Choice of ethereum clients for private blockchain: Assessment from proof of authority perspective. *ResearchGate*. DOI: 10.1109/icbc51069.2021.9461085.

Stewart (2025). Pearson's correlation coefficient: Definition, formula, and facts. Available at: https://www.britannica.com/topic/Pearsons-correlation-coefficient. Accessed: 2025-08-14.

Tareen, F. N., Alvi, A. N., Malik, A. A., Javed, M. A., Khan, M. B., Saudagar, A. K. J., Alkhathami, M., and Abul Hasanat, M. H. (2023). Efficient load balancing for blockchain-based healthcare system in smart cities. *Applied Sciences*, 13(4):2411. DOI: 10.3390/app13042411.

Yli-Huumo, J., Ko, D., Choi, S., Park, S., and Smolander, K. (2016). Where is current research on blockchain technology?—a systematic review. *PLOS ONE*, 11(10):e0163477. DOI: 10.1371/journal.pone.0163477.