

Fast&Securious: Improving the Performance of Permissioned Blockchain Applications

João Durão^{1,2}[0009–0006–2133–2276], Samih Eisa²[0000–0003–0972–4171], and
Miguel L. Pardal^{1,2}[0000–0003–2872–7300]

¹ Instituto Superior Técnico, Universidade de Lisboa, Portugal
`{joaopedrocostacorreia,miguel.pardal}@tecnico.ulisboa.pt`

² INESC-ID, Lisbon, Portugal
`samih.eisa@inesc-id.pt`

Abstract. Permissioned blockchains allow known organizations to share data and verify transactions without relying on a central operator. In agri-food supply chains, they can share traceability data across producers, distributors, and retailers. Hyperledger Fabric is a widely used platform for such systems, combining a shared ledger with multi-party transaction endorsement. However, its distributed and multilayered architecture makes performance difficult to assess. Existing approaches often study isolated components, require manual setup, and miss cross-layer effects. In this work, we present Fast&Securious, a testing framework for Fabric applications. We evaluate our approach by profiling FoodSteps, a food traceability system, to identify bottlenecks and validate optimizations. We found that requests involving Fabric were on average $9\times$ slower than database-only requests, motivating an architectural change to introduce a cache and reduce response times from 349 ms to 7.72 ms, a $45\times$ improvement.

Keywords: Hyperledger Fabric · Blockchain · Performance Evaluation · Reproducibility · Food Traceability

1 Introduction

Blockchain technology [12] has gained significant adoption across enterprise domains, including healthcare [14], supply chain management [15], and agriculture [13]. Among permissioned blockchain frameworks, Hyperledger Fabric (HLF) [1] has emerged as a leading platform for enterprise deployments due to its modular architecture, which allows components such as consensus to be configured for specific applications [7]. The participants in an HLF blockchain have X.509 identification and authentication. These organizations operate peers that maintain ledgers and execute chaincode. Transactions must be endorsed by multiple peers before submission to the ordering service, which is responsible for sequencing transactions into blocks.

1.1 Performance Challenges

Blockchain systems such as HLF are inherently complex, with numerous configurable parameters spanning multiple architectural layers. The impact of each configuration choice on overall system performance is often unclear, making it difficult to predict how changes will affect real-world deployments. Several factors make blockchain performance evaluation particularly challenging:

- *Complex interactions*: performance depends simultaneously on network delays, consensus protocol [7], resource allocation, and workload patterns;
- *Instrumentation challenge*: synchronizing timestamped data from multiple nodes and layers requires careful orchestration and tooling;
- *Cross-layer effects*: bottlenecks at one layer (e.g., network jitter) may manifest as performance degradation at another (e.g., application-level latency), making root cause analysis difficult without holistic monitoring;
- *Reproducibility issue*: distributed setups introduce environmental variability, making it hard to ensure consistent conditions across runs.

Enterprise applications often require low latency and high throughput. To verify these requirements we need to empirically measure the performance of the system. Without a structured and methodical approach, analysis risks addressing symptoms through isolated fixes rather than identifying root causes. Existing approaches to blockchain performance evaluation often fall short by focusing on isolated layers without considering real workloads, network conditions, or cross-layer dependencies.

1.2 Objectives

This work presents Fast&Securious, a reproducible and methodical testing framework designed to benchmark and optimize the performance of blockchain-based applications. In this paper, the framework is applied to FoodSteps³, a food traceability system built on HLF.

The objectives are: to enable *testing across every architectural layer*, from the blockchain consensus layer to the application-facing API layer; to provide cross-layer instrumentation that allows *attribution of latency* to specific system components; to perform automated and repeatable experiments that *empirically validate optimization hypotheses*.

2 Background and Related Work

A blockchain [12] is a decentralized, distributed digital ledger that securely records transactions across a network of computers. It consists of chronologically linked blocks, each containing transactions. Once added, blocks are computationally infeasible to alter, making the ledger immutable.

³ More information about FoodSteps available at <https://foodsteps.pt>

Blockchains can be classified by their access control model. In *public (permissionless) blockchains*, such as Bitcoin [10] and Ethereum [11], anyone can participate in transaction validation and consensus, and no single entity controls the ledger. In *private (permissioned) blockchains*, such as Hyperledger Fabric [1], only authorized entities can join the network and participate in consensus. This constraint enables faster consensus protocols and provides stronger access control guarantees, which are better suited for enterprise deployments.

2.1 Hyperledger Fabric

Hyperledger Fabric (HLF) [1] is a modular, permissioned blockchain framework developed under the Linux Foundation. Its architecture is organized around *organizations* that host one or more *peers*, which maintain the ledger and execute smart contracts (chaincode), while *orderer nodes* establish the total transaction order and batch transactions into blocks. A *Membership Service Provider (MSP)* manages participant identities through X.509 certificates, ensuring authentication and accountability. The ledger consists of an immutable blockchain log. On top of it, HLF provides *world state*, a key-value store database, containing the latest asset states, backed by either LevelDB or CouchDB [16], with the latter supporting JSON queries and index-based lookups.

HLF follows an *execute-order-validate model*: transactions are first simulated by a selected subset of endorsing peers, then ordered into a block, and finally validated and committed by all peers.

Client interaction is handled through the *Fabric Gateway*, embedded in each peer, which collects endorsements and submits transactions on the client's behalf. A write transaction proceeds through four phases: *Proposal*, where the client submits the transaction to a Gateway peer that forwards it to the required endorsing peers; *Endorsement*, where each endorsing peer simulates the transaction, produces a signed read-write set, and returns it to the Gateway; *Submit*, where the Gateway sends the endorsed transaction to the ordering service, which batches, orders, and creates a block; and *Commit*, where peers validate and commit the block to the ledger, after which the Gateway notifies the client once finalization is complete. For read-only queries, the Gateway routes the request to a single peer, which evaluates the chaincode against the world state and returns the result directly, bypassing ordering and commit.

2.2 Performance Evaluation of Blockchain Systems

Performance evaluation of blockchain systems can be approached *analytically* or *empirically* [2]. Analytical methods use mathematical models, such as queueing theory or Stochastic Petri Nets, to predict performance. Empirical methods measure system behavior through direct experimentation using workloads and monitoring tools.

Log-Based Monitoring Zheng et al. [3] proposed a real-time performance monitoring framework applicable to multiple blockchain platforms, including HLF. Rather than relying on monitoring based on remote procedure calls (RPC), which introduces significant overhead and can interfere with normal system execution, their framework adopts a log-based methodology: by parsing execution logs generated by validating peers, it passively captures performance data with low impact on the running system. They demonstrated that their approach introduces only 0.41% throughput reduction with 1,000 requests, compared to 17.67% for RPC-based monitoring on Ethereum.

Overall metrics include throughput, average response delay, transactions per CPU-second, memory-second, disk I/O, and network data, enabling fair cross-platform comparison across heterogeneous hardware configurations.

Stage-specific metrics break down the transaction lifecycle into peer discovery, transaction propagation, smart contract execution, state update, and consensus, helping identify bottlenecks within the Fabric architecture.

Their cross-platform evaluation found that permissioned systems substantially outperform proof-of-work systems. Fabric achieved 600 TPS and CITA 256 TPS, compared to 5.5 TPS for Ethereum and 3.9 TPS for Parity, primarily due to faster consensus mechanisms, though at the cost of higher network consumption.

Its main limitation is that it depends on the logs exposed by the blockchain implementation. Some internal events are not recorded, making certain performance behaviors only partially observable.

METHODA Rezabek et al. [4] introduced METHODA (Multilayer Environment and Toolchain for Holistic NetwOrk Design and Analysis), a framework for comprehensive, reproducible evaluation of large-scale distributed systems including blockchain platforms.

The framework implements a four-phase orchestration workflow, represented in Figure 1, and controlled through YAML configuration modules for Ansible. The first phase, *Install*, provisions the physical infrastructure and prepares the cluster environment. The second phase, *Setup*, deploys the software stack and monitoring infrastructure required to run experiments and collect data. The third phase, *Scenario*, executes the experiments according to predefined workload configurations, dispatching requests and collecting performance data in real time. The fourth and final phase, *Process*, analyzes the collected logs, reconstructs per-request latency timelines, and generates structured performance reports.

This design provides granular control across all deployment layers: from CPU core allocation and memory limits to network traffic shaping using `netem` and `qdisc` configurations. It supports precise time synchronization via PTP for nanosecond-precision measurements and automated fault injection for robustness testing.

METHODA was validated on four systems: Algorand, Ethereum 2.0, the FROST threshold signature scheme, and TEE-based deployments. For Algorand, it demonstrated that relay node core count (8 vs 16 cores) had minimal

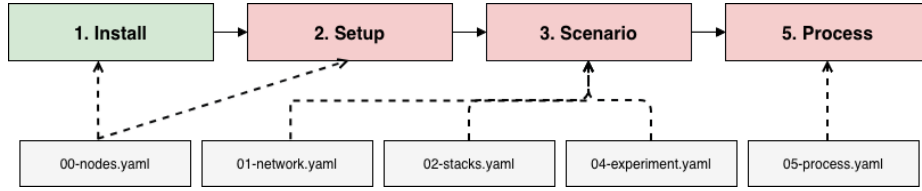


Fig. 1. METHODA’s 4-phase workflow (adapted from [4])

impact on throughput under workloads up to 6,000 TPS in their configuration. For Ethereum 2.0, it emulated Maximal Extractable Value (MEV) extraction scenarios, showing how builders prioritize higher gas-fee transactions within slots. The FROST with TEE integration showed negligible overhead, achieving average end-to-end latencies of approximately 40 ms for 32 nodes with 8,092-byte messages.

3 Fast&Securious Framework

Fast&Securious aims to be an automated and reproducible testing framework for HLF applications. It provides granular control over system parameters and helping identify performance bottlenecks and validate optimization hypotheses using evidence-based analysis.

3.1 Design Overview

Fast&Securious is built directly on METHODA’s four-phase (Install, Setup, Scenario, Process) orchestration workflow, adapted to an application-layer context. Where METHODA was originally designed and validated on network-level. The framework is composed of three categories of components. First, the *orchestration layer*: Ansible playbooks configured by YAML drive the entire experiment lifecycle, from monitoring stack deployment to workload dispatch and log collection. Second, the *monitoring infrastructure*: a PL Stack (Promtail [6] + Loki [5]) provides centralized, passive log collection from all system components with low runtime overhead. Third, the *workload and instrumentation layer*: a custom Load Generator issues requests to the system under test, and timestamp instrumentation embedded in the target application’s services enables per-stage latency decomposition without requiring changes to the blockchain itself.

An orchestrator component is responsible for controlling the overall workflow through all phases, coordinating which actions and workloads are executed in each phase, and ensuring that results are saved between runs.

3.2 Added Components

To support the four-phase workflow in an application-layer context, Fast&Securious introduces three new components: *Log Collection and Aggregation*, *Load Generation* and *Instrumentation*.

Log Collection and Aggregation A passive log collection and aggregation infrastructure deployed during the Setup phase. Promtail is used to automatically scrape logs from all running services with negligible runtime overhead. Loki centralizes and indexes the collected streams with persistent storage, making all experiment data queryable across runs.

Load Generator A configurable component that dispatches HTTP requests to the system under test, mimicking the behavior of real clients. It supports constant-rate generation for steady-state analysis, ramped load increases to identify saturation points, and burst patterns to evaluate behavior under sudden traffic spikes. Every request is assigned a unique identifier at dispatch time, enabling end-to-end correlation of all log entries generated by that request across all system layers.

Backend Instrumentation Timestamp markers embedded in the target application, at all layers, at key checkpoints throughout each request’s processing path. Each marker is emitted as a structured log entry carrying the request’s correlation identifier. During the Process phase, these markers are joined across all system layers to reconstruct complete per-request latency timelines and attribute observed latency to specific components, without requiring any changes to the blockchain network itself.

3.3 Targeted Metrics

With this methodology and its associated components, the framework focuses on three primary target metrics across all experiments: latency, throughput, and transaction success rate. *End-to-end response time with latency breakdown* is defined as the total time elapsed from request initiation at the Load Generator to response receipt, and decomposed into per-stage contributions using instrumentation timestamps, thereby enabling latency to be attributed to specific system components and layers. *Transaction throughput* is defined as the number of requests successfully processed per unit of time, measured both across the system as a whole and per request type, and used to capture system capacity under load while revealing saturation points. *Transaction success rate* is defined as the proportion of requests that complete successfully relative to those that time out, return errors, or are rejected, a metric that is particularly relevant for HLF-backed operations, where network instability or peer unavailability can affect reliability independently of latency.

4 Case Study: FoodSteps

To validate Fast&Securious, we applied the framework to FoodSteps, a food traceability system for the agro-food supply chain, designed to enhance food safety and transparency by tracking products through each stage of the supply

chain, providing tamper-proof information about the origin and handling of food. It uses HLF as the primary and authoritative storage for traceability data.

The FoodSteps system is represented in Figure 2 and composed by: App, Services, Blockchain, Database, Adapters, Certifiers and Devices.

The *App* is the user-facing modular interfaces for querying product traceability information. The *Services* offer RESTful APIs that mediate communication between apps and the blockchain backend, implementing business logic such as authentication, settings configuration, and product trace queries. The *Blockchain* is an HLF network (v2.5) that records all events regarding food traceability and executes smart contracts (chaincode) governing product data. The *Database* stores non-traceability data including user information, configurations, and service-related records. The *Adapters* are integration components connecting external enterprise systems and internet of things devices to the blockchain. Each adapter is implemented as an independent unit, translating source-specific data formats into a standard format. Data flows through a message queue to a Blockchain Writer service, which consumes messages and invokes the appropriate chaincode on the HLF network. Finally, the *Certifiers and Devices* provide certified attestations and real-time sensor data (e.g., temperature, humidity) as additional sources of information recorded on the blockchain.

At the Services layer requests from the App either resolve against CouchDB directly (for authentication and settings) or must contact the HLF peer to execute a chaincode query (for product trace data). The CouchDB also acts as a fallback mechanism for food traceability data: when the HLF network is unavailable, traceability records previously retrieved from the blockchain are persisted here, ensuring the system remains operational. These databases are distinct from the CouchDB instances used internally by HLF as world state storage.

The infrastructure necessary for the deployment of the system is described in Table 1.

Table 1. FoodSteps infrastructure overview.

Component	Deployed Services	Nodes
K3s Cluster [17]	3 namespaces: frontend for the user-facing application, couchdb for the CouchDB database cluster, and food-steps-backend for the services layer.	3
HLF Node	Hosts all HLF network components for a single-organization deployment: 2 endorsing peers, 1 orderer, and the Certificate Authority.	1

Applying Fast&Securious to FoodSteps requires instantiating each phase of the four-phase workflow as in Figure 1, within the constraints of the existing FoodSteps configuration, an overview of this application is represented in Figure 2.

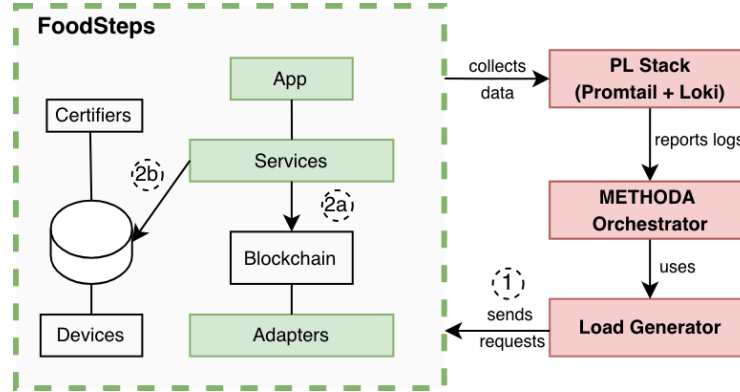


Fig. 2. Fast&Securious Design Overview

Install phase. Since FoodSteps is already deployed and operational, no tests were done in the production infrastructure as that would lead to down time and incomparable results, a replicated local deployment was used. No changes were made to the cluster topology or the HLF network configuration, the framework adapts to the production environment configurations, reusing its existing deployment settings throughout all experiments.

Setup phase. The full system was deployed in this phase. The HLF network deployment was done via Fablo [19] instantiating the components using Docker [22] containers, while the services layer and CouchDB cluster were deployed on K3s⁴. The Load Generator was deployed as a separate Docker container alongside the rest of the system, ensuring all components required for experimentation were in place before any scenario was executed.

To enable fine-grained latency decomposition, the FoodSteps backend services were instrumented with timestamp markers at key checkpoints throughout each request’s processing path. Timestamps were recorded using monotonic clocks, since each request is handled entirely by a single pod, there is no cross-machine synchronization requirement, and monotonic clocks eliminate the risk of clock skew caused by mid-request NTP [18] corrections. Each log entry carries a correlation identifier propagated from the Load Generator at request dispatch, allowing all entries belonging to a single end-to-end request to be joined during post-processing. Instrumentation covers every architectural layer of the Backend Services from entry into the Services layer to the point of CouchDB access, and the point of HLF chaincode invocation, making it possible to attribute total latency to each stage independently.

Scenario phase. Experiments were executed using the workloads and conditions defined in the 03-actions configuration file that specifies the workload profile

⁴ K3s is a lightweight Kubernetes distribution designed for resource-constrained, edge, and development environments. More information at <https://k3s.io/>

for each experiment as well as firewall rules applied mid-experiment to simulate different failure conditions. With this configuration, all planned experiments were run against the deployed system in a controlled and repeatable manner.

Process phase. After each scenario, logs were collected from both the Docker containers hosting the HLF network and the K3s pods running the services layer. Promtail then consumes these log files and applies experiment-specific parsing rules to extract the relevant entries, reconstructing end-to-end latency timelines and computing per-stage breakdowns. The extracted data is stored in a Loki instance, which is then queried through a Jupyter Notebook [21] to process and present the metrics in a human-readable form. Finally, Quarto [20] automatically generates a structured report for each experiment from these processed results.

5 Evaluation

This section presents the results obtained from FoodSteps using Fast&Securious through two experiments targeting the Services layer. Each experiment is organized around a research question, the workloads used, and the results obtained. A final discussion summarizes the findings across both experiments.

5.1 Experimental Deployment

The experiments were conducted in a replicated local deployment of FoodSteps, where four separate Virtual Machines running Ubuntu 24.04.4 LTS [24] were used, all hosted on a MacBook M1 Pro (16 GB RAM) using UTM virtualization [23]. The deployment comprised a K3s cluster running across three of the four virtual machines, each with 3 CPU cores and 40 GB of disk space, and a separate VM running the HLF network with 4 CPU cores and 40 GB of disk space. The Load Generator ran as an additional isolated process to avoid resource contention with the services under test.

5.2 Experiment 1: Cost of HLF Access at the Service Layer

Research Question. What is the latency cost of accessing the remote HLF peer for a chaincode query compared to serving a request directly from the in-cluster CouchDB ?

Setup. Four request types with distinct execution paths were profiled under identical conditions. `GET /v1/blockchain/trace` retrieves product traceability data directly from HLF by invoking HLF chaincode on a remote peer node. The remaining three endpoints, `POST /v1/auth/login`, `POST /v1/auth/logout`, and `GET /v1/users/settings`, serve their responses entirely from the in-cluster CouchDB.

Load. The Load Generator dispatched a constant-rate workload of 200 requests per endpoint type. End-to-end latency was measured from request dispatch to response receipt at the Load Generator; service-layer latency was derived from the instrumentation timestamps within the backend.

Results. Table 2 shows that *blockchain.trace* is the slowest request, averaging 235.21 ms. It also has the highest variability, with a 258.68 ms standard deviation. Figure 3 confirms this visually: HLF requests show large latency spikes, sometimes above 1 s, while the other requests remain lower and more stable.

Table 2. End-to-end latency — HLF vs CouchDB.

Request type	Avg (ms)	Std (ms)	× slower vs fastest
blockchain.trace (HLF — remote)	235.21	258.68	+940.75%
auth.login (CouchDB)	99.44	10.35	+340.00%
auth.logout (CouchDB)	22.60	5.10	baseline
getSettings (CouchDB)	34.70	6.19	+53.54%

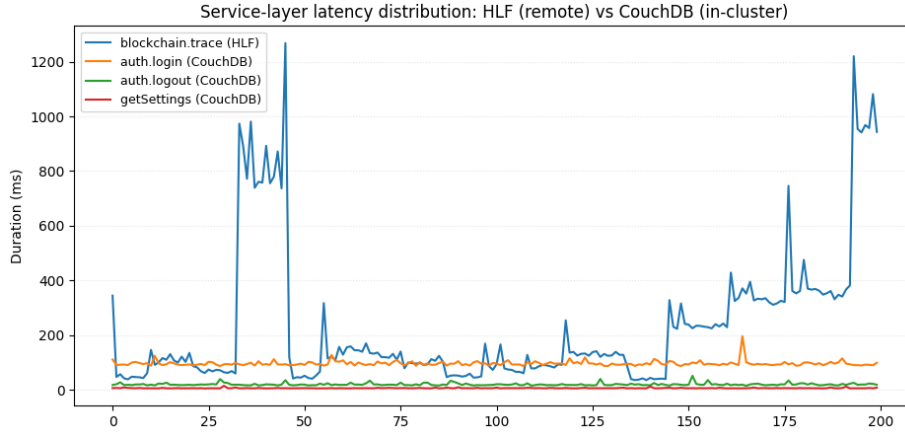


Fig. 3. Service-layer latency distribution: HLF (remote) vs CouchDB (in-cluster)

5.3 Experiment 2: CouchDB as Cache for HLF Query Results

Research Question. If HLF query results are cached in CouchDB on first access, how much latency can be saved on subsequent requests, and what overhead does the cache population step introduce ?

Data Immutability. The data stored on HLF consists of *EPCIS trace events* which are immutable records of supply chain events such as product harvesting, processing, and transportation. Once an event is committed to the ledger, it cannot be altered or deleted. A cached copy of these records in CouchDB is always consistent with the ledger, there is no scenario in which the authoritative data on HLF diverges from what was written to the cache on first access.

Setup. Three conditions were evaluated against `GET /v1/products/:id/trace`. *Cache Load* represents the first access to a product trace on a cold cache: the backend queries HLF, writes the result to CouchDB, and returns it to the client (*CouchDB miss* $\rightarrow$ *HLF* $\rightarrow$ *CouchDB write*). *Cache Hit* represents a subsequent request to the same product on a warm cache: the backend reads from CouchDB without contacting HLF (*CouchDB hit* $\rightarrow$ *return*). *Normal Trace* is the existing FoodSteps baseline where CouchDB acts as a fallback rather than a cache: HLF is always attempted first, then CouchDB is read as a secondary source (*HLF* $\rightarrow$ *CouchDB read*).

Load. The Load Generator first issued cache-load requests to populate the cache for a fixed set of product identifiers, then issued cache-hit requests to the same identifiers. Each condition used 200 requests with multiple repetitions.

Results. Table 3 shows that *Cache Hit* is much faster than both alternatives, averaging 5.52 ms with very low variability. Figure 4 shows that *Cache Hit* remains low, while *Cache Loading* and *Normal Trace* show large spikes and similar latency behavior.

Table 3. Service-layer latency — cache hit vs cache miss vs baseline.

Condition	Avg (ms)	Std (ms)	% slower vs Cache Hit
Cache Loading	258.45	281.00	+4585.29%
Cache Hit	5.52	1.44	baseline
Normal Trace	250.07	280.51	+4433.50%

5.4 Discussion

The two experiments together reveal the dominant cost structure of the FoodSteps Services layer. The HLF network interaction is the primary bottleneck: any request path that includes a chaincode invocation incurs latency an order of magnitude higher than equivalent in-cluster CouchDB accesses, with substantially higher variance due to the unpredictability of consensus timing, network conditions, and peer load. This is not a configuration issue but a structural consequence of the execute-order-validate architecture, where even a read-only chaincode evaluation requires a full network round-trip to a peer.

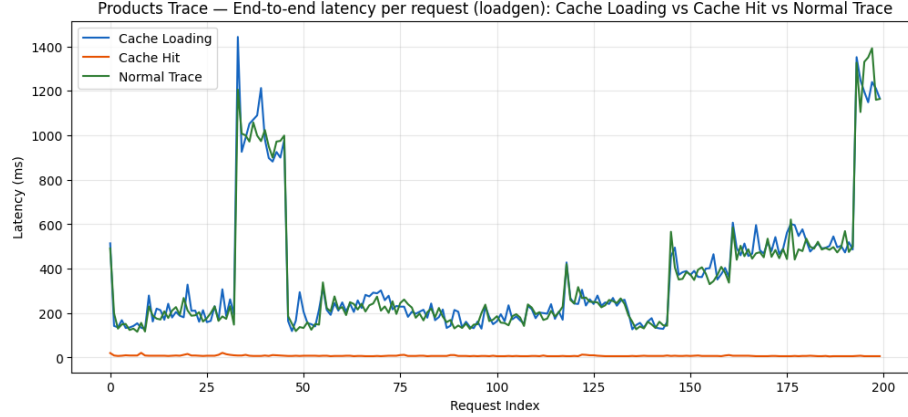


Fig. 4. End-to-end latency per request: Cache Loading vs Cache Hit vs Normal Trace

Experiment 1 further shows that even among CouchDB-only requests, latency differences are attributable to application logic rather than infrastructure, the number of database accesses per request directly governs response time. This demonstrates that the instrumentation methodology provides the granularity needed to distinguish infrastructure-driven from logic-driven latency, a distinction that coarser benchmarking tools cannot make.

Experiment 2 confirms that the caching optimization directly addresses the bottleneck identified in Experiment 1, but is only possible given the immutability of the data stored on the blockchain. By serving repeated reads from CouchDB rather than HLF, cache hits achieve a $45\times$ latency reduction with near-zero variance which is a critical property for user-facing applications. The fact that cache loading is comparable in cost to the baseline normal trace confirms that the optimization does not introduce new overhead, only eliminates the repeated cost of HLF access for data that has already been retrieved and stored locally.

6 Conclusion

This paper presented Fast&Securious, a methodical, reproducible, and automatic testing framework for Hyperledger Fabric applications. The framework addresses a gap in existing approaches: the inability to measure end-to-end performance of a complete blockchain application across all its architectural layers, in a controlled and repeatable manner. The framework was applied to FoodSteps, a traceability system for the agro-food supply chain, with two concrete experiments targeting the Services layer. The experiments validated both the ability to identify bottlenecks with per-stage precision and its effectiveness in targeted optimizations. Future work will extend the evaluation to the Adapters layer to benchmark data ingestion throughput and test the system’s crash-fault-tolerant consensus behavior by varying the number of available peers.

Supported by national funds by Fundação para a Ciência e a Tecnologia, I.P. (FCT) under projects UID/50021/2025 (DOI: <https://doi.org/10.54499/UID/50021/2025>) and UID/PRR/50021/2025 (DOI: <https://doi.org/10.54499/UID/PRR/50021/2025>) and by Blockchain.PT – Decentralize Portugal with Blockchain Agenda, (Project no 51), WP 1: Agriculture and Agri-food, Call no 02/C05-i01.01/2022, funded by the Portuguese Recovery and Resilience Program (PRR), The Portuguese Republic and The European Union (EU) under the framework of Next Generation EU Program.

References

1. E. Androulaki et al., “Hyperledger Fabric: A Distributed Operating System for Permissioned Blockchains,” in *Proceedings of the Thirteenth EuroSys Conference*, 2018, pp. 1–15. doi: 10.1145/3190508.3190538.
2. C. Fan, S. Ghaei, H. Khazaei, and P. Musilek, “Performance Evaluation of Blockchain Systems: A Systematic Survey,” *IEEE Access*, vol. 8, pp. 126927–126957, 2020. doi: 10.1109/ACCESS.2020.3006078.
3. P. Zheng et al., “A detailed and real-time performance monitoring framework for blockchain systems,” in *Proceedings of the 40th International Conference on Software Engineering: Software Engineering in Practice*, 2018, pp. 134–143. doi: 10.1145/3183519.3183546.
4. F. Rezabek et al., “Multilayer Environment and Toolchain for Holistic Network Design and Analysis,” in *2023*.
5. Grafana Labs, *Loki: Like Prometheus, but for Logs*, 2024. <https://grafana.com/oss/loki/>.
6. Grafana Labs, *Promtail: Agent for Loki*, 2024. <https://grafana.com/docs/loki/latest/send-data/promtail/>.
7. J. Jayabalan and J. N., “A Study on Distributed Consensus Protocols and Algorithms: The Backbone of Blockchain Networks,” in *2021 International Conference on Computer Communication and Informatics (ICCCI)*, 2021, pp. 1–10. doi: 10.1109/ICCCI50826.2021.9402318.
8. D. Huang, X. Ma, and S. Zhang, “Performance Analysis of the Raft Consensus Algorithm for Private Blockchains,” *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 50, no. 1, pp. 172–181, 2020. doi: 10.1109/TSMC.2019.2895471.
9. S. King and S. Nadal, “Ppcoin: Peer-to-peer crypto-currency with proof-of-stake,” *self-published paper, August*, vol. 19, no. 1, 2012.
10. S. Nakamoto, *Bitcoin: A Peer-to-Peer Electronic Cash System*, 2008. Available at <https://bitcoin.org/bitcoin.pdf>.
11. G. Wood, *Ethereum: A Secure Decentralised Generalised Transaction Ledger*, 2014. Available at <https://ethereum.github.io/yellowpaper/paper.pdf>.
12. M. Di Pierro, “What Is the Blockchain?,” *Computing in Science & Engineering*, vol. 19, no. 5, pp. 92–95, 2017. doi: 10.1109/MCSE.2017.3421554.
13. N. Patelli and M. Mandrioli, “Blockchain technology and traceability in the agri-food industry,” *Journal of Food Science*, vol. 85, no. 11, pp. 3670–3678, 2020. doi: 10.1111/1750-3841.15477.
14. A. Haleem, M. Javaid, R. P. Singh, R. Suman, and S. Rab, “Blockchain technology applications in healthcare: An overview,” *International Journal of Intelligent Networks*, vol. 2, pp. 130–139, 2021. doi: 10.1016/j.ijin.2021.09.005.

15. L. Gupta et al., “Blockchain Based Solution to Enhance Drug Supply Chain Management for Smart Pharmaceutical Industry,” in *2022 IEEE 10th Region 10 Humanitarian Technology Conference (R10-HTC)*, 2022, pp. 330–335. doi: 10.1109/R10-HTC54060.2022.9929913.
16. The Apache Software Foundation, *Apache CouchDB*. Available at <https://couchdb.apache.org/>.
17. Rancher Labs, *k3s: Lightweight Kubernetes*. Available at <https://k3s.io/>.
18. D. L. Mills, J. Martin, J. Burbank, and W. Kasch, *Network Time Protocol Version 4: Protocol and Algorithms Specification*, RFC 5905, 2010. Available at <https://www.rfc-editor.org/rfc/rfc5905>.
19. Hyperledger Labs, *Fablo: A Tool for Building Hyperledger Fabric Networks*. Available at <https://github.com/hyperledger-labs/fablo>.
20. Posit, *Quarto: Scientific and Technical Publishing System*. Available at <https://quarto.org/>.
21. Project Jupyter, *Jupyter Notebooks*. Available at <https://jupyter.org/>.
22. Docker, Inc., *Docker Documentation*. Available at <https://docs.docker.com/>.
23. UTM, *UTM Virtualization for macOS*. Available at <https://mac.getutm.app/>.
24. Canonical Ltd., *Ubuntu Operating System*. Available at <https://ubuntu.com/>.