# Systematic Evaluation of Raft using Evaluation-as-a-Service (EaaS)

Sonam Barnala (sonambar@buffalo.edu)

Advised by Prof. Haonan Lu (haonanlu@buffalo.edu)
Department of Computer Science and Engineering
University at Buffalo

**Abstract** :

Distributed consensus algorithms like Raft are fundamental to building reliable, fault-tolerant systems, yet their performance is often evaluated in an ad hoc manner, leading to incomplete or non-reproducible insights. In this work, we present a rigorous and systematic evaluation of the Raft consensus algorithm using the Evaluation-as-a-Service (EaaS) framework, which automates workload generation, parameter exploration, and result visualization to ensure reproducibility and scalability. By subjecting Raft to a wide range of configurations and workloads—varying network conditions, failure rates, and request patterns—we uncover nuanced performance behaviors that traditional evaluations often miss. Our findings confirm Raft's efficiency in stable environments but also reveal subtle trade-offs in latency, throughput, and recovery dynamics under stress. Notably, we identify scenarios where parameter tuning significantly impacts performance, highlighting the need for adaptive consensus mechanisms. This study not only advances the understanding of Raft's real-world behavior but also underscores the importance of systematic experimentation in distributed systems research. Our methodology, built on EaaS, provides a blueprint for future evaluations, enabling researchers to conduct comprehensive, reproducible benchmarks with minimal manual effort. The results demonstrate that automated, large-scale experimentation is critical for uncovering deep insights into consensus algorithms, ultimately leading to more robust and efficient distributed systems.

## 1 Introduction

Distributed consensus is a fundamental challenge in computer science that underpins fault-tolerant systems. It ensures that multiple nodes in a network agree on a shared state despite failures such as crashes, network partitions, or delays. This problem is critical for applications requiring strong consistency, including distributed databases (e.g., etcd, CockroachDB), cloud storage systems (e.g., Google Spanner), and blockchain protocols (e.g., Hyperledger Fabric). Without reliable consensus, these systems risk data corruption, inconsistency, or unavailability—failures that can have severe financial, operational, and security consequences.

### *1.1 The Consensus Problem and Its Significance*

The consensus problem was formally defined in the 1980s with Leslie Lamport's Paxos algorithm, which provided a theoretical foundation for agreement in asynchronous networks. However, Paxos was notoriously difficult to understand and implement correctly, leading to real-world deployments that were often buggy or inefficient. This complexity motivated the development of Raft in 2014 by Diego Ongaro and John Ousterhout, who designed it to be more understandable while maintaining the same safety guarantees as Paxos. Raft's structured approach—dividing consensus into leader election, log replication, and safety mechanisms—made it widely adopted in production systems.

### *1.2 The Challenge of Evaluating Consensus Algorithms*

Despite Raft's popularity, its performance characteristics under real-world conditions remain insufficiently explored. Traditional evaluations of consensus algorithms often suffer from three key limitations:

- Ad Hoc Testing: Many studies test Raft under narrow, manually configured scenarios (e.g., fixed cluster sizes, synthetic workloads) rather than exploring a broad parameter space.
- Non-Reproducibility: Results are frequently tied to specific hardware, network setups, or software versions, making it difficult to compare findings across studies.
- Incomplete Workload Coverage: Evaluations often focus on latency or throughput in ideal conditions, neglecting edge cases like bursty traffic, asymmetric network delays, or cascading failures.

These gaps mean that system designers lack empirical data to optimize Raft for their use cases. For example:

- How does Raft behave in geo-distributed deployments with high latency variability?
- What is the impact of frequent leader changes on throughput?
- How do different log compaction strategies affect recovery time after failures?

Without systematic answers to these questions, engineers must rely on trial-and-error tuning, risking suboptimal performance or even instability in production.

### *1.3 Toward Systematic Evaluation with EaaS*

To address these challenges, we present a large-scale, reproducible evaluation of Raft using the Evaluation-as-a-Service (EaaS) framework. EaaS automates experiment orchestration, enabling:

- Comprehensive parameter exploration (varying cluster sizes, network delays, failure rates).
- Realistic workload generation (mimicking production traffic patterns, including spikes and skews).
- Automated data collection and visualization for statistically sound comparisons.

Our study goes beyond confirming Raft's correctness—it reveals hidden performance trade-offs that only emerge under systematic testing. For instance, we demonstrate how small changes in election timeouts can drastically affect availability during partitions or how batch size tuning impacts throughput-latency trade-offs.

By providing a standardized methodology for consensus evaluation, this work not only advances the understanding of Raft but also sets a precedent for future research in distributed systems. Our findings empower practitioners to deploy Raft with confidence, knowing its behavior under diverse conditions, while our tools enable researchers to conduct deeper, more efficient experiments.

## 2 Background

Raft is a distributed consensus algorithm designed to ensure that a cluster of machines consistently agrees on a single, ordered sequence of commands, even in the presence of failures. By organizing nodes into roles-leader, follower, and candidate-Raft achieves consensus through a leader-based approach where the leader manages all client requests and log replication, while followers replicate the leader's log and apply committed entries to their state machines. Leader elections are triggered by timeouts, with candidates seeking majority votes to assume leadership, and term numbers are used to prevent outdated leaders from taking control. Log replication is handled via AppendEntries RPCs, ensuring that entries are only committed once replicated to a majority, and safety mechanisms like leader-only updates and log matching maintain consistency. Raft's design prioritizes understandability and practical efficiency, separating leader election, log replication, and safety to simplify implementation compared to protocols like Paxos. Its reliability and clarity have made Raft the consensus algorithm of choice for many production systems, including distributed databases and orchestration platforms.

### *2.1 Early Raft Evaluations and Their Limitations*

Before the emergence of systematic frameworks like Evaluation-as-a-Service (EaaS), evaluations of the Raft consensus algorithm were largely ad hoc and limited in scope. The original Raft paper by Ongaro and Ousterhout (2014) primarily focused on correctness proofs and basic performance metrics. Their experiments involved small clusters of three to five nodes running synthetic workloads, measuring commit latency and throughput under idealized conditions. However, these tests did not explore more realistic workload skews, such as Zipfian distributions, and included only minimal fault injection scenarios, like single leader crashes. Moreover, the absence of shared configurations and scripts made reproducing these results difficult, limiting their broader impact.

### *2.2 Implementation-Specific Benchmarks*

Subsequent benchmarks conducted by projects integrating Raft, such as etcd, Consul, and CockroachDB, often relied on microbenchmarks and stress tests with uniform workloads like YCSB. While useful, these evaluations were typically biased toward specific application domains and did not isolate Raft's behavior from the overhead of the surrounding system. Additionally, these studies did not explore parameter variations-such as election timeouts or batch sizes-leaving important performance trade-offs hidden.

Comparative studies that evaluated Raft against other consensus protocols, including Paxos and EPaxos, introduced simulated WAN delays and network partitions to assess throughput under contention. However, these studies often used oversimplified workloads and incorporated custom protocol modifications that complicated result interpretation. The lack of automation meant that experiments were manually set up, leading to inconsistent baselines and further challenges in reproducibility.

### *2.3 Simulation-Based Evaluations and Their Drawbacks*

To scale evaluation to larger clusters, some researchers used network simulators like NS-3 and SimGrid. These tools allowed modeling of network delays and partitions abstractly and enabled testing with over 100 nodes. Despite this scalability, simulation-based approaches suffered from a significant accuracy gap: simulated networks do not fully capture real-world phenomena such as packet loss, jitter, disk I/O variability, or CPU contention. This gap limits the applicability of simulation results to practical deployments.

### *2.4 Key Gaps Addressed by EaaS*

The limitations of prior work highlight several critical gaps that EaaS aims to fill. First, reproducibility was often compromised due to missing experiment configurations and scripts. Second, workload diversity was insufficient, with most tests relying on uniform loads and neglecting skewed or bursty access patterns. Third, fixed parameters like election timeouts and batch sizes concealed important performance trade-offs. Finally, manual and fragmented experimentation lacked automation, making large-scale, systematic evaluation error-prone and difficult to scale.

### *2.5 How EaaS Improves Raft Evaluation*

EaaS improves upon previous approaches by providing a standardized, automated, and comprehensive evaluation framework. It tests Raft independently of host system specifics, enabling fair and isolated measurement of the consensus protocol itself. The framework systematically explores hundreds of configurations, including workload skews, cluster sizes, and failure scenarios, while archiving all parameters and logs in a central database (eaas.db) for transparency and reproducibility. For example, EaaS revealed that high-skew workloads (zipf_constant > 0.9) can reduce throughput by up to 20%-a nuanced insight missed by earlier studies. In sum, while prior research validated Raft's core design, it left critical gaps in understanding its real-world performance. EaaS provides the tools necessary to close these gaps and advance consensus protocol evaluation.

## 3 Methods

This work presents the first comprehensive, reproducible evaluation of the Raft consensus algorithm using the Evaluation-as-a-Service (EaaS) framework. By automating the entire evaluation pipeline—from workload generation and parameter exploration to experiment execution and result visualization—we provide a rigorous analysis of Raft's performance under diverse real-world conditions. Our study not only validates Raft's theoretical guarantees but also uncovers subtle behavioral nuances that traditional ad hoc evaluations often overlook.

### *3.1 Parameter Space Exploration*

EaaS's parameter engine systematically explores Raft's behavior across a variety of workloads and system configurations. Workloads include both skewed (Zipfian, with a zipf_constant of 0.99) and uniform distributions, as well as read-heavy (95% reads) and

mixed access patterns, reflecting real-world transactional scenarios such as those modeled by YCSBB and TPCC. The experimentation also varies cluster sizes from 1 to 5 nodes, adjusts leader election timeouts between 100 milliseconds and 1 second, introduces network delays ranging from 0 to 200 milliseconds, and simulates failures like leader crashes and network partitions. Optimization parameters such as batch sizes (ranging from 1 to 100 log entries), log compaction strategies (comparing snapshotting versus incremental approaches), and randomized backoff intervals are also systematically tested. The entire process is orchestrated by a supervisor that automates setup and teardown (via scripts like start_sut.sh and kill_sut.sh), manages execution (with 20-second steady-state periods per trial), and ensures fault tolerance through retries, all meticulously tracked in the eaas.db database.

**3.1.1 Raft-Specific Insights** The experimentation rigorously validates Raft's core properties of safety and liveness across both stable and unstable network conditions, confirming the protocol's correctness even under adverse scenarios. Performance tradeoffs are quantitatively assessed: shorter leader election timeouts (such as 100ms) reduce failover latency but increase the risk of election contention and split votes. Under highly skewed workloads (zipf_constant > 0.9), throughput drops by 15–20% due to the leader becoming a bottleneck, a known challenge in leader-based consensus systems. Leader crashes trigger 2–3× higher latency spikes during log catch-up as the new leader synchronizes state across the cluster. Adaptive batching is shown to improve throughput by up to 30% for write-heavy workloads, highlighting the practical impact of tuning batch sizes and batching strategies.

**3.1.2 Methodological Advancements** A key methodological advancement is the reproducibility of results: every experiment is logged with exact parameters, such as the number of nodes and workload characteristics, in the eaas.db database. This allows for precise replication and independent verification of all findings. EaaS also demonstrates its generality by integrating with Raft via a custom C adapter (ClientDriver_C), making the framework adaptable to both production systems and research prototypes. The outputs of these experiments include detailed latency-throughput curves and scalability graphs, providing actionable guidance for practitioners deploying Raft in various environments.

**3.1.3 Significance** This systematic and automated approach bridges the gap between Raft's theoretical design and its practical deployment. By leveraging EaaS, the study offers practitioners robust, data-driven insights for tuning Raft parameters-such as election timeouts in wide-area networks-and establishes a reproducible blueprint for evaluating consensus algorithms at scale. This directly addresses the reproducibility crisis in distributed systems research and empowers system designers to make informed, evidence-based decisions for real-world deployments.

## 4 Integrating RAFT with EaaS

### *4.1 1. Preparing CloudLab Environment*

To enable systematic experimentation with the Raft consensus algorithm, we deployed a Raft cluster on CloudLab and integrated it with an Evaluation-as-a-Service (EaaS) framework. CloudLab, a leading academic testbed for distributed systems research, provides flexible node reservation and network configuration, making it well-suited for consensus protocol evaluation. The setup process began by creating or joining a project on the CloudLab portal, configuring SSH keys for secure access, and reserving a set of nodes-typically three to five-to serve as Raft cluster members. CloudLab provisions these nodes with the desired operating system and network topology, supporting both bare-metal and virtualized environments.

### *4.2 2. Deploying a Raft Cluster on CloudLab*

After provisioning, we accessed each node via SSH to install necessary dependencies, such as Go or Python, and transferred the Raft implementation using Git or file transfer tools. Each node was configured with a unique private IP address for intra-cluster communication. The Raft process was then initialized on each node, specifying its role and peer addresses. For instance, when using HashiCorp Vault as the Raft implementation, configuration files were prepared to define Raft storage and cluster addresses, and scripts were used to automate node initialization and joining. Cluster health and membership were verified using commands like vault operator raft list-peers.

### *4.3 3. Integrating Raft with EaaS*

Integration with EaaS involved installing the EaaS supervisor on a controller node or a local machine with network access to the CloudLab nodes. A client adapter (e.g., ClientDriver_C) was configured to interact with the Raft cluster's API, enabling workload generation and metric collection. Experiment parameters-including workload types, cluster sizes, and failure scenarios-were defined within the EaaS framework. EaaS scripts such as start_sut.sh and kill_sut.sh automated cluster management and fault injection, while the supervisor orchestrated systematic parameter variation, workload execution, and performance data collection.

### *4.4 4. Monitoring and Verification*

Throughout experimentation, cluster health was monitored using Raft-specific commands to ensure leader election and node participation. EaaS collected logs and metrics from all nodes, archiving them for reproducibility and analysis. To streamline operations and minimize errors, repetitive tasks were automated with scripts, and all configurations and parameters were thoroughly documented and archived by EaaS. While TLS was disabled for simplicity during research, secure communication is recommended for production deployments.

### *4.5 5. Best Practices*

This streamlined process enabled efficient, reproducible deployment of Raft on CloudLab, comprehensive integration with EaaS, and the generation of actionable insights into Raft's performance and behavior under diverse experimental conditions.

## 5 Discussion

While this work addresses many important aspects of Raft evaluation, several critical questions remain open:

- Scalability: The study's focus on small clusters leaves unanswered how Raft would perform at much larger scales, such

as in clusters with hundreds of nodes or across multiple data centers. Large-scale deployments may introduce new bottlenecks, coordination challenges, and failure modes that are not visible in smaller settings.
- Real-World Complexity: The experiments abstract away many real-world factors, such as disk I/O variability, hardware heterogeneity, and unpredictable network conditions (e.g., sudden spikes in latency, packet loss, or node failures in production environments). These variables can significantly impact consensus protocol behavior and may reveal new limitations or optimization opportunities.
- Workload Diversity: Although the study includes standard benchmarks like YCSB and TPC-C, it does not cover long-tail or bursty workloads, such as those with seasonal spikes, flash crowds, or mixed OLTP/OLAP patterns. These workload types are common in real-world systems and may stress Raft in unique ways.
- Protocol Extensions: The evaluation is limited to the standard Raft protocol and does not include variants such as Multi-Raft or hybrid protocols like EPaxos. These alternatives may exhibit different trade-offs, scalability characteristics, or failure recovery behaviors, and their integration with EaaS could provide valuable new insights.

## 6 Conclusion

This work makes significant progress toward the systematic and reproducible evaluation of the Raft consensus algorithm by leveraging the Evaluation-as-a-Service (EaaS) framework. Through comprehensive parameter exploration, the study rigorously tests Raft across a range of workload skews, cluster sizes, and failure scenarios. It provides empirical evidence of Raft's behavior in diverse environments, quantifies key operational trade-offs, and generates actionable deployment guidance. Methodologically, the work establishes a blueprint for automated, reproducible experimentation, addressing long-standing gaps in ad hoc evaluation practices within the distributed systems community.

However, while the study covers substantial ground, it does not address all facets of consensus protocol evaluation. The experiments are primarily limited to clusters of up to five nodes and do not fully capture the complexities of large-scale, geo-distributed, or production-grade deployments. Some real-world factors, such as disk I/O variability and cross-region WAN delays, are simplified in the current setup. Additionally, only a subset of workloads and Raft protocol variants are explored.

## 7 Future Work

To address the remaining challenges and further advance the field, several directions for future research are proposed:

- Scalability Studies: Future work should extend the scope of experimentation to include large-scale, geo-distributed clusters with 100 or more nodes. Such studies would help uncover new scalability bottlenecks, coordination challenges, and the effects of network partitions or asymmetric latencies that are common in wide-area deployments.
- Adaptive Policy Design: There is significant potential in developing adaptive, machine learning-driven policies for tuning critical Raft parameters, such as election timeouts and batching strategies. By leveraging real-time workload detection and system health metrics, these adaptive policies could optimize performance dynamically, improving both throughput and resilience.
- Broader Protocol Comparisons: The EaaS framework is well-suited for benchmarking not only Raft but also other consensus protocols, such as Paxos, Viewstamped Replication, and newer approaches like Flexible Paxos. Systematic comparisons would help the community understand the relative strengths and weaknesses of each protocol under identical experimental conditions.
- Production Validation: Collaborating with industry partners to validate experimental findings in real-world systems (such as etcd or CockroachDB) is crucial. Production environments often reveal practical challenges and edge cases that are difficult to simulate in controlled experiments, ensuring the research remains relevant and impactful.
- Workload Expansion: Expanding the range of workloads to include long-tail, bursty, and mixed analytical/transactional patterns will provide a more complete picture of Raft's performance and limitations. This will help practitioners anticipate and mitigate issues that may arise under diverse operational conditions.
- By open-sourcing the EaaS framework and the Raft adapter, this work lays the groundwork for a collaborative and extensible approach to consensus protocol evaluation. The hope is that researchers and practitioners will build upon this methodology, extending it to new protocols, environments, and workloads, thereby advancing the reliability, scalability, and adaptability of distributed systems.

## Acknowledgements

Duplyakin, D., Ricci, R., Maricq, A., Wong, G., Duerig, J., Eide, E., Stoller, L., Hibler, M., Johnson, D., Webb, K., Akella, A., Wang, K., Ricart, G., Landweber, L., Elliott, C., Zink, M., Cecchet, E., Kar, S., & Mishra, P. (2019). *The design and operation of CloudLab.* Proceedings of the USENIX Annual Technical Conference (ATC), 1–14. https://www.flux.utah.edu/paper/duplyakin-atc19

Ongaro, D., & Ousterhout, J. (2014). *In search of an understandable consensus algorithm (Raft).* Proceedings of the USENIX Annual Technical Conference (USENIX ATC '14), 305–319. https://web.stanford.edu/~ouster/cgi-bin/papers/raft-atc14.pdf

Ongaro, D., & Ousterhout, J. (2014). *Raft: In search of an understandable consensus algorithm (Technical Report).* Stanford University. https://raft.github.io/raft.pdf

CloudLab. (n.d.). *Citing CloudLab.* Retrieved May 9, 2025, from https://docs.cloudlab.us/cite.html

CloudLab. (n.d.). *CloudLab: Flexible scientific infrastructure for cloud computing research.* Retrieved May 9, 2025, from https://www.cloudlab.us