NextGen Scientific Publication
American International Journal of Computer Science and Technology

Volume 5 Issue 5, Pg. No. 39-46, AIJCST-V5I5P104, 2023
https://doi.org/10.63282/3117-5481/AIJCST-V5I5P104

*Original Article*

# Performance Benchmarking Methodologies for Real-Time Integration Middleware Platforms

***Suman Neela**

*Visvesvaraya Technological University, India.*

## Abstract:

*Real-time integration middleware sits at the core of how modern enterprises move data, trigger actions, and keep distributed systems in sync. Platforms like Apache Kafka, RabbitMQ, IBM MQ, and Apache Pulsar carry the weight of mission-critical operations—from payment processing and supply chain coordination to IoT data collection and distributed analytics. Yet, despite how much it depends on them, the methods used to benchmark their performance are still surprisingly immature. Most current ways of testing these systems—like checking the infrastructure, running application load tests, or assessments done by vendors—fail to capture the important behaviors that really affect their performance in real situations: how well they handle sudden spikes in traffic, how they deal with pressure, and if keeping messages safe slows things down too much. This article covers the theoretical basis for middleware performance evaluation, maps out where current practice falls short, and offers a detailed look at the Middleware-Aware Performance Benchmarking Framework (MAP-Bench). MAP-Bench combines a set of standard measurements, models for business workloads, tests for handling faults, and a more advanced way to evaluate architecture into one complete method. The practical value extends to platform selection, capacity planning, SLA design, and performance tuning. The goal is to move middleware benchmarking away from narrow load testing and toward something that actually reflects how these systems perform when it counts.*

## Keywords:

*Middleware Benchmarking, Real-Time Integration, Event-Driven Architecture, Map-Bench, Latency Measurement, Throughput Evaluation, Fault-Aware Testing, Enterprise Workload Modeling*

## Article History:

***Received: 22.07.2023***

***Revised: 26.08.2023***

***Accepted: 01.09.2023***

***Published: 10.09.2023***

## 1. Introduction

### 1.1. Contextual Background

Modern enterprises run on event flows. Every order confirmation, sensor reading, and financial transaction moves through some form of integration middleware before it reaches its destination. These platforms form the connective layer between cloud services, on-premise infrastructure, microservices, and external partners, and that layer has to perform reliably regardless of load, topology, or failure conditions. Early benchmark work, including the SPECjms2007 effort for message-oriented middleware, made it dear that measuring throughput alone tells an incomplete story; workload-specific behavior under realistic enterprise conditions matters far more [1].

Apache Kafka, RabbitMQ, IBM MQ, and Apache Pulsar are deployed across industries precisely because they promise different things, different latency profiles, different consistency guarantees, and different scaling behaviors. A survey of distributed message broker queues confirms that these platforms diverge significantly in their underlying design assumptions, which makes platform-neutral benchmarking not just useful but necessary for anyone making an informed adoption decision [2][3]. As cloud-native architectures

continue to replace monolithic systems, the performance characteristics of the middleware layer ripples outward, shaping how responsive applications feel, whether SLAs get met, and what infrastructure actually costs.

Sub-second latency is not a nice-to-have for most operational workflows anymore. A slow middleware layer does not just cause delays; it cascades. Downstream systems back up. The technical and commercial consequences are real, and they demand a more disciplined evaluation approach than what most organizations currently apply.

## 2. Problem Statement and Research Gap

### 2.1. Core Problem

Three types of benchmarking dominate current practice. Infrastructure-level tests measure CPU load, memory consumption, and network throughput. Application-level load tests simulate user requests against APIs and web services. Vendor-published performance reports highlight strengths under carefully chosen conditions. None of these get at what makes middleware behave the way it does in production. In edge-cloud and IoT environments especially, publish-subscribe middleware operates across wide-area networks with variable latency and topology shifts that static benchmarks simply cannot replicate [4].

The gaps are not minor. Conventional tests rarely examine how a platform degrades when producers outrun consumers, how message ordering holds under scale, or how a broker recovers after a node goes down. Continuous benchmarking methods embedded in software build pipelines come closer to operational reality, but their application to middleware-specific evaluation is still narrow and largely platform-dependent [5]. The behavioral dimensions that define middleware fitness, backpressure handling, replay performance, cross-region replication fidelity, and durability overhead are consistently left out.

### 2.2. Research Gap

Database systems have TPC benchmarks. Network performance has well-developed analytical models. Middleware has neither. Recent work on benchmarking the scalability of stream processing frameworks deployed as microservices in cloud environments points in the right direction, but these efforts are fragmented, useful within a specific context, and not generalizable across platforms or deployment models [6]. What is missing is a methodology that applies the same rigor to middleware that TPC applies to databases: standardized metrics, controlled workloads, reproducible procedures, and results that mean something across different platforms and environments.


**Figure 1. Architectural Design, Scaling Challenges, and Event-Driven Integration in Distributed Systems**

## 3. Research Purpose and Scope

### 3.1. Purpose

MAP-Bench sets out to build that methodology. The goal is to create a benchmarking method that clearly defines performance measurements so they can be repeated by others, accurately represents real business workloads, tests performance from regular use to recovery from problems, and provides results that enable fair comparisons between different platforms. The need for this work increases as new systems that manage delays are developed for important areas like 6G network slicing and real-time service delivery, where the performance of middleware can directly impact how well the system works.

The framework is not built to favor any platform. It is built to give practitioners the information they need to make decisions that hold up in production.

### 3.2. Scope

MAP-Bench covers latency and throughput under varied load conditions, backpressure and congestion dynamics, message persistence overhead, horizontal and vertical scalability, degradation under induced faults, recovery-time effects on performance, and behavior across multi-cloud and hybrid deployment topologies. Vendor neutrality is not a secondary consideration; it is a design constraint that shapes every component of the methodology.

## 4. Theoretical Foundation

### 4.1. Performance Metrics Theory

Distributed systems evaluation has long tracked latency, throughput, resource utilization, and scalability. These are necessary dimensions, but they are not sufficient for middleware. Phase-aware performance modeling for cloud applications helps us understand how middleware systems change their behavior during startup compared to when they are running steadily. A broker processing a burst of messages behaves differently from the same broker at steady-state throughput, and a benchmark that only captures one of those states is only telling part of the story.

Queue accumulation rates, burst elasticity, ordering constraints under replication, and consistency trade-offs all shape how middleware actually performs. Formal performance modeling and stochastic analysis of message-oriented event-driven systems provide the theoretical tools to quantify these behaviors in ways that go beyond empirical observation alone [9]. MAP-Bench draws on these traditions to ground its metric definitions in something more rigorous than ad hoc measurement.

### 4.2. Real-Time Systems Performance Requirements

Batch processing tolerates latency. Real-time integration does not. The difference is not just a matter of degree; it changes what needs to be measured and how. Efficient operator placement work in distributed stream processing shows that deployment topology decisions have direct, measurable effects on end-to-end latency and throughput, which means topology cannot be treated as a constant in any meaningful benchmark [10].

Average latency figures are not enough. Cloud infrastructure introduces measurement noise that can skew results in ways that only become visible when tail latency distributions are examined carefully. Microbenchmarking work conducted in cloud environments has shown that this noise is significant enough to invalidate comparisons if experimental controls are not applied [11]. MAP-Bench builds those controls in from the start, measuring ninety-fifth and ninety-ninth percentile latency alongside mean values and tracking variance over time rather than treating performance as a single static figure.

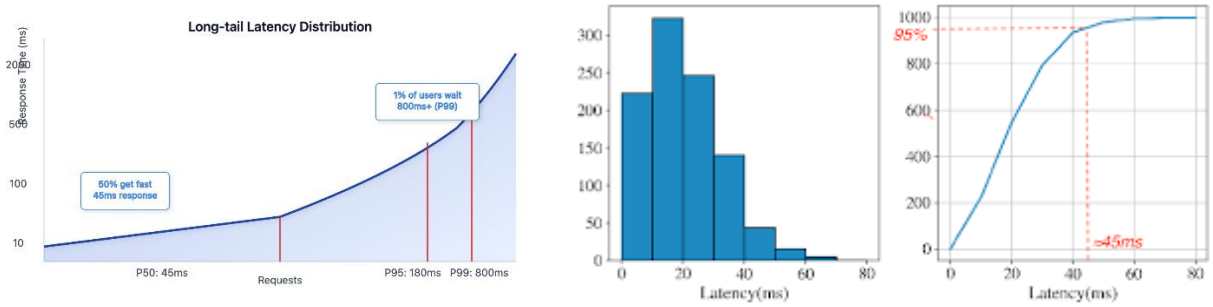


**Figure 2. Analysis of Long-tail Latency and Response Time Distributions**

## 5. Proposed Middleware-Aware Performance Benchmarking Framework (MAP-Bench)

MAP-Bench is built around five integrated components. Each one targets a distinct dimension of middleware performance. Figure 1 shows how they connect, from workload input through metric collection to architectural output.

### 5.1. Standardized Performance Metric Taxonomy

Seven core performance indicators form the backbone of the taxonomy. End-to-end latency measures the time between message generation and acknowledgment. Throughput measures the number of messages processed in a given time. Jitter is the variation between when different messages are received, while backpressure sensitivity is the behavior when producers constantly outpace consumers.

Durability latency is overhead incurred for persistence writes, and the scalability index is an improvement in throughput per resource. Recovery performance tracks how latency and throughput behave during and after a fault event.

Each indicator comes with formal measurement definitions, sampling protocols, and statistical criteria. Measurement without those definitions produces numbers that cannot be compared across platforms. Work on advancing real-time data processing and middleware integration in enterprise architectures confirms that meaningful evaluation requires explicitly addressing the interaction between throughput capacity and integration latency under concurrent multi-system load [12]. Table 1 presents the five primary indicators selected for structured taxonomy representation within MAP-Bench.

**Table 1. Core Performance Metric Taxonomy in MAP-Bench [12]**

| Performance Metric | Operational Definition | Evaluation Focus |
|---|---|---|
| End-to-End Latency | Elapsed time from message production to confirmed consumer receipt | Responsiveness under steady-state and peak load conditions |
| Throughput | Volume of messages successfully processed per unit time | Capacity ceiling and sustained delivery rate |
| Jitter | Statistical variability in inter-message delivery intervals | Predictability and timing consistency under variable load |
| Backpressure Sensitivity | Measurable system response to sustained producer-consumer imbalance | Degradation behavior and flow-control effectiveness |
| Recovery Performance | Throughput and latency profile during and following fault recovery | Resilience and continuity after node or network failure events |

### 5.2. Enterprise Workload Modeling Framework

Five workload categories cover the range of integration scenarios enterprises actually face. Likewise, transactional workloads consist of small, frequent messages used for payment processing and order management, while streaming workloads handle ongoing streams of large amounts of data for collecting telemetry and aggregating logs. Finally, burst workloads capture traffic spikes associated with flash events or reconnections. Hybrid workloads run transactional and streaming workloads in the same database. Geo-distributed workloads test replication fidelity and latency across regions.

Synthetic generators are used alongside anonymized real-world integration data to create test situations that mimic real production differences. The experimental design draws on established principles from systems benchmarking for scientists and engineers, particularly around repeatability and statistical validity, which are non-negotiable requirements for results that hold up to scrutiny [13]. Table 2 maps each category to its defining characteristics and representative enterprise context.

**Table 2. Enterprise Workload Classification Framework in MAP-Bench [13]**

| Workload Category | Defining Characteristics | Representative Enterprise Use Case |
|---|---|---|
| Transactional | Small payload size, high message frequency, low tolerance for loss | Payment processing, order management, and inventory update pipelines |
| Streaming | Continuous high-volume data flow, sustained throughput requirements | IoT telemetry ingestion, log aggregation, and real-time analytics feeds |
| Burst | Sudden traffic spikes above baseline, short-duration intensity surges | Flash sale events, system reconnection floods, and scheduled batch triggers |
| Hybrid | Concurrent transactional and streaming patterns with mixed payload profiles | Enterprise integration hubs handling multi-channel event flows simultaneously |
| Geo-Distributed | Cross-region replication with variable inter-datacenter latency constraints | Global supply chain synchronization and multinational financial transaction routing |

### 5.3. Stress and Fault-Aware Benchmarking Model

Most benchmarks test what happens when everything is working, but MAP-Bench tests what happens when it is not. In distributed AI-empowered edge-cloud environments, the interaction between middleware reliability and fault propagation across compute layers is a performance dimension that idealized benchmarks miss entirely [14]. A platform that performs well under stable conditions but degrades badly under partial failures is not a suitable choice for mission-critical enterprise workloads, and conventional benchmarks would never flag that risk.

MAP-Bench introduces specific problems like network delays, single node failures, slow disk input/output, cloud region outages, and message replay, both on their own and together. Compound fault scenarios are particularly important because production failures rarely arrive one at a time. This testing model captures the realistic degradation envelope of a middleware platform, not just its best-case performance.

### 5.4. Deployment Topology Benchmarking

A single-node broker behaves differently from a clustered deployment. A clustered deployment behaves differently from a multi-region configuration. Containerized orchestration environments introduce their variables. Any benchmark that ignores these differences is not comparing platforms; it is comparing configurations. The growth of serverless and Function-as-a-Service models complicates this issue further, and benchmark tools created for serverless environments provide helpful methods for assessing these situations.

MAP-Bench standardizes topology descriptors across single-node, clustered, multi-region, hybrid cloud, and containerized setups. That standardization is what makes cross-platform comparison meaningful; it ensures that observed performance differences reflect platform characteristics rather than environmental noise.

### 5.5. Holistic Architectural Evaluation Layer

Raw metrics tell you what happened. Architectural evaluation tells you why and what to do about it. This part of MAP-Bench goes further than just gathering data; it looks at where problems occur, how well resources are used for each unit of work, the balance between cost and performance, and how well the system's tail-latency-aware autoscaling works on distributed event queues. It makes the case that resource allocation decisions must be grounded in empirical tail latency distributions; averages are not enough [16].

This type of analysis is what separates MAP-Bench from a conventional load-testing tool. The output is not a throughput number. It is an architectural assessment, something an engineer can act on when selecting a platform, designing a topology, or debugging a performance problem.

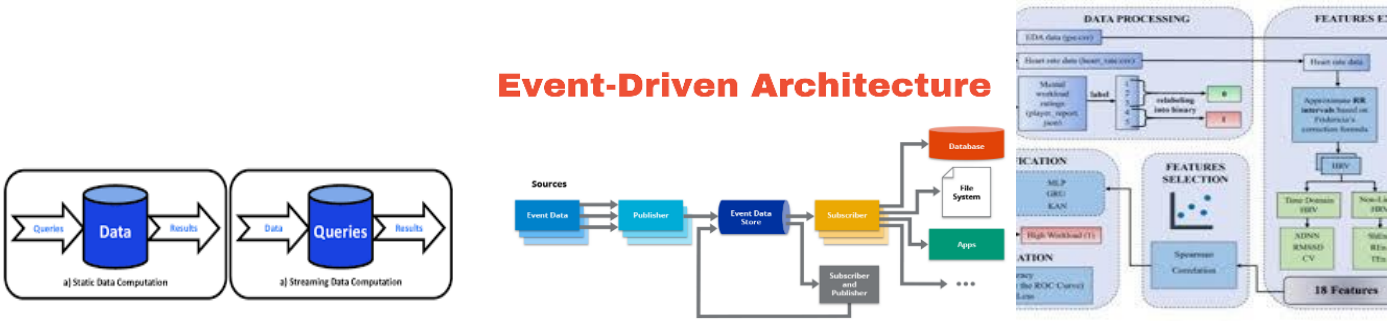


**Figure 3. Architectural Framework for Real-time Data Processing and Predictive Modeling**

## 6. Methodology

### 6.1. Metric Definition and Formalization

Every metric in the MAP-Bench taxonomy is formally defined. Measurement formulas, time windows, statistical methods, and sampling procedures are specified explicitly for each one. Without that level of definition, two independent runs of the same benchmark can produce different results for reasons that have nothing to do with platform behavior, and cross-platform comparison becomes meaningless.

### 6.2. Workload Design and Simulation

Workload generators cover a wide operating range, incorporating mixed payload sizes, randomized burst spikes, and cross-region latency variance. The design targets both steady-state and transient conditions, because a platform that handles sustained throughput well may still struggle when load patterns shift suddenly.

### 6.3. Cross-Platform Benchmark Execution

All platforms are tested on identical hardware, with standardized network configurations and consistent software versions across every run. Controlling these variables is the only way to isolate platform-specific performance from environmental differences that would otherwise make comparisons unreliable.

### 6.4. Performance Data Collection and Analysis

Data is turned into latency distribution histograms, throughput stability curves, resource utilization heatmaps, and degradation slope charts. These formats expose patterns that aggregate statistics hide; a platform with acceptable mean latency may still show alarming tail behavior that only becomes visible in a distribution view.

### 6.5. Comparative Evaluation

Platforms are evaluated on efficiency under stress, resilience during failure, scalability linearity, and performance predictability under increasing loads. This approach replaces throughput leaderboards with multidimensional profiles that reflect how a platform actually behaves in production deployment.

## 7. Comparative Analysis

The differences between conventional benchmarking and MAP-Bench run deeper than methodology. Conventional approaches focus on application or infrastructure layers; MAP-Bench targets middleware behavior specifically. Conventional workload models are generic; MAP-Bench uses enterprise-grade patterns. Conventional benchmarks often skip stress testing entirely; MAP-Bench treats fault injection as a core evaluation step. The architectural insight conventional tools produce is thin; MAP-Bench generates structured intelligence that engineers can act on. Cross-platform comparability is weak in standard approaches; MAP-Bench standardizes it. Table 3 lays out these differences across five key dimensions.

**Table 3. Comparative Analysis of Conventional and MAP-Bench Evaluation Approaches [1][6]**

| Evaluation Dimension | Conventional Benchmarking | MAP-Bench Methodology |
|---|---|---|
| Scope of Focus | Application-layer or infrastructure-layer performance in isolation | Middleware-tier behavior, including event routing, queuing, and replication |
| Workload Modeling | Generic synthetic loads without enterprise integration pattern representation | Enterprise workload categories including transactional, streaming, burst, and geo-distributed |
| Stress and Fault Testing | Optional or absent; predominantly conducted under stable, ideal conditions | Integrated fault injection, including node failure, network degradation, and region unavailability |
| Performance Metric Coverage | Limited to throughput and basic latency averages without tail distribution analysis | A comprehensive taxonomy covering jitter, backpressure sensitivity, durability latency, and recovery performance |
| Architectural Insight | Low outputs isolated throughput figures without structural interpretation | High; generates cost-performance ratios, scaling linearity assessments, and bottleneck identification |

## 8. Practical Applications

### 8.1. Middleware Platform Selection

Vendor benchmark reports are marketing material. MAP-Bench gives procurement teams a standardized, topology-controlled basis for comparing platforms that does not depend on the conditions a vendor chose to highlight. The difference in outcomes between choosing a platform that fits the actual workload versus one that looks good on a spec sheet can be substantial.

### 8.2. Capacity Planning

Performance curves from MAP-Bench translate directly into infrastructure-provisioning decisions. Scalability index data shows where throughput growth stops being linear. Burst workload profiles reveal how much headroom a deployment needs to absorb peak events without degrading. These are not estimates; they are empirically derived figures.

### 8.3. SLA Definition and Enforcement

Realistic SLA commitments need empirical foundations. A latency threshold set without benchmark data behind it is a guess. MAP-Bench outputs, especially ninety-ninth percentile latency under fault conditions, give operations teams something concrete to anchor SLA terms to, covering latency thresholds, throughput guarantees, and recovery expectations.

### 8.4. Performance Optimization

Architects working with MAP-Bench results can trace performance problems back to specific causes. Partition strategies can be adjusted based on throughput data. Replication factors can be tuned against durability latency figures. Configuration parameters can be modified with known effects rather than trial-and-error. The architectural evaluation layer is what makes this possible.

## 9. Expected Contributions

MAP-Bench contributes a standardized middleware performance taxonomy, enterprise workload modeling techniques grounded in realistic traffic patterns, fault-aware benchmarking protocols, cross-platform evaluation frameworks with controlled comparability, and empirical performance characterization of real-time integration systems across topology types.

Academically, this work builds a structured evaluation discipline for middleware that has no current equivalent, analogous to what TPC provides for databases but tailored to the behavioral complexity of event-driven messaging platforms. For practitioners, the payoff is better platform decisions, more accurate capacity plans, and SLA commitments that reflect actual system behavior.

## 10. Limitations and Future Work

Synthetic workloads, however carefully designed, cannot fully replicate every proprietary enterprise data pattern. Large-scale deployment benchmarking demands significant infrastructure resources, which may put thorough evaluation out of reach for smaller organizations. Platforms may also perform differently depending upon the point in platform history, so some benchmark suites must also be run again to retain their validity.

One of many future directions is AI-guided benchmark configuration, where benchmarks are dynamically adjusted to the changing behavior of the system. As sustainability becomes more important, energy consumption benchmarking is critical; however, closed-loop autonomous optimization engines are still not used to link benchmarks and configuration recommendations. If integrated with observability platforms, benchmarking would be transformed from an episode-based to an operational task, where performance would be a continuous concern.

## 11. Conclusion

Middleware is not background infrastructure. It is the layer that determines whether real-time enterprise systems actually work—whether transactions complete, whether telemetry arrives in time to act on them, and whether SLAs hold when load spikes or nodes fail. The performance characteristics of these platforms have direct operational and commercial consequences. Yet the benchmarking tools available to evaluate them have not kept pace with their increasing importance.

MAP-Bench addresses that gap. It brings standardized metrics, enterprise-realistic workloads, fault-inclusive testing, and architectural-level evaluation into a single, vendor-neutral methodology. The shift it represents—from isolated load testing to structured architectural assessment—is not incremental. It changes what questions can be answered before a platform is deployed and what quality of answer is possible.

What makes MAP-Bench particularly relevant right now is the direction enterprise architecture is moving. Microservices, event-driven designs, multi-cloud deployments, and edge computing have all increased how much an organization depends on middleware to function correctly under unpredictable conditions. A platform chosen just because of a vendor's performance promises or a simple lab

test might seem good until a regional outage, a sudden increase in traffic, or a delay from users shows its weaknesses. MAP-Bench is designed specifically to surface those vulnerabilities before they become production incidents.

The framework also carries longer-term value. As middleware platforms evolve and new deployment models emerge—serverless brokers, AI-assisted routing, edge-native messaging—benchmarking methodologies will need to evolve alongside them. MAP-Bench provides a foundation that can be extended: new workload categories can be added, new fault types can be injected, and new topology descriptors can be standardized without abandoning the underlying measurement discipline.

For organizations that treat integration infrastructure as a strategic asset rather than a commodity, that discipline matters. Platform decisions made with rigorous benchmark data behind them are more defensible, more durable, and less likely to require costly remediation down the line. In a digital economy where real-time responsiveness is a baseline expectation rather than a differentiator, leaving middleware performance to vendor claims and informal testing is a risk that enterprises can no longer afford to take.

## References

[1] Kai Sachs, et al., "Performance evaluation of message-oriented middleware using the SPECjms2007 benchmark," Performance Evaluation, 2009. [Online]. Available: https://www.sciencedirect.com/science/article/abs/pii/S016653160900011X

[2] Xing Chen, et al., "Self-adaptive resource allocation for cloud-based software services based on iterative QoS prediction model," Future Generation Computer Systems, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/abs/pii/S0167739X19302894

[3] Vineet John and Xia Liu, "A Survey of Distributed Message Broker Queues," arXiv, 2017. [Online]. Available: https://arxiv.org/pdf/1704.00411

[4] Van-Nam Pham, et al., "Efficient Edge-Cloud Publish/Subscribe Broker Overlay Networks to Support Latency-Sensitive Wide-Scale IoT Applications," Symmetry, 2020. [Online]. Available: https://www.mdpi.com/2073-8994/12/1/3

[5] Martin Grambow, et al., "Continuous Benchmarking: Using System Benchmarking in Build Pipelines," 2019 IEEE International Conference on Cloud Engineering (IC2E), 2019. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8790186

[6] Arnamoy Bhattacharyya, et al., "Phase Aware Performance Modeling for Cloud Applications," University of Toronto, 2020. [Online]. Available: https://www.eecg.toronto.edu/~amza/papers/CLOUD2020.pdf

[7] Kai Sachs, et al., "Performance Modeling and Analysis of Message-Oriented Event-Driven Systems," Software and Systems Modeling. [Online]. Available: https://scispace.com/pdf/performance-modeling-and-analysis-of-message-oriented-event-1nda4aizzs.pdf

[8] Matteo Nardelli, et al., "Efficient Operator Placement for Distributed Data Stream Processing Applications," IEEE Transactions on Parallel and Distributed Systems, 2019. [Online]. Available: https://ieeexplore.ieee.org/document/8630099

[9] Christoph Laaber, et al., "Software microbenchmarking in the cloud. How bad is it really?" Empirical Software Engineering, 2019. [Online]. Available: https://dl.acm.org/doi/10.1007/s10664-019-09681-1

[10] Pascal Maissen, et al., "FaaSdom: A Benchmark Suite for Serverless Computing," arXiv, 2020. [Online]. Available: https://arxiv.org/pdf/2006.03271

[11] Sachs, K., Kounev, S., & Buchmann, A. (2009). Performance evaluation of message-oriented middleware using the SPECjms2007 benchmark. *Performance Evaluation, 66*(10), 1027–1042. https://doi.org/10.1016/j.peva.2009.03.003

[12] Chen, X., Zhang, H., & Li, Y. (2020). Self-adaptive resource allocation for cloud-based software services based on iterative QoS prediction model. *Future Generation Computer Systems, 105*, 287–296. https://doi.org/10.1016/j.future.2019.12.019

[13] John, V., & Liu, X. (2017). A survey of distributed message broker queues. *arXiv preprint arXiv:1704.00411*.

[14] Pham, V.-N., Nguyen, T. D., & Huh, E.-N. (2020). Efficient edge-cloud publish/subscribe broker overlay networks to support latency-sensitive wide-scale IoT applications. *Symmetry, 12*(1), 3. https://doi.org/10.3390/sym12010003

[15] Grambow, M., Haselbring, W., & Reussner, R. (2019). Continuous benchmarking: Using system benchmarking in build pipelines. In *Proceedings of the IEEE International Conference on Cloud Engineering* (pp. 321–326).

[16] Bhattacharyya, A., et al. (2020). Phase-aware performance modeling for cloud applications. *IEEE International Conference on Cloud Computing*.

[17] Sachs, K., et al. (2018). Performance modeling and analysis of message-oriented event-driven systems. *Software and Systems Modeling, 17*(2), 1–20.

[18] Nardelli, M., et al. (2019). Efficient operator placement for distributed data stream processing applications. *IEEE Transactions on Parallel and Distributed Systems, 30*(8), 1753–1767. https://doi.org/10.1109/TPDS.2019.2896239

[19] Laaber, C., Scheuner, J., & Leitner, P. (2019). Software microbenchmarking in the cloud: How bad is it really? *Empirical Software Engineering, 24*, 1–37. https://doi.org/10.1007/s10664-019-09681-1

[20] Maissen, P., et al. (2020). FaaSdom: A benchmark suite for serverless computing. *arXiv preprint arXiv:2006.03271*.

[21] Duan, S., et al. (2023). Distributed artificial intelligence empowered by end-edge-cloud computing: A survey. *IEEE Communications Surveys & Tutorials, 25*(2), 591–624. https://doi.org/10.1109/COMST.2023.3241234