

A Review of Comparative Studies on Performance Evaluation of Communication Mechanisms for Microservices

Berkcan Çiftçi[1(✉)] and Birol Çiloğlugil[2]

[1] Institute of Natural Sciences, Department of Computer Engineering, Ege University, 35100 Bornova, Izmir, Turkey
berkcanciftci35@gmail.com

[2] Faculty of Engineering, Department of Computer Engineering, Ege University, 35100 Bornova, Izmir, Turkey
birol.ciloglugil@ege.edu.tr

Abstract. As organizations increasingly adopt microservices to improve agility and scalability, selecting the right communication mechanisms becomes critical for system efficiency and fault tolerance. Therefore, a variety of communication technologies, including but not limited to HTTP REST, HTTP/2, gRPC, GraphQL, MQTT, AMQP, Apache Kafka, RabbitMQ, NATS Streaming, RocketMQ, Apache Pulsar, Redis, ActiveMQ, ActiveMQ Artemis, and WebSocket, are employed in a wide range of applications. Benchmarking frameworks such as Apache JMeter and the OpenMessaging Benchmark Framework are utilized to evaluate performances of different communication technologies in a variety of environments. For this purpose, performance evaluation metrics such as throughput, latency and response time are assessed to compare the performances of inter-process communication (IPC) technologies. Hence, this paper presents a review of studies that provide a comparative analysis of communication mechanisms employed in microservice architectures, focusing on the communication technologies, benchmarking frameworks, and performance evaluation metrics utilized for IPC. The findings indicate that HTTP REST and gRPC are the most frequently evaluated synchronous communication technologies because of their simplicity and compatibility. Besides, Kafka and RabbitMQ, which enhance scalability and fault tolerance, are identified as the most widely assessed asynchronous communication technologies. In addition, Apache JMeter is determined as the most popular benchmarking framework to conduct comparative performance evaluations. Finally, the most commonly employed performance evaluation metrics are detected as throughput and latency.

Keywords: Microservice Architecture · Inter-process Communication · Inter-service Communication · Benchmark Framework · Performance Evaluation Metrics

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2025
O. Gervasi et al. (Eds.): ICCSA 2025, LNCS 15650, pp. 98–113, 2025.
https://doi.org/10.1007/978-3-031-96962-1_7

1 Introduction

Microservice architectures have received significant attention in recent years, as a result of the need for improved agility, scalability, and fault tolerance in modern software systems. As organizations transition from monolithic to microservice-based architectures, the selection of appropriate inter-process communication (IPC) mechanisms has emerged as a critical factor influencing system performance and reliability [1].

Microservice communication mechanisms can be categorized as synchronous and asynchronous communication technologies, with HTTP REST, HTTP/2, gRPC, and GraphQL being examples of synchronous blocking communication technologies, whereas Kafka, RabbitMQ, MQTT, AMQP, NATS Streaming, RocketMQ, Apache Pulsar, Redis, ActiveMQ, ActiveMQ Artemis and WebSocket representing asynchronous non-blocking communication technologies [2].

The choice between synchronous and asynchronous communication mechanisms is crucial to increase system performance and enhance scalability and reliability [1]. Synchronous communication, exemplified by HTTP REST and gRPC, is widely adopted for its simplicity and compatibility, but may introduce inefficiencies in high-load scenarios [30]. Asynchronous communication, facilitated by message brokers like RabbitMQ and Kafka, supports decoupled, event-driven interactions, enhancing throughput and fault tolerance [31].

Apache JMeter [3] and the OpenMessaging Benchmark Framework [4], are utilized as benchmarking frameworks to evaluate IPC technologies by using various performance evaluation metrics. Throughput, latency, response time, availability, and scalability, are the most frequently assessed performance evaluation metrics to analyze efficiency and reliability of IPC technologies ([25,26,29]).

This paper aims to explore the communication technologies, benchmarking frameworks, and performance evaluation metrics utilized for IPC to provide a structured understanding of the studies presenting a comparative performance evaluation to optimize communication in microservice architecture. Hence, the research questions (RQs) of this study can be stated as given below:

- **RQ1.** Which technologies are utilized for inter-process communication in microservice architectures?
- **RQ2.** Which benchmarking frameworks are employed for evaluation of inter-process communication technologies?
- **RQ3.** Which performance evaluation metrics are leveraged for comparison of inter-process communication technologies?

2 Microservice Communication Technologies

Communication technologies can be categorized according to communication and collaboration styles such as syncronous blocking, asyncronous non-blocking, request-response and event-driven communication [2]. There are different technology choices that are used for implementation such as remote procedure call

(SOAP and gRPC), HTTP REST, GraphQL and message brokers. In addition, there are also protocols designed for specific environments (MQTT and AMQP). The most widely used inter-process communication technologies are briefly introduced below.

HTTP REST is a stateless architectural style for designing web services that use standard HTTP methods such as GET, POST, PUT, and DELETE. RESTful APIs are widely adopted due to their simplicity, scalability, and ease of integration with various platforms [5]. However, HTTP REST relies on polling mechanisms for real-time updates, which can introduce inefficiencies in high-frequency communication scenarios [6].

HTTP/2 is a major revision of the HTTP protocol, designed to improve web performance by reducing latency and optimizing resource usage. It introduces features like multiplexing, header compression, and server push, enabling faster and more efficient communication between clients and servers [7]. As HTTP/2 supports backward compatibility with HTTP/1.1 and provides high performance and scalability, it is widely adopted in modern web applications and APIs.

gRPC is an RPC framework developed by Google that uses HTTP/2 for data transfer and Protocol Buffers (protobuf) for interface definition. Because of its high performance and cross-language support, gRPC is widely used in microservices architectures, especially for real-time communication, and low-latency applications. It also supports bidirectional streaming, flow control, and authentication to develop more efficient systems with high interoperability [8].

GraphQL, developed by Facebook, is a query language for APIs that is widely used in modern web and mobile applications with complex data requirements and rapid iterations. HTTP REST relies on fixed endpoints while GraphQL makes it possible for clients to request exactly the data they need to reduce overfetching and underfetching of information [9]. GraphQL enables clients to define the structure of the response to support writing more efficient and flexible queries.

Message Queuing Telemetry Transport (MQTT) protocol is a lightweight messaging standard developed for low-bandwidth and high-latency networks that supports publish/subscribe (pub/sub) model. In the pub/sub model, which increases scalability and resilience, the producers send messages to topics, and consumers process them asynchronously. As it supports pub/sub messaging and provides low level of overhead and high quality of service, MQTT is highly adopted in constrained devices and Internet of Things (IoT) applications [10]. However, MQTT lacks built-in security mechanisms and requires additional layers for encryption and authentication [11].

Advanced Message Queuing Protocol (AMQP) is an open standard that provides reliable and interoperable messaging for message-oriented middleware. While protocols like MQTT focus on lightweight communication, AMQP is designed for robust transactional messaging with high reliability [12]. Due to its support of message queuing, routing, and security, AMQP is widely used in financial services and enterprise messaging systems [13].

Kafka is a distributed event streaming platform that is based on a pub/sub model to provide high-throughput and fault-tolerant data processing [14]. The partitioned architecture of Kafka enables efficient parallel processing as it is ideal for event-driven architectures and real-time analytics [15].

RabbitMQ is a message broker that facilitates reliability by using the AMQP protocol. RabbitMQ is highly adopted in distributed systems and microservices as it supports flexible routing, message durability and acknowledgment mechanisms in an effective way [16].

NATS Streaming is a lightweight and easy to deploy messaging platform developed based on the NATS platform. NATS Streaming is ideal for use cases like event sourcing and real-time data processing as a result of its support of features like message persistence, at-least-once delivery, and durable subscriptions [17]. Therefore, NATS Streaming provides a robust and scalable solution for distributed systems and microservices that require high-performance, fault tolerance, and guaranteed and low-latency message delivery.

RocketMQ is a distributed messaging and streaming platform developed by Alibaba which enables high availability, high throughput, low latency, high scalability, and fault tolerance, and widely used in large-scale, mission-critical applications. Due to its support for pub/sub messaging, message tracing and transactional messaging, it is generally employed in financial transactions, log aggregation, and real-time analytics [18].

Apache Pulsar combines the advantages of traditional messaging systems with the flexibility of streaming platforms to provide features like multi-tenancy, geo-replication, and tiered storage [19]. It is designed as a pub/sub messaging platform that offers low-latency message delivery, high performance, scalability, and durability which makes it useful for real-time analytics, event-driven architectures, and IoT applications.

Redis (Remote Dictionary Server) is an open-source, in-memory data storage technology that supports various data structures such as strings, hashes, lists, and sets. Thus, it is widely employed as a database and cache memory because of its speed and versatility. However, due to its pub/sub messaging capabilities, Redis can also be used as a message broker for lightweight, real-time communication in distributed systems that require high-performance operations with low latency [20].

ActiveMQ is an open-source message broker that implements the Java Message Service (JMS) API and supports message protocols like MQTT and AMQP to provide flexible and reliable communication of distributed systems [21]. ActiveMQ offers features such as message persistence, clustering and failover for developing enterprise applications.

ActiveMQ Artemis is successor of the ActiveMQ message broker and supports message protocols such as MQTT and AMQP. It provides a robust architecture to enable message persistence, clustering, and high availability [22] and offer high-performance, reliability, scalability and flexibility.

WebSocket is a full-duplex communication protocol that enables persistent, low-latency connections between clients and servers. While HTTP REST is a

synchronous request-response driven protocol, WebSocket facilitates real-time communication with minimal overhead to develop interactive applications such as online gaming and financial trading [23]. However, WebSocket's persistent connections can increase server resource consumption, requiring careful management in large-scale deployments [24].

3 Related Work

The studies presented in Table 1 offer a comprehensive overview of research focused on the evaluation and optimization of communication protocols in microservice architectures. The studies are listed in ascending order by year, with four studies being conducted in the years 2020 and 2023; and two studies each in 2022 and 2024, reflecting the growing interest in this area. The studies in Table 1 are discussed below based on the IPC technologies used, benchmarking frameworks, and performance evaluation metrics as defined by the research questions.

Table 1. Primary Studies on Evaluation of Microservices Communication Mechanisms

ID	Year	Title	Ref
S1	2020	A study on Modern Messaging Systems- Kafka, RabbitMQ and NATS Streaming	[25]
S2	2020	Low Latency Message Brokers	[26]
S3	2020	Evaluating the Impact of Inter-Process Communication in Microservice Architectures	[27]
S4	2020	A Fair Comparison of Message Queuing Systems	[28]
S5	2022	Evaluating the Inter-Service Communication on Microservice Architecture	[29]
S6	2022	Microservice Communication For IoT-based Systems. Architecture Review and Performance Test	[30]
S7	2023	Benchmarking Message Queues	[31]
S8	2023	Comparative Evaluation of Java Virtual Machine-Based Message Queue Services: A Study on Kafka, Artemis, Pulsar, and RocketMQ	[32]
S9	2023	Optimized Strategy for Inter-Service Communication in Microservices	[33]
S10	2023	Evaluation of microservice communication while decomposing monoliths	[34]
S11	2024	Performance evaluation of microservices communication with REST, GraphQL, and gRPC	[35]
S12	2024	Performance Analysis of RESTFUL Web Services and RABBITMQ for Microservices based Systems on Cloud Environment	[36]

Sharvari and Sowmya Nag [25] compared the performance, features, and use cases of three message brokers which includes Kafka, RabbitMQ, and NATS Streaming. No experimental comparison was made. It was determined that Kafka is well-suited for high-throughput and low-latency applications, such as real-time analytics, due to its distributed commit log and partition-based architecture. RabbitMQ was identified as ideal for complex routing and critical systems requiring message persistence and guaranteed delivery, particularly in IoT scenarios. NATS Streaming was found to be lightweight and capable of rapid message delivery with at-least-once guarantees but exhibited higher latency compared to Kafka and RabbitMQ.

Hegde and Nagaraja [26] evaluated the performance, features, and use cases of three low-latency message brokers that Redis, Kafka, and RabbitMQ. No experimental comparison was made. It was observed that Redis demonstrated the highest throughput and lowest latency due to its in-memory architecture, making it suitable for scenarios prioritizing speed over guaranteed delivery. It was also stated that by providing robust scalability and flexible message delivery guarantees, Kafka is appropriate for big data streaming and real-time analytics. RabbitMQ was identified as offering advanced routing and message persistence, making it reliable for critical applications requiring guaranteed delivery.

Shafabakhsh et al. [27] evaluated the performance efficiency and availability of IPC methods in microservice architectures, comparing HTTP REST, gRPC and RabbitMQ. Through experimental load testing using Apache JMeter, it was highlighted that synchronous methods, particularly gRPC, perform better under low load conditions, demonstrated higher throughput and lower latency. However, under high-load scenarios, RabbitMQ outperforms synchronous methods, offering better throughput, scalability, and system resilience. It was also concluded that the choice of IPC mechanism should align with system requirements: synchronous communication is ideal for immediate response scenarios, while asynchronous communication is better suited for high-load, fault-tolerant systems.

Fu et al. [28] conducted a comparative analysis of five message queueing systems: Kafka, RabbitMQ, RocketMQ, ActiveMQ, and Pulsar, evaluated by a custom benchmarking framework. Kafka excels in high-throughput scenarios like log processing due to batch processing and partition-based storage. RocketMQ, optimized for low latency, is ideal for real-time applications. RabbitMQ supports advanced features like priority and delay queues, making it suitable for multi-functional messaging. ActiveMQ has limited scalability and is best for lightweight applications. Pulsar offers robust scalability and global ordering but incurs network overhead.

Weerasinghe and Perera [29] conducted an experiment in which the performance of HTTP REST, gRPC, and WebSocket technologies was compared in the context of inter-process communication in microservice architectures. Significant improvements were observed in response time, latency and throughput, with gRPC outperforming HTTP REST and WebSocket.

Zakutynskyi and Rabodzey [30] conducted evaluation of various communication protocols within IoT-based microservice systems. The technologies evaluated included HTTP REST, MQTT, AMQP, and gRPC. Performance metrics such as throughput, concurrency, scalability, and connection establishment time were analyzed using Apache JMeter. The study revealed that MQTT and AMQP demonstrated superior performance in high-load scenarios such as HTTP REST and gRPC, exhibited greater efficiency for large file transfers.

Md. Showkat Hossain Chy et al. [31] presented a comparative evaluation of four popular Java Virtual Machine (JVM) based message queue (MQ) services: ActiveMQ Artemis, Kafka, RabbitMQ and Redis. The evaluation is based on performance metrics like CPU utilization, memory usage, garbage collection, latency, and throughput using the OpenMessaging Benchmark Framework. Redis is optimal for low-latency use cases, Kafka for high-throughput scenarios, and ActiveMQ Artemis and RabbitMQ for balanced performance in moderate workloads.

Chy et al. [32] aimed to provide a comprehensive comparative evaluation of four prominent JVM-based MQ services—Kafka, Pulsar, Artemis, and RocketMQ—focusing on performance evaluation metrics such as latency, throughput, CPU utilization, memory usage, and garbage collection efficiency. Using the OpenMessaging Benchmark Framework, the study evaluates these MQ services. Kafka is optimal for high-throughput systems, Artemis excels in low-latency scenarios, and Pulsar offers a balanced approach with moderate resource usage. RocketMQ, while efficient for smaller messages, faces challenges with larger payloads.

Weerasinghe and Perera [33] proposed a solution to reduce the inter-process communication latency in microservices by using an asynchronous communication pattern with the Redis Stream data structure. Apache JMeter was used to benchmark the performance of the Redis-based solution against HTTP REST. Experimental results show that Redis outperforms HTTP REST in terms of response time and throughput.

Kazanavičius and Mažeika [34] conducted a comprehensive evaluation of communication technologies, focusing on the transition from monolithic architectures to cloud-native, microservices-based systems. Five communication technologies were compared: HTTP REST, RabbitMQ, Kafka, gRPC, and GraphQL, analyzing their performance, scalability, and suitability. The experimental setup was designed with microservices implemented in $C\#$, and performance metrics such as latency, throughput, and memory consumption were evaluated. Different topologies, including linear, tree, and star, were also examined to understand how communication technologies perform in diverse architectural setups. The results revealed that the lowest latency for small messages was achieved by RabbitMQ, which was identified as being ideal for high-throughput, low-latency scenarios. GraphQL was recognized as flexible and efficient for querying structured data, but it was observed to incur higher CPU usage and was identified as being best suited for star topologies, where a central service functions as a query engine. Kafka, though effective for event-driven architectures, struggled

with high-throughput scenarios. HTTP REST was acknowledged for its simplicity and widespread adoption, but limitations in throughput were observed under heavy loads. It was identified that gRPC is considered superior for low-latency and high-throughput applications.

Niswar et al. [35] conducted a comparative performance analysis of HTTP REST, GraphQL, and gRPC within a microservices architecture. Given the critical importance of selecting the appropriate communication protocol for optimal system performance in microservices, two distinct data retrieval methods, flat data fetching and nested data fetching, were focused on. Through experiments using Redis and MySQL databases, it was found that gRPC exhibited the fastest response times, while HTTP REST demonstrated the lowest CPU utilization. gRPC's utilization of HTTP/2, with its multiplexing feature allowing multiple requests over a single connection, makes it highly efficient for concurrent remote procedure calls. The simplicity and low resource consumption of HTTP REST are considered to offer distinct advantages. GraphQL, despite its flexibility, is less suitable for high-concurrency environments due to its higher CPU demands and slower response times.

Bux and Shenoy [36] investigated synchronous and asynchronous interprocess communication in microservices-based systems by comparing RESTful web services and RabbitMQ on Amazon Web Services (AWS) cloud environment. They found significant performance improvements, demonstrating that RabbitMQ outperforms HTTP REST in terms of throughput, average request processing time, and scalability, especially under high concurrent user loads.

4 Findings

The findings for each research questions are discussed below based on the primary studies listed in Table 1.

RQ1. Which technologies are utilized for inter-process communication in microservice architectures?

The communication technologies used in the primary studies listed in Table 1 are systematically summarized in Table 2. When the number of communication technologies compared in the primary studies was analyzed, [28,30,34] were the studies that evaluated the highest number of different technologies, with five technologies analyzed in each study. In these three studies, no common technologies were observed; however, HTTP REST and gRPC were evaluated in [30,34], whereas Kafka and RabbitMQ were examined in [28,34]. Furthermore, the performance of four technologies was compared in [31,32], with Kafka and ActiveMQ Artemis being the common technologies in both studies. Besides, three different technologies were evaluated in [25–27,29,35], while [32,36] compared the performance of two different technologies.

Figure 1 illustrated the frequency of IPC technologies compared in the primary studies based on Table 2, with a focus on the most widely used technologies

Table 2. Overview of Communication Technologies Utilized in Primary Studies

ID	HTTP REST	HTTP2	gRPC	Graph QL	MQTT	AMQP	Kafka	Rabbit MQ	NATS Stream-ing	Rocket MQ	Pulsar	Redis	Active MQ	Active MQ Artemis	Web Socket
S1							✓	✓	✓						
S2							✓	✓				✓			
S3	✓		✓					✓							
S4							✓	✓		✓	✓		✓		
S5	✓		✓												✓
S6	✓	✓	✓		✓	✓									
S7							✓	✓				✓		✓	
S8							✓			✓	✓			✓	
S9	✓											✓			
S10	✓		✓	✓			✓	✓							
S11	✓		✓	✓											
S12	✓							✓							

and listing the technologies utilized in three or more studies. RabbitMQ ([25–28,31,34,36]) and HTTP REST ([27,29,30,33–36]) were the technologies most frequently evaluated in primary studies, with seven different studies examining each technology, whereas three studies ([27,34,36]) utilized both technologies. Moreover, Kafka is also a widely used technology, with its performance being evaluated in six studies ([25,26,28,31,32,34]). Besides, gRPC is used in five studies ([27,29,30,34,35]), followed by Redis with three studies ([26,31,33]).

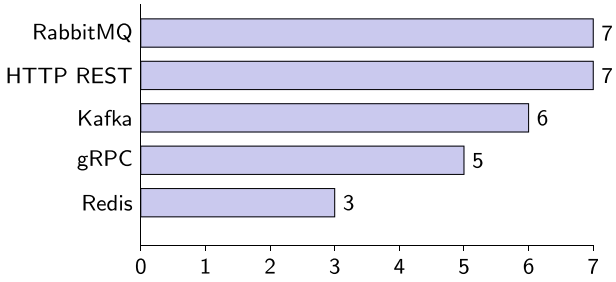


Fig. 1. Usage Statistics of Communication Technologies

As emerging technologies not listed in Fig. 1 for simplicity purposes, the performances of ActiveMQ Artemis ([31] [32]), RocketMQ ([28] [32]), Apache Pulsar ([28] [32]) and GraphQL ([34] [35]) were examined in two studies each. In addition, technologies such as HTTP/2 [30], NATS Streaming [25], ActiveMQ [28], WebSocket [29], and protocols like MQTT [30] and AMQP [30], were each investigated in one study.

RQ2. Which benchmarking frameworks are employed for evaluation of inter-process communication technologies?

Table 3. Benchmark Frameworks and Evaulation Metrics utilized in Primary Studies

ID	Benchmark Framework	Performance Evaulation Metrics
S1	No Benchmark	Message Delivery Guarantee, Message Persistence, Message Ordering, Latency, Throughput, Availability, Scalability
S2	No Benchmark	Message Delivery Guarantee, Message Persistence, Message Ordering, Latency, Throughput, Availability, Scalability
S3	Apache JMeter	Throughput, Availability
S4	Custom B. Tool	Latency, Throughput
S5	Apache JMeter	Response Time, Throughput, Latency, Payload Size Effect
S6	Custom B. Tool	Init connection time, Messages / Requests per second, Time to send 1M messages/requests, Throughput, Data transferring performance
S7	OpenMessaging B.F.	Latency, Throughput
S8	OpenMessaging B.F.	Latency, Throughput, CPU Utilization, Memory Usage, Garbage Collection
S9	Apache JMeter	Transaction Per Second (TPS), Response Time, Turnaround Time
S10	Custom B. Tool	Latency, Throughput, Request / Response Size, Microservice Application Size, Memory Usage Size, Boot Time, Architecture, Topology Libraries
S11	Apache JMeter	Response Time, CPU Utilization
S12	Apache JMeter	Total Number Of Request Processed, Throughput, Average Request Processing Time

The benchmarking frameworks utilized for evaluation of IPC technologies in primary studies listed in Table 1 are presented in Table 3 with the performance evaluation metrics used in each study, which will be discussed for RQ3. Figure 2 illustrated the frequency of the benchmarking frameworks used for comparison of IPC technologies based on Table 3. Apache JMeter is observed as the most widely used benchmarking framework, with a usage count of five ([27,29,33,35,36]), reflecting its popularity for performance testing in IPC evaluations. As the second popular approach, three studies([28,30,34]) developed their own custom benchmarking tool. The OpenMessaging Benchmark Framework is utilized in two studies ([31,32]), highlighting its role in standardized performance evalua-

tions of messaging systems. Furthermore, two studies ([25,26]) conducted theoretical comparisons without using any benchmarking tools.

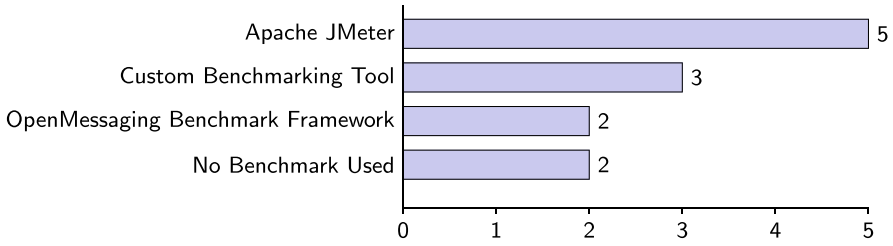


Fig. 2. Usage Statistics of Benchmarking Frameworks

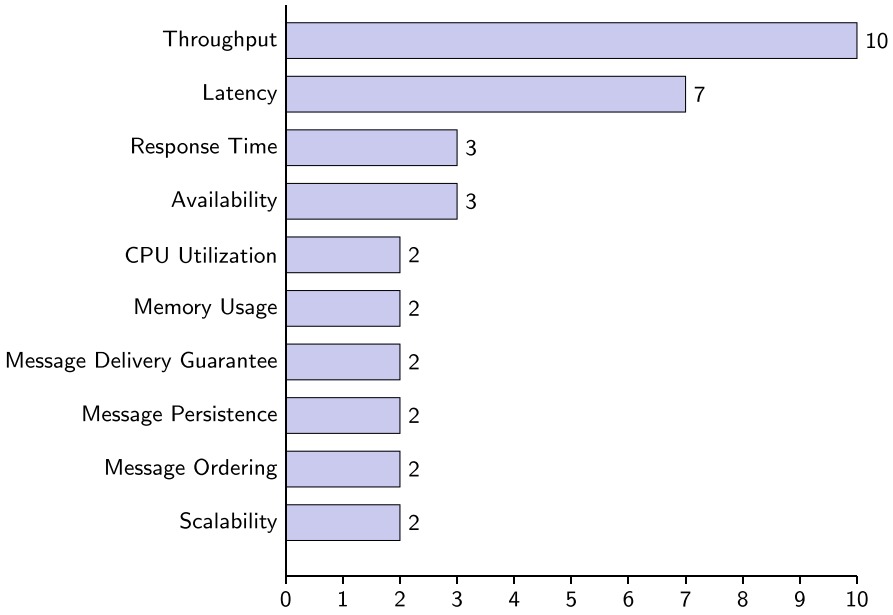


Fig. 3. Evaluation Metric-wise Distribution of Primary Studies.

Developing a custom benchmarking tool is a challenging task that requires a lot of effort; however, a new tool may need to be developed based on some specific requirements affecting the evaluation. A new custom benchmarking tool is developed by [28], because the evaluation parameters of ready-to-use tools can be set differently for each tool, so the parameters cannot be arbitrarily set as needed. On the other hand, each tool supports testing different number of metrics and measures them in different ways, thus new custom benchmarking tools utilizing new metrics were proposed by [30,34].

RQ3. Which performance evaluation metrics are leveraged for comparison of inter-process communication technologies?

The performance evaluation metrics used for comparing performance of IPC technologies in primary studies listed in Table 1 were presented in Table 3. To increase understandability, Table 4 presents a detailed representation of the metrics utilized by each primary study. Furthermore, Fig. 3 illustrated the frequency of the metrics used for comparison of IPC technologies.

Table 4. Overview of Performance Evaluation Metrics Utilized in Primary Studies

ID	Msg. Del.	Msg. Per.	Msg. Ord.	Latency	Through-put	Avail.	Scale.	Resp. Time	CPU Util.	Mem. Usage	Other Metrics
S1	✓	✓	✓	✓	✓	✓	✓				
S2	✓	✓	✓	✓	✓	✓	✓				
S3					✓	✓					
S4				✓	✓						
S5				✓	✓			✓			Payload Size Effect
S6					✓						Init, Msg/s, 1M Msg, Data
S7				✓	✓						
S8				✓	✓				✓	✓	Garbage Col.
S9								✓			TPS, Turn. Time
S10				✓	✓					✓	Req/Res, App Size, Boot, Arch, Topo
S11								✓	✓		
S12					✓						Req. Count, Avg. Time

Throughput is identified as the most frequently evaluated metric, appearing in ten studies ([25–32,34,36]), followed by latency, which is evaluated in seven studies ([25,26,28,29,31,32,34]). Besides, response time ([29,33,35]) and

availability ([25–27]) were each utilized by three studies. Six metrics were used by two studies each, with four of them (Message Delivery Guarantee, Message Persistence, Message Ordering and Scalability) being used by the same non-experimental studies ([25,26]). The other two metrics utilized in two studies are CPU utilization ([32,35]) and Memory Usage ([32,34]).

In addition, there are evaluation metrics used by only a single study, including Payload Size Effect [29], Init Connection Time [30], Messages/Request Per Second [30], Time to Send 1M Messages/Requests [30], Data Transferring Performance [30], Garbage Collection [32], Transaction Per Second (TPS) [33], Turnaround Time [33], Request/Response size [34], Microservice Application Size [34], Boot Time [34], Architecture [34], Topology [34], Total Number Of Request Processed [36], Average Request Processing Time [36].

5 Discussion and Conclusion

In this paper, a detailed analysis of comparative studies employing various communication technologies, benchmarking frameworks, and performance evaluation metrics to evaluate IPC technologies is provided. The findings are presented based on the utilized communication technologies, the benchmarking frameworks and the performance evaluation metrics, with each of them being investigated by a corresponding research question.

As the most dominant category among the primary studies represented in Table 1, [27,29,30,33,34,36] explore the impact of inter-process communication in microservices in a broader way by comparing performances of synchronous and asynchronous communication technologies. As another dominant category, [25,26,28,31,32] focus on performance evaluation of modern message brokers like Kafka, RabbitMQ, while [35] is the only study conducting experiments with only synchronous communication technologies. In addition, some of these studies focused on evaluating the performance of communication protocols with specific deployment purposes such as IoT [30] and cloud environments [36].

The studies in Table 1 can be categorized into three according to the case studies employed; [25,26] did not perform any experiments, [27–30] performed hypothetical experiments, and [31–35], and [36] conducted experiments in real-world scenarios.

HTTP REST, Kafka and RabbitMQ emerge as the most frequently studied technologies, HTTP REST is widely used due to its simplicity and compatibility features [30], while Kafka excels in high-throughput scenarios [31] and RabbitMQ performs well in environments requiring complex routing and fault tolerance [27]. The findings indicate that synchronous methods like HTTP REST and gRPC are ideal for low-load, immediate-response scenarios [30], while asynchronous methods such as RabbitMQ and Kafka are better suited for high-load, fault-tolerant systems [31].

Among benchmarking frameworks, Apache JMeter is pivotal for evaluating IPC performance, as it is the most frequently employed one ([27,29,33,35,36]). Besides, some custom benchmarking tools have been developed ([28,30,34])

based on specific requirements, and the OpenMessaging Benchmark Framework is also utilized in some studies ([31,32]).

Performance evaluation metrics, including throughput, latency, response time, CPU utilization, availability, memory usage, message delivery guarantee, message persistence, message ordering, and scalability, are frequently assessed to compare the efficiency of communication methods ([25,26,29]). Among these metrics, throughput and latency are the most commonly employed ones as illustrated in Fig. 3.

Future research could focus on hybrid communication architectures that combine synchronous and asynchronous methods, leveraging their respective strengths to address specific system requirements. Another promising avenue involves integrating machine learning techniques to dynamically optimize communication frameworks based on workload patterns, enhancing adaptability and performance. Finally, expanding the scope of case studies to include edge computing and 5G-enabled IoT environments could provide valuable insights into the scalability and reliability of IPC mechanisms in next-generation systems.

References

1. Karabey Aksakalli, I., Çelik, T., Can, A.B., Tekinerdogan, B.: Deployment and communication patterns in microservice architectures: a systematic literature review. J. Syst. Softw. **180**, 111014 (2021). https://doi.org/10.1016/j.jss.2021.111014
2. Newman, S.: (2021). Building Microservices: Designing fine-grained systems (2nd ed.). O'Reilly Media. ISBN 9781492034025
3. Apache Software Foundation (2025). Apache JMeter. Retrieved from https://jmeter.apache.org/
4. OpenMessaging Project (2025). OpenMessaging Benchmark Framework. Retrieved from https://openmessaging.cloud/benchmark/
5. Fielding, R.T.: Architectural styles and the design of network-based software architectures (Doctoral dissertation, University of California, Irvine) (2000)
6. Richardson, L., Amundsen, M.: RESTful Web APIs. O'Reilly Media (2013)
7. IETF. (2015). HTTP/2: Hypertext Transfer Protocol Version 2. RFC 7540. Retrieved from https://httpwg.org/specs/rfc7540.html
8. Google (2023). gRPC: A high-performance, open-source universal RPC framework. Retrieved from https://grpc.io
9. GraphQL Foundation (2023). GraphQL: A query language for your API. Retrieved from https://graphql.org/
10. Banks, A., Gupta, R.: MQTT version 3.1.1. OASIS Standard (2014)
11. Singh, K., Kim, T.: Security analysis of MQTT protocol in IoT applications. IEEE Internet Things J. **5**(6), 4680–4690 (2018)
12. Vinoski, S.: Advanced message queuing protocol. IEEE Internet Comput. **10**(6), 87–89 (2006)
13. Dobbelaere, P., Esmaili, K.: Kafka versus RabbitMQ: a comparative study of two distributed messaging frameworks. In: Proceedings of the International Conference on Cloud Engineering, pp. 35–42 (2017)
14. Kreps, J., Narkhede, N., Rao, J.: Kafka: a distributed messaging system for log processing. Proc. NetDB **11**(1), 1–7 (2011)

15. Narkhede, N., Shapira, G., Palino, T.: Kafka: The definitive guide. O'Reilly Media (2017)
16. Videla, A., Williams, J.: RabbitMQ in action. Manning Publications (2012)
17. NATS.io. NATS Streaming: A high-performance messaging system (2023). Retrieved from https://nats.io
18. Apache RocketMQ (2023). RocketMQ: Distributed messaging and streaming platform. Retrieved from https://rocketmq.apache.org/
19. Apache Pulsar (2023). Apache Pulsar: Distributed pub/sub messaging system. Retrieved from https://pulsar.apache.org/
20. Redis Labs (2023). Redis: The open-source, in-memory data store. Retrieved from https://redis.io
21. Apache ActiveMQ (2023). ActiveMQ: Open-source message broker. Retrieved from https://activemq.apache.org/
22. Apache ActiveMQ Artemis (2023). ActiveMQ Artemis: High-performance messaging broker. Retrieved from https://activemq.apache.org/components/artemis/
23. Fette, I., Melnikov, A.: The WebSocket Protocol. RFC 6455. Retrieved from https://www.rfc-editor.org/rfc/rfc6455 (2011)
24. Lubbers, P., Albers, B.: Pro HTML5 Programming: Powerful APIs for Richer Internet Application Development. Apress (2010). Retrieved from https://link.springer.com/book/10.1007/978-1-4302-2791-5
25. Sharvari, T., Sowmya Nag, K.: A study on modern messaging systems: Kafka, RabbitMQ, and NATS Streaming. J. Distrib. Syst. **15**(3), 123–145 (2020). https://doi.org/10.48550/arXiv.1912.03715
26. Hegde, R.G., Nagaraja, G.S.: Low latency message brokers. Int. Res. J. Eng. Technol. (IRJET), **7**(5), 2731–2738 (2020). https://www.irjet.net/archives/V7/i5/IRJET-V7I5523.pdf
27. Shafabakhsh, B., Lagerström, R., Hacks, S.: Evaluating the impact of inter-process communication in microservice architectures. In: 8th International Workshop on Quantitative Approaches to Software Quality (QuASoQ 2020). CEUR Workshop Proceedings (2020). http://ceur-ws.org/ISSN/1613-0073
28. Fu, G., Zhang, Y., Yu, G.: A fair comparison of message queuing systems. IEEE Access **8**, 220081–220093 (2020). https://doi.org/10.1109/ACCESS.2020.3046503
29. Weerasinghe, L.D.S.B., Perera, I.: Evaluating the inter-service communication on microservice architecture. In: 2022 IEEE International Conference on Innovations in Technology (INOCON), (pp. 1–6). IEEE (2022). https://doi.org/10.1109/INOCON60754.2024.10511747
30. Zakutynskyi, I.V., Rabodzey, I.E.: Performance characteristics of communication protocols in IoT-based microservice systems. Telecommun. Radio Eng. **4**(74), 73–78 (2022). https://doi.org/10.18372/1990-5548.74.17311
31. Maharjan, R., Chy, M.S.H., Arju, M.A., Cerny, T.: Benchmarking message queues. Telecom. **4**(2), 298–312 (2023). https://doi.org/10.3390/telecom4020018
32. Chy, M.S.H., Arju, M.A.R., Tella, S.M., Cerny, T.: Comparative evaluation of java virtual machine-based message queue services: a study on Kafka, Artemis, Pulsar, and RocketMQ. 1 Electronics **12**(23), 4792 (2023). https://doi.org/10.3390/electronics12234792
33. Weerasinghe, S., Perera, I.: Optimized strategy for inter-service communication in microservices. Int. J. Adv. Comput. Sci. Appl. **14** (2023). https://doi.org/10.14569/IJACSA.2023.0140233
34. Kazanavičius, J., Mažeika, D.: Evaluation of microservice communication while decomposing monoliths. Comput. Inform. **42**(1), 1–36 (2023). @split

35. Niswar, M., Safruddin, R.A., Bustamin, A., Aswad, I.: Performance evaluation of microservices communication with REST, GraphQL, and gRPC. Int. J. Electron. Telecommun. **70**(2), 429–436 (2024)
36. Bux, R., Shenoy, G.S.: Performance analysis of RESTful web services and RabbitMQ for microservices-based systems on cloud environment. In: 2024 3rd International Conference on Innovations in Technology (INOCON), (pp. 1–6). IEEE (2024). https://doi.org/10.1109/INOCON60754.2024.10511747