A Comparative Study Evaluation of Kafka and RabbitMQ: Performance, Scalability and Stress Test in Distributed Messaging Systems

Muhammad Rias Ramadan [1,*)], Ach Muhyil Umam [2)], Anani Asmani [3)], and Royyana Muslim Ijtihadie [4)]

[1, 2, 3, 4)] Department of Informatics, Institut Teknologi Sepuluh Nopember
Surabaya, Indonesia

E-mail: 6025241049@student.its.ac.id[1)], muhyil.umam1@gmail.com[2)], 6025241008@student.its.ac.id[3)], and roy@if.its.ac.id[4)]

ABSTRACT

In the message broker ecosystem, Apache Kafka and RabbitMQ (Robust Messaging Queue) are the two most widely adopted solutions. Both have fundamental differences in terms of architecture and performance characteristics. Kafka is designed for high-throughput, scalable streaming data processing, while RabbitMQ offers flexible message routing, delivery reliability, and complex queue management. This study presents a comprehensive comparative analysis between Apache Kafka and RabbitMQ to evaluate performance, scalability, and stress test resilience to select the most suitable message broker in modern distributed system architecture. The experimental testing process was conducted in four different scenarios: first, message size variations (1 kilobyte/KB, 10 KB, and 100 KB) to measure performance based on payload size; second, message volume variations (10,000, 50,000, and 100,000 messages) to see throughput limits and resource usage; third, variations in the number of consumers (1, 5, and 10 consumers) to measure the scalability of the consumer system; fourth, high-intensity stress tests (100,000 messages in 10 seconds) to evaluate stability and latency during overload. The key performance metrics used include throughput, latency, CPU (Central Processing Unit) usage, and RAM (Random Access Memory) consumption, which are carefully evaluated. The overall results of the experiment show that RabbitMQ is more suitable for systems that are sensitive to speed and message volume, while Kafka is more appropriate for extreme workloads with high durability requirements. This experiment provides empirical data concluding that RabbitMQ is highly effective for applications requiring high-volume individual message delivery with low latency, while Kafka's strength lies in handling large-scale data streams and maintaining stability under intense and sustained loads.

Keywords: Message queue, Kafka, RabbitMQ, throughput, latency, CPU utilisation, RAM consumption, stress test, experimental scenario.

1. Introduction

The rapid development of information technology has driven the need for software systems capable of processing data quickly. One of the architectural approaches that is widely used to meet these needs is a distributed system. This distributed system is a collection of independent computing components, such as servers, computers, or other software processes that communicate with each other and coordinate through message passing [1]. In its implementation, there are various models of distributed systems, one of which is the microservices architecture.

Microservices architecture is a software development approach that divides the system into small, autonomous, and separate services based on specific business needs, where each service communicates with the other in an integrated manner. Each service is usually developed, managed, and scaled independently, thus providing flexibility, scalability, and ease of management of the system as a whole [2]. In the implementation of microservices archi-

* Corresponding author.
Received: July 10th, 2025. Revised: November 4th, 2025. Accepted: December 4th, 2025.
Available online: January 15th, 2026.

DOI: https://doi.org/10.12962/j24068535.v24i1.a1345

tecture, communication between services is a very crucial component. It requires an intermediary that can bridge the exchange of messages between services efficiently, reliably, and flexibly [3]. Message broker for an intermediary system that receives, stores, and distributes messages between components or services in the system. By using a message broker, the system becomes more loosely coupled, more scalable, and more resilient to disruption than traditional synchronous systems [4]. There are several previous studies describing message broker platforms, but the two most prominent and widely used technologies are Apache Kafka and RabbitMQ [5], which despite having similar functionality, are designed with different technical approaches and architectures.

Apache Kafka is a distributed streaming platform for processing high-volume data in real-time. It uses distributed commit logs for persistent data storage and independent consumption by multiple consumers. Apache Kafka excels in high throughput, scalability, and efficiency, and is widely used for log processing, activity tracking, and real-time analytics [6]. Meanwhile, RabbitMQ is an AMQP-based message broker that excels in message flow control, delivery reliability, and routing flexibility. The platform is suitable for business-critical systems such as financial transactions and enterprise back-end services that require guaranteed delivery and complex queue management [7].

Although both Apache Kafka and RabbitMQ are popular message brokers, there is no one-size-fits-all solution. Their performance varies greatly depending on the requirements and the way they are used. Previous research, such as that conducted by [8], has shown that the study confirmed Apache Kafka's superiority in throughput for small messages, while RabbitMQ consistently showed lower latency. This discrepancy confirms that there is no one universal solution, and system selection needs to consider the specific context of use. However, to date, there is no comprehensive and systematic comparative approach to evaluating both systems from the three main dimensions of performance, scalability, and burst. Therefore, we conducted a comparative analysis to examine the performance of both systems in depth, with the aim of identifying which technology is superior under various conditions, especially when handling small, medium and large messages, with the number of messages and the number of customers (according to the scenario). Thus, this research determines which message broker is the most optimal and efficient for handling messages on a small and large scale.

With this research we contribute a comparative analysis between Apache Kafka and RabbitMQ with a multi-dimensional evaluation that not only measures performance but also evaluates the two platforms based on three metrics with four test scenarios including tests based on variations in message size, tests based on the number of messages sent, tests based on the increase in the number of consumers and stress tests to identify the limit points and performance degradation patterns of each platform. To provide a more complete picture of the operational efficiency and computational costs required by each message broker.

Based on this description, the formulation of the problem in this study is as follows:

1. How to create various scenarios in conducting performance comparison experiments based on different variables?
2. In what scenario are Apache Kafka and RabbitMQ superior, and how do the performances of the two compare?

This research aims to comprehensively analyse and compare the two message broker platforms, namely Apache Kafka and RabbitMQ, so that users and researchers can choose the technology that best suits their system needs. The expected benefits of this research are to provide practical references and empirical data related to the performance, scalability, and resource use efficiency of the two platforms, so that they can help in making technical decisions, planning system architecture, and developing further research in the field of distributed systems and microservices.

The following sections of this paper are organized as follows: Section 2 discusses previous studies related to micro-services architecture, distributed systems, message-oriented middleware, and performance comparisons between Apache Kafka and RabbitMQ. Section 3 describes the tools used in conducting the experiments. Section 4 describes the testing methodology, which includes scenario design, test parameters, and evaluation metrics used to measure the performance of both platforms. Section 5 presents the test results obtained, performance analysis,

scalability, resource efficiency, and a discussion comparing the two systems. Finally, Section 6 summarizes the research results based on the experimental results.

2. Related Work

In modern distributed systems, message brokers are an important component for ensuring smooth data exchange between services. The two most widely used technologies are Apache Kafka and RabbitMQ, which, despite having similar purposes, are designed with different architectural concepts [9], [10]. Kafka is known as a distributed streaming platform capable of handling very large amounts of real-time data streams. Kafka works by storing data in topics and partitions, enabling parallel processing that makes it very fast and easy to scale [14]. Research shows that Kafka can achieve a throughput of up to 1 million messages per second [9], and in other tests, it was even able to receive around 420,000 messages per second on standard hardware [11]. However, Kafka also has weaknesses, especially when it has to read data from disk storage, which causes latency to increase significantly [10]. With these characteristics, Kafka is considered more suitable for needs such as log processing, event streaming, and real-time activity monitoring.

Unlike Kafka, RabbitMQ is a traditional message broker that uses the AMQP protocol to provide flexibility in message routing [10], [15]. RabbitMQ uses Exchange and Binding components to determine the direction of messages to specific queues, making it suitable for complex service communications such as request reply patterns. The main advantage of RabbitMQ lies in its low latency; research has proven that RabbitMQ is capable of maintaining latency below 10 ms under ideal conditions [10]. RabbitMQ is also known to be reliable in high failure conditions, making it more suitable for task scheduling or communication between services in microservice systems [9]. However, in terms of throughput, RabbitMQ tends to be lower than Kafka, especially in basic settings [10]. Several studies have also compared RabbitMQ with other brokers such as ActiveMQ, Artemis, Pulsar, and RocketMQ, but the focus is still limited to basic metrics such as CPU usage and latency [9], [13].

In assessing the performance of message brokers, researchers typically use two main metrics: throughput and latency. Throughput indicates how many messages can be processed in one second, while latency indicates the time it takes for a message to travel from the sender to the recipient [16], [17]. In addition to performance, another important aspect is scalability, which is the ability of a system to continue to run well even when the load increases [18]. Modern systems typically rely on a horizontal scaling approach by adding more brokers to distribute the load, and its effectiveness can be seen from the stability of CPU and RAM usage after adding nodes [19]. Research on the scalability of stream processing systems also shows that most modern frameworks can work almost linearly as long as cloud resources are sufficient [12]. However, this research does not specifically evaluate the behavior of message brokers under extreme load conditions that are common in microservice ecosystems.

Unlike previous studies, this study provides a more comprehensive contribution by conducting a multi-scenario analysis that not only measures the basic performance of Kafka and RabbitMQ, but also assesses their behavior in scalability and stress tests or burst tests. Extreme load testing, such as sending 100,000 messages in 10 seconds, was conducted to simulate traffic spikes that often occur in real-world applications, for example, when many users access the service simultaneously [20], [21], [22]. The four-scenario approach used variations in message size, number of messages, number of consumers, and extreme load testing provides more in-depth and relevant analysis results for modern system needs, thereby addressing the shortcomings of previous studies that tended to focus on only one or two metrics [23].

3. Methodology

This section describes the methods used, starting from the experimental design that contains the scenarios used, the implementation configuration that describes the process of running both message broker technologies, and the measurement metrics used to evaluate the technologies that this research compares, as shown in Fig. 1.

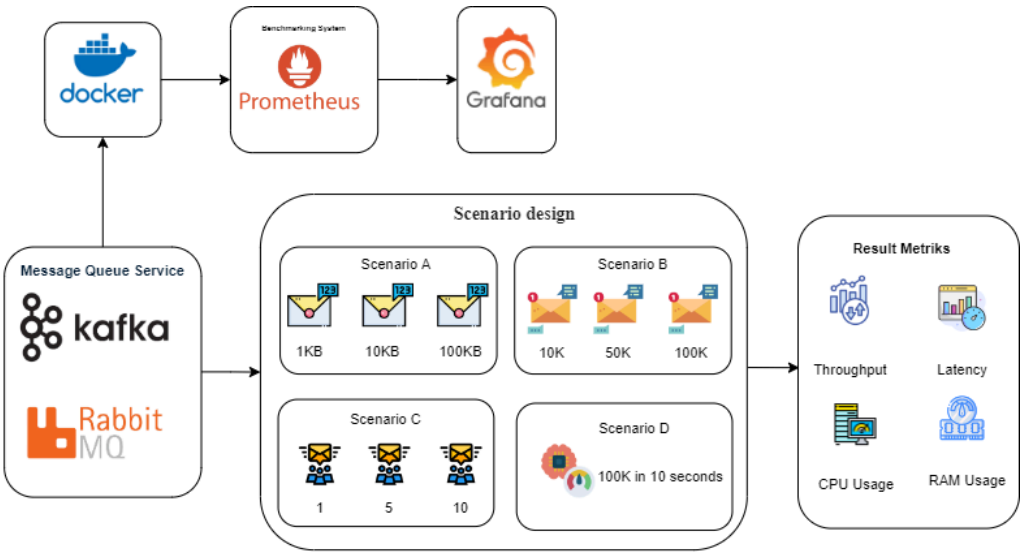


Fig. 1: Block experiment scenario.

3.1. Experiment

This research uses a comparative experimental approach to evaluate the performance characteristics of Apache Kafka and RabbitMQ. Four different test scenarios were used to analyse the response of both message queue services under varying workload conditions. The following are the experimental scenarios used in this research:

1. Scenario A
   Message Size Variation Test. This scenario specifically aims to measure the impact of message payload size variation on system throughput, message delivery latency, and computing resource consumption. The test was conducted by sending 10,000 messages with three size variations: 1 KB, 10 KB, and 100 KB
2. Scenario B
   Message Volume Variation Test. The main focus of this scenario is to analyse in depth how significantly increasing the message volume affects the system throughput, relaying latency, and resource usage by the message broker. The test was conducted by sending 10 KB messages in three variations of number: 10,000, 50,000, and 100,000 messages.
3. Scenario C
   Consumer Scalability Test. This scenario was specifically designed to compare the efficiency and scalability of the two systems in distributing the workload to different numbers of consumers. The test was conducted by sending 50,000 10 KB messages that would be received by three different numbers of consumers: 1, 5, and 10 consumers.
4. Scenario D
   Stress Test. Aimed to measure the system's ability to handle burst traffic. The test was conducted by sending 100,000 10 KB messages with a target completion time of 10 seconds.

3.2. Experiment Configuration and Test Procedure

The testing process of this research is run in a virtualised environment, orchestrated using Docker Compose. The host system used was Ubuntu 20.04 LTS, Windows 11, running Docker Desktop 4.26.1 and Docker Compose v2.24.1. Each message queue service was deployed on an identical VM with an Intel Core i7 CPU, 16 GB RAM, and 512 GB HDD. The environment includes a three-broker Apache Kafka cluster (confluentinc/cp-kafka:7.3.2) with ZooKeeper, and a three-node RabbitMQ cluster (rabbitmq:3.11-management) interconnected. Monitoring is performed by Prometheus (prom/prometheus), which collects metrics from the Kafka broker and RabbitMQ nodes every 15 seconds via JMX Exporter, with data visualisation in Grafana.

The test implementation was developed using Python 3, utilising the kafka-python library for Kafka, pika for RabbitMQ, psutil for resource monitoring, numpy for cy-latent statistical analysis, json for message serialisation,

and threading for parallel execution. Custom scripts (common.py, kafka_test.py, rabbitmq_test.py, run_tests.py, start_experiment.py) manage the test logic, including creation/deletion of unique topics/queues, message delivery with acknowledgement (acks= 'all' for Kafka) or persistence (delivery_mode=2 for RabbitMQ) and timestamp, and message consumption with prefetch_count=1 setting for RabbitMQ.

The test procedure was performed 10 times at identical settings to reduce bias. After initialisation of the Docker Compose environment and a 180-second wait phase, the start_experiment.py script verified connectivity to all brokers. The run_tests.py script then runs each scenario sequentially; for each run, a unique topic/queue is created and cleared. Producers send messages and consumers process them, with MetricCollector recording latency, throughput, CPU, and RAM in situ. Results are collected in benchmark_results.json. After each experiment, the JVM was left to "cool down" for 10 minutes with restarts and idles to remove residual state. The Docker environment was then thoroughly stopped unless requested to remain active.

3.3. Measurement Metrics

To conduct the performance evaluation, the performance metrics used in this study aim to provide a comprehensive overview of the performance of the two message brokers, which include:

1. Throughput (Messages/second)
   Measures the rate at which messages are successfully sent by the producer and confirmed per second. It serves as a key indicator of the message processing capacity of a system.
2. Latency (Milliseconds)
   Measures the delay time it takes for a message to travel from the point of delivery by the producer to the point of receipt by the consumer (end-to-end latency). Lower latency indicates more responsive and efficient communication.
3. CPU Usage (Percentage)
   This metric measures the CPU and memory utilisation on the machine while running the test script (client). It refers to the average percentage utilisation of the Central Processing Unit by the message broker and client (producer/consumer) processes during the test period. This metric provides a deep insight into the efficient use of computing resources.
4. RAM Usage (Percentage)
   Shows the average percentage of Random-Access Memory utilisation by the message broker and client processes during the experiment. This metric effectively reflects the efficiency and effectiveness of system memory management.

4. Results and Evaluation

In the results of experiments and performance evaluations of two message broker systems, namely Apache Kafka and RabbitMQ, based on three main aspects of performance, scalability, stress test. The evaluation was carried out based on a series of test scenarios that had been carried out using the Producer-Consumer communication model. Where the experiment is carried out in the Docker environment, the parameters observed in each experiment include throughput, latency, CPU and memory. Tests were conducted for each system with variations in the number of messages, number of consumers, burst (message delivery within 10s) and message size to measure the scalability and reliability of each system.

4.1. Scenario A

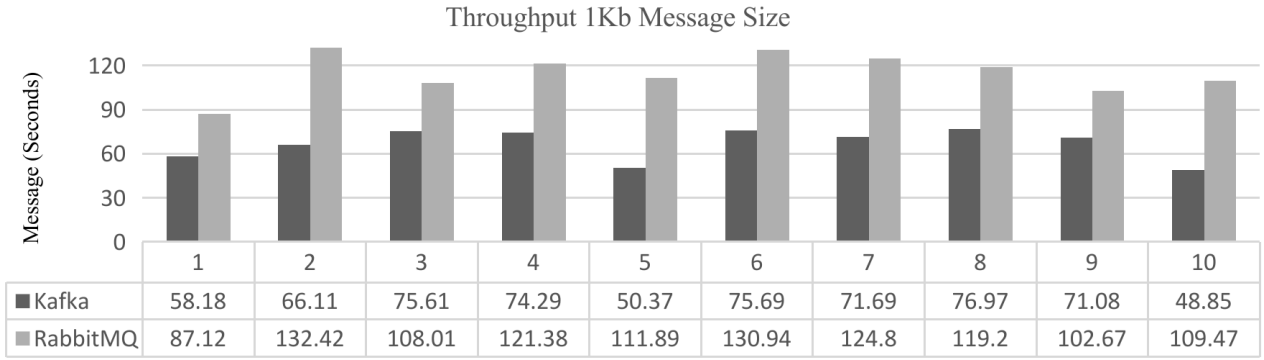
The results of the evaluation of scenario A include an experiment of message measurement with the aim of measuring performance based on the size of the payload, where the variables used are message sizes of 1KB, 10KB and 100KB in the same number of messages, namely 100,000 messages.

4.1.1. Message Size Evaluation Result: 1KB

1. Throughput

   Experiments showing throughput with a message size of 1KB showed that RabbitMQ consistently has much higher throughput than Kafka for 1KB messages, with RabbitMQ reaching peaks of up to 132 msgs/

sec compared to Kafka's only 77 msgs/sec. RabbitMQ processes 1.5 to 2 times more messages per second on average, indicating higher efficiency for small message queue use cases than Kafka, as shown in Fig. 2.

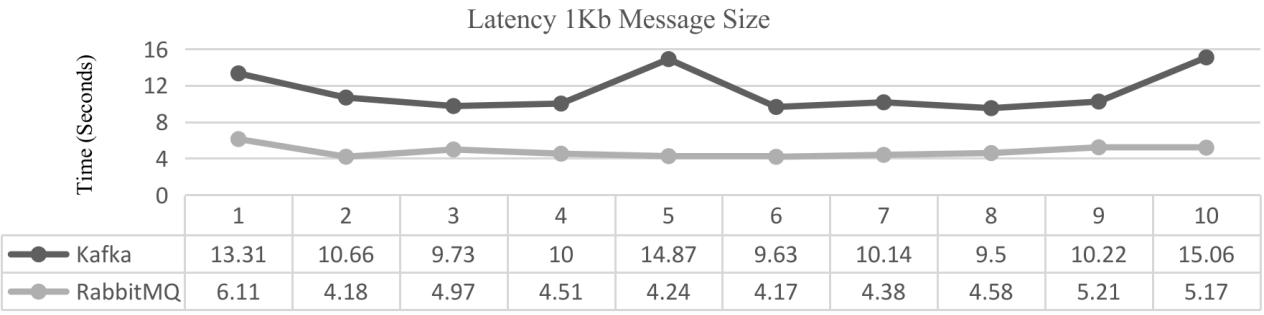


	1	2	3	4	5	6	7	8	9	10
Kafka	58.18	66.11	75.61	74.29	50.37	75.69	71.69	76.97	71.08	48.85
RabbitMQ	87.12	132.42	108.01	121.38	111.89	130.94	124.8	119.2	102.67	109.47

Fig. 2: Result experiments throughput (message/seconds) 1KB.

2. Latency

In this measurement, latency with a message size of 1KB showed that RabbitMQ consistently had much lower and more stable latency than Kafka for 1KB messages across all 10 tests. RabbitMQ latency generally ranges from 4-6 seconds, while Kafka shows higher latency, often 2 to 3 times slower, with values ranging from 9.5 to 15.06 seconds, as shown in Fig. 3.

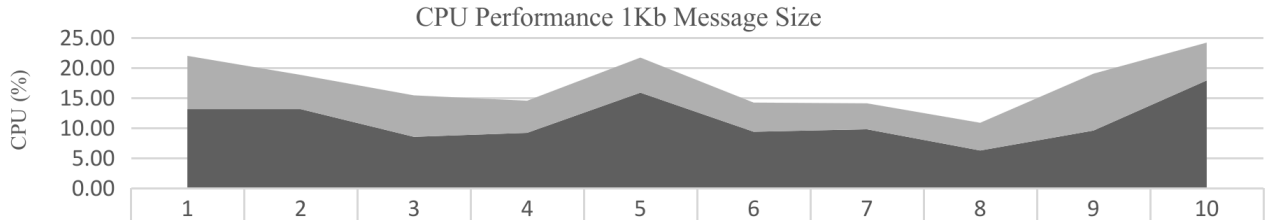


	1	2	3	4	5	6	7	8	9	10
Kafka	13.31	10.66	9.73	10	14.87	9.63	10.14	9.5	10.22	15.06
RabbitMQ	6.11	4.18	4.97	4.51	4.24	4.17	4.38	4.58	5.21	5.17

Fig. 3: Result experiment latency (Seconds) 1KB.

3. CPU

CPU usage for 1KB message size shows Kafka consistently uses higher CPU than RabbitMQ for 1KB messages, with 10 tests. Kafka reaches a peak CPU usage of 17.86% (test 10), while RabbitMQ only reaches 9.44% (test 9). This indicates that Kafka generally requires about 1.5 to 2 times more CPU, these values can be seen in Fig. 4.



	1	2	3	4	5	6	7	8	9	10
RabbitMQ	8.84	5.68	6.88	5.33	5.82	4.84	4.31	4.60	9.44	6.28
Kafka	13.10	13.10	8.50	9.16	15.83	9.34	9.75	6.23	9.54	17.86

Fig. 4: Result experiments CPU (%) 1KB.

4. RAM

RAM usage in Fig. 5 shows that RabbitMQ consistently used more RAM than Kafka for 1KB messages across all 10 tests. RabbitMQ operates with RAM usage between 82%-95%, while Kafka ranges from 71%

Table 1: Message size comparison results 1 KB.

Metric	Measurement Type	Unit	Kafka	RabbitMQ
Throughput	Average	msg/s	66.88	114.79
	Standard Deviation	msg/s	10.67	13.86
Latency	Average	ms	11.31	4.75
	Standard Deviation	ms	2.21	0.42
CPU	Average	%	11.24	6.20
	Standard Deviation	%	3.61	1.56
RAM	Average	%	76.92	88.90
	Standard Deviation	%	4.77	4.27

to 86%. This indicates that RabbitMQ has significantly higher RAM requirements, around 10-20% higher than Kafka, making it an important factor to consider if memory availability is a major constraint in the infrastructure.

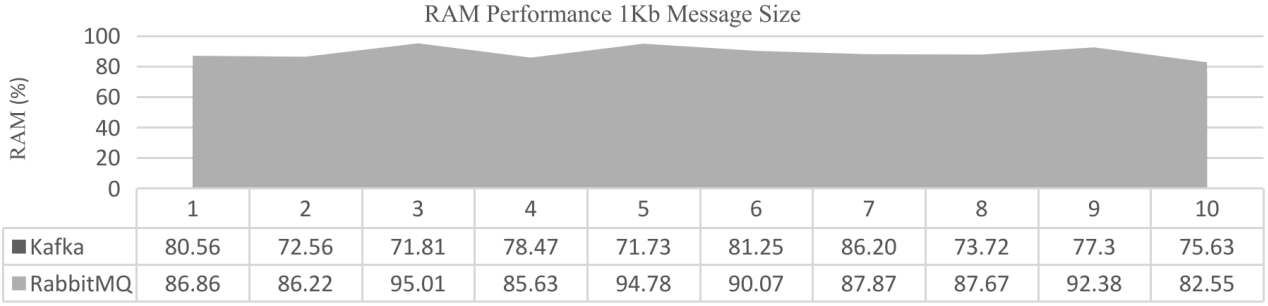


Fig. 5: Result experiments RAM (%) 1KB.

The comparison of performance between Kafka and RabbitMQ for sending 1 KB messages shows significant differences in most metrics whose values are shown in Table 1 below. In general, RabbitMQ demonstrates superior performance in core processing speed and efficiency. In terms of performance metrics, RabbitMQ recorded a much higher average throughput (114.79 msg/s) compared to Kafka (66.88 msg/s), indicating its ability to process nearly twice the number of messages per second. RabbitMQ's superiority is even more pronounced in the Latency metric, where the average delay time is only 4.75 ms, much lower than the 11.31 ms achieved by Kafka. The stability of these results is also supported by a very low Standard Deviation for RabbitMQ (0.42 ms), indicating that its latency is highly consistent and has minimal fluctuations compared to Kafka (2.21 ms).

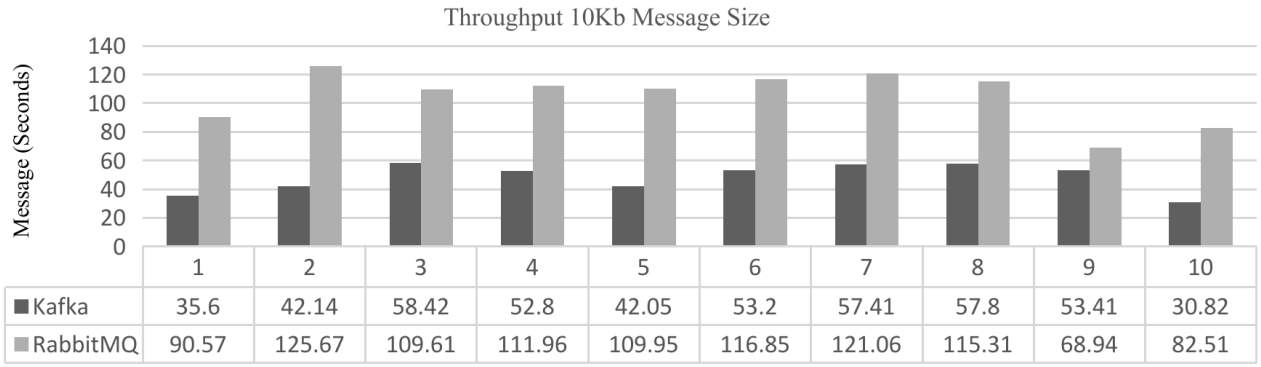
In terms of resource usage, RabbitMQ maintains operational efficiency with lower average CPU usage (6.20% compared to 11.24% on Kafka). RabbitMQ's CPU usage consistency is also better (1.56) than Kafka (3.61). This indicates that RabbitMQ processes 1 KB messages with a lighter and more stable computational load. However, the only metric where Kafka excels is average RAM usage, at 76.92%, which is more efficient than RabbitMQ's 88.90%. Based on the analysis of the average values and Standard Deviation, it can be concluded that RabbitMQ is the more optimal choice for 1 KB message delivery scenarios where the priorities are high speed (Throughput), low latency, and CPU efficiency. The substantial differences in average Latency and CPU, combined with RabbitMQ's much smaller Standard Deviation, provide strong evidence that this performance advantage is statistically significant and stable, not just random variation.

4.1.2. Message Size Evaluation Results: 10KB

1. Throughput

Experimental results with a message size of 10KB show that RabbitMQ consistently has a much higher throughput than Kafka in all tests. RabbitMQ peaked at 125 msgs/sec (test 2) and 121 msgs/sec (test 7), while

Kafka only peaked at around 58 msgs/sec, indicating that RabbitMQ is able to process around 2 to 3 times more messages per second, as shown in Fig. 6.

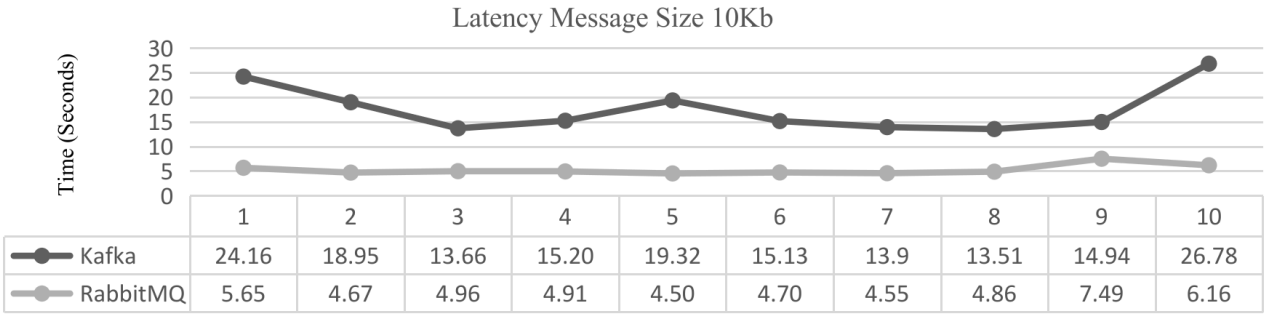


	1	2	3	4	5	6	7	8	9	10
Kafka	35.6	42.14	58.42	52.8	42.05	53.2	57.41	57.8	53.41	30.82
RabbitMQ	90.57	125.67	109.61	111.96	109.95	116.85	121.06	115.31	68.94	82.51

Fig. 6: Experiment results throughput (msg/s) 10KB.

2. Latency

The 10KB message size latency graph shows RabbitMQ consistently has lower and more stable latency than Kafka for 10KB messages across all tests. RabbitMQ's latency generally ranges from 4.50-7.49 seconds, while Kafka's is much higher, often 3 to 5 times, with significant spikes up to 26.78 seconds (test 10), as shown in Fig. 7.

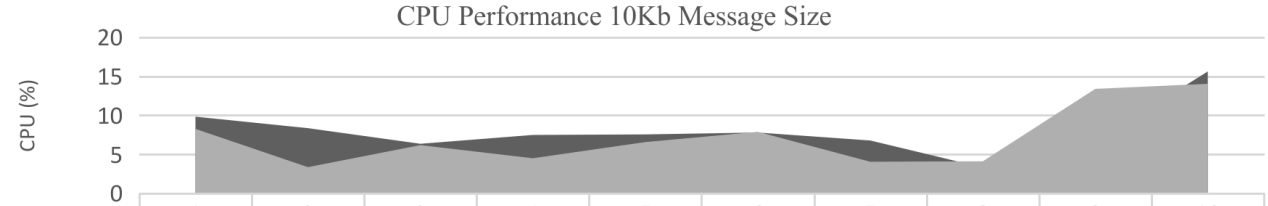


	1	2	3	4	5	6	7	8	9	10
Kafka	24.16	18.95	13.66	15.20	19.32	15.13	13.9	13.51	14.94	26.78
RabbitMQ	5.65	4.67	4.96	4.91	4.50	4.70	4.55	4.86	7.49	6.16

Fig. 7: Experiment result latency (s) 10KB.

3. CPU

CPU usage in the graph shown in Fig. 8, for Kafka and RabbitMQ on 10KB messages, where both systems show fluctuations. Although there is no consistent dominance, Kafka reaches a peak CPU usage of up to 15.61% (test 10), while RabbitMQ reaches 14.03% (test 10).



	1	2	3	4	5	6	7	8	9	10
Kafka	9.82	8.35	6.35	7.48	7.55	7.79	6.77	3.31	6.58	15.61
RabbitMQ	8.26	3.36	6.18	4.50	6.57	7.88	4.05	4.11	13.38	14.03

Fig. 8: Experiment result CPU (%) 10KB.

4. RAM The RAM usage shown in Fig. 9 indicates that RabbitMQ consistently shows higher RAM requirements for 10KB messages, fluctuating between 84% to 93%. Compare this to Kafka, which is generally in the range of 71% to 86%, implying that Kafka is slightly more efficient in memory management at this load. This

Table 2: Comparison results of 10 KB message size.

Metric	Measurement Type	Unit	Kafka	RabbitMQ
Throughput	Average	msg/s	105.24	48.37
	Standard Deviation	msg/s	9.94	18.38
Latency	Average	ms	5.25	17.56
	Standard Deviation	ms	4.68	0.95
CPU	Average	%	7.23	7.96
	Standard Deviation	%	3.16	3.79
RAM	Average	%	88.39	78.93
	Standard Deviation	%	5.38	3.56

significant difference, where RabbitMQ uses about 10-20% more RAM, highlights the importance of careful memory allocation in the infrastructure when choosing a messaging broker.

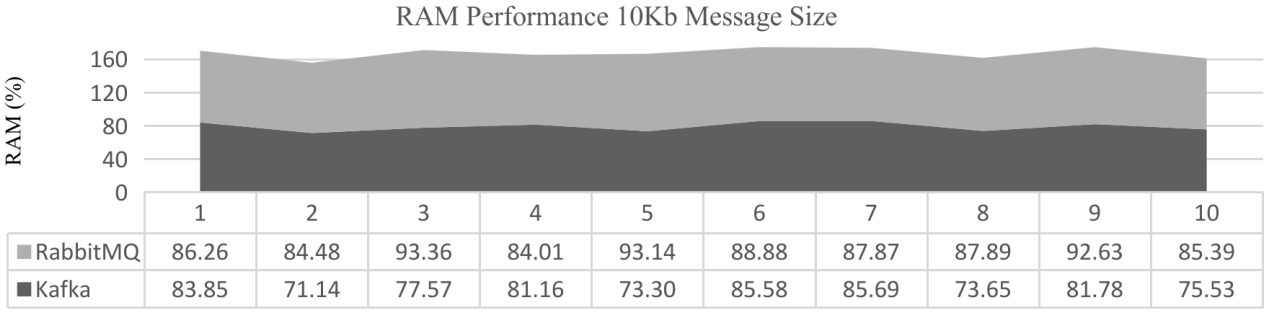


Fig. 9: Experiment results RAM (%) 10KB.

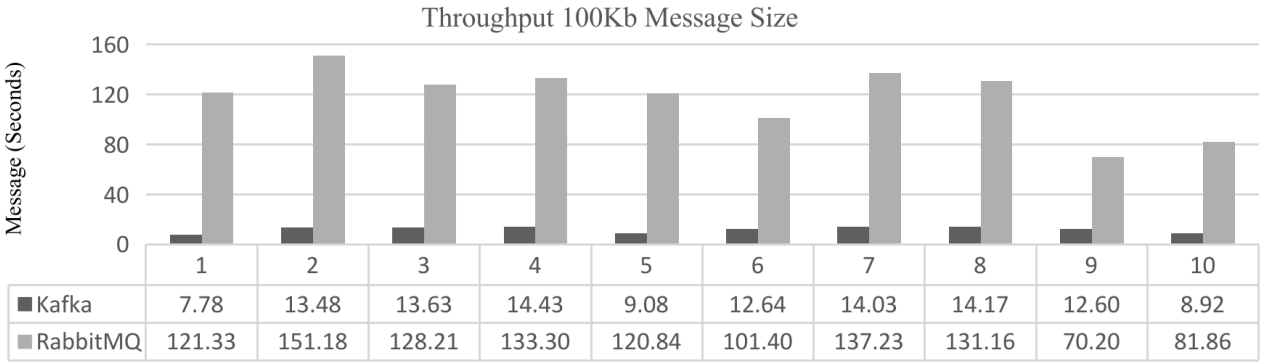
The performance comparison of Kafka and RabbitMQ by sending 10KB of data is summarised in Table 2. Kafka shows superiority in throughput (105.24 msgs/s vs. 48.37 msgs/s RabbitMQ) and latency (5.25 ms vs. 17.56 ms RabbitMQ), marking a shift in the previous performance dominance with 1KB messages. Although Kafka used slightly more CPU (7.23% vs. 7.96% RabbitMQ), RabbitMQ was more efficient in RAM usage (78.93% vs. 88.39% Kafka). This indicates that Kafka is more effective in processing larger batches of data, while RabbitMQ remains more memory efficient. The performance comparison between Kafka and RabbitMQ for sending 10 KB messages shows that Kafka is the more optimal choice for this medium workload, dominating key performance metrics. Kafka recorded a significantly higher average throughput (105.24 msg/s), more than double the speed of RabbitMQ (48.37 msg/s). This advantage is reinforced by Kafka's very low average latency (5.25 ms), which is far superior to RabbitMQ (17.56 ms).

Analyzing stability through Standard Deviation (SD), Kafka's throughput is more stable (SD 9.94 msg/s) than RabbitMQ (SD 18.38 msg/s). Meanwhile, RabbitMQ's latency shows very high consistency (SD 0.95 ms) despite its high average. The significant difference in average latency indicates that Kafka's performance is superior, and this difference is likely statistically significant. In terms of resource usage, the average CPU usage of both is relatively balanced (Kafka: 7.23%; RabbitMQ: 7.96%), where Kafka achieves a much higher throughput with a slightly lower computational load. Conversely, RabbitMQ is more efficient in average RAM usage (78.93%) compared to Kafka (88.39%), with slightly higher consistency (SD 3.56% vs. 5.38% for Kafka).

4.1.3. Message Size Evaluation Results: 100KB

1. Throughput

The throughput of 100KB message size shows RabbitMQ achieving values up to 151 msgs/sec (test 2) while Kafka is only 14 msgs/sec (test 4), as shown in Fig. 10.

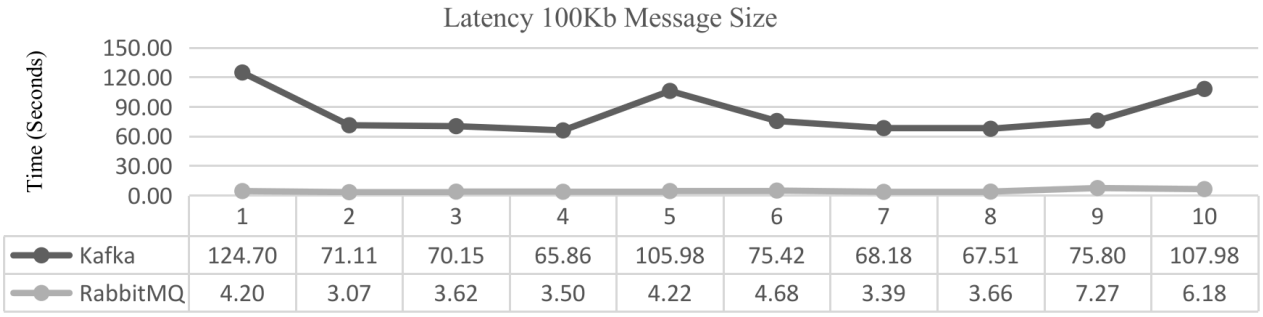


	1	2	3	4	5	6	7	8	9	10
Kafka	7.78	13.48	13.63	14.43	9.08	12.64	14.03	14.17	12.60	8.92
RabbitMQ	121.33	151.18	128.21	133.30	120.84	101.40	137.23	131.16	70.20	81.86

Fig. 10: Experiment results throughput (msg/s) 100KB.

2. *Latency*

Based on the latency graph of 100KB message size, it can be seen that RabbitMQ consistently shows very low latency between 3 and 7 seconds. In contrast, Kafka is much higher and unstable, jumping up to 124.70 seconds in the first test and consistently 10 to 30 times slower, as shown in Fig. 11.

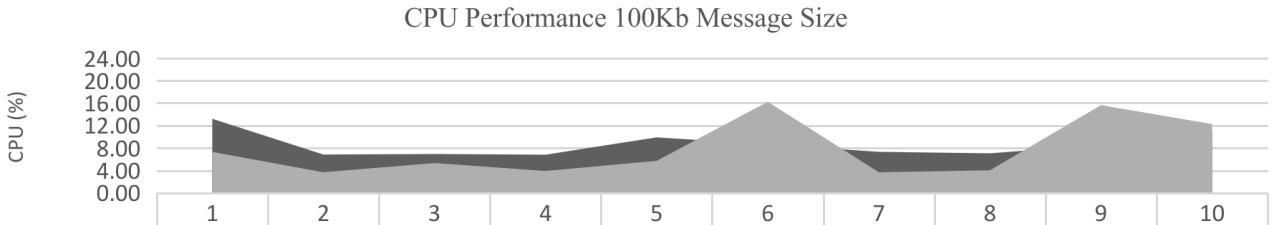


	1	2	3	4	5	6	7	8	9	10
Kafka	124.70	71.11	70.15	65.86	105.98	75.42	68.18	67.51	75.80	107.98
RabbitMQ	4.20	3.07	3.62	3.50	4.22	4.68	3.39	3.66	7.27	6.18

Fig. 11: Experiment result latency (s) 100KB.

3. *CPU*

The graph in Fig. 12 below shows that no message broker is consistently more efficient in CPU usage. Kafka and RabbitMQ both show significant fluctuations. Kafka reaches a peak of 13.20% (test 1), while RabbitMQ jumps up to 16.23% (test 6) and 15.63% (test 9). These sharp fluctuations indicate that 100KB messages substantially increase the computational load on both brokers, thus requiring adequate CPU allocation.



	1	2	3	4	5	6	7	8	9	10
Kafka	13.20	6.84	6.94	6.81	9.90	8.65	7.32	7.06	8.57	11.03
RabbitMQ	7.31	3.69	5.33	3.92	5.72	16.23	3.69	4.04	15.63	12.27

Fig. 12: Experiment result CPU (%) 100KB.

4. *RAM*

The RAM usage in Fig. 13 shows that both message brokers are very RAM-intensive for a 100KB message, with RabbitMQ consistently using more, fluctuating between 85% to 95%. Kafka also uses high RAM, in the range of 81% to 91%, but is slightly more efficient than RabbitMQ. The high RAM consumption

Table 3: Comparison result of 100 KB message size.

Metric	Measurement Type	Unit	Kafka	RabbitMQ
Throughput	Average	msg/s	12.08	117.67
	Standard Deviation	msg/s	2.50	25.53
Latency	Average	ms	83.27	4.38
	Standard Deviation	ms	21.24	1.35
CPU	Average	%	8.63	7.78
	Standard Deviation	%	2.16	5.01
RAM	Average	%	86.46	88.54
	Standard Deviation	%	3.27	3.92

of both brokers underscores the importance of allocating sufficient memory to handle large data loads, as it can affect the occurrence of performance bottlenecks.

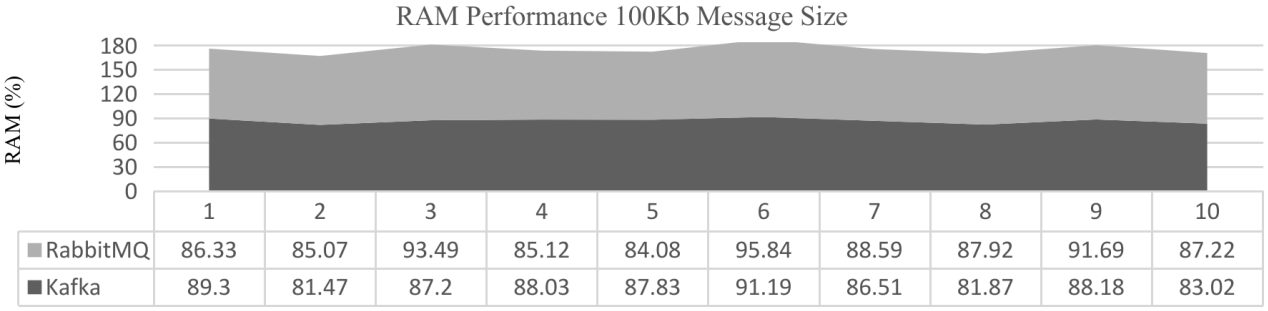


Fig. 13: Experiment results RAM (%) 100KB.

The performance comparison, whose values are shown in Table 3 below between Kafka and RabbitMQ for sending 100 KB messages shows that RabbitMQ clearly outperforms Kafka in core performance metrics, confirming Kafka's performance limitations when handling larger data loads in this scenario. In terms of performance metrics, RabbitMQ recorded a far superior average throughput (117.67 msg/s), nearly ten times the speed of Kafka (12.08 msg/s). RabbitMQ's superiority is even more striking in its very low average latency (4.38 ms), compared to Kafka's very high latency (83.27 ms). Analyzing stability through Standard Deviation (SD) further reinforces these findings.

RabbitMQ's latency is very stable (SD 1.35 ms), while Kafka shows very high volatility (SD 21.24 ms), indicating that latency on Kafka is highly unpredictable. Kafka's throughput is very low (SD 2.50 msg/s) compared to RabbitMQ (SD 25.53 msg/s), but Kafka's very low average figure outweighs the significance of its SD. In terms of resource usage, the average CPU usage of both is relatively balanced (RabbitMQ: 7.78%; Kafka: 8.63%), although RabbitMQ is capable of achieving significantly higher throughput with slightly lower CPU consumption. Meanwhile, the RAM usage of both is also very similar (RabbitMQ: 88.54%; Kafka: 86.46%). Overall, for 100 KB messages, RabbitMQ provides superior performance in terms of delivery speed (throughput) and low latency, with a very substantial advantage. Based on the very large average difference and RabbitMQ's superior stability, its advantage at this message size is statistically significant.

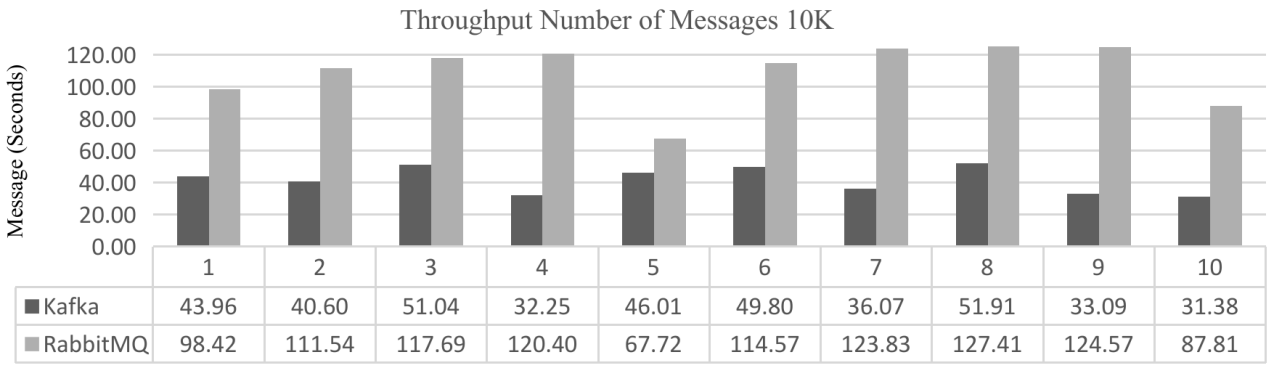
4.2. Scenario B

The evaluation results of scenario B include experimenting with the number of messages with the aim of seeing throughput limits & resource usage, where the variables used are the number of messages, 10K, 50K and 100K messages in the same message size of 1KB.

4.2.1. Evaluation Results Number of Messages: 10K

1. Throughput

The results showed RabbitMQ outperformed Kafka in throughput on tests with 10,000 messages. RabbitMQ peaked at 127 msgs/sec (test 8) and 124 msgs/sec (test 9), while Kafka only ranged from 31-51 msgs/sec. RabbitMQ processes about 2 to 3 times more messages per second, as shown in Fig. 14.

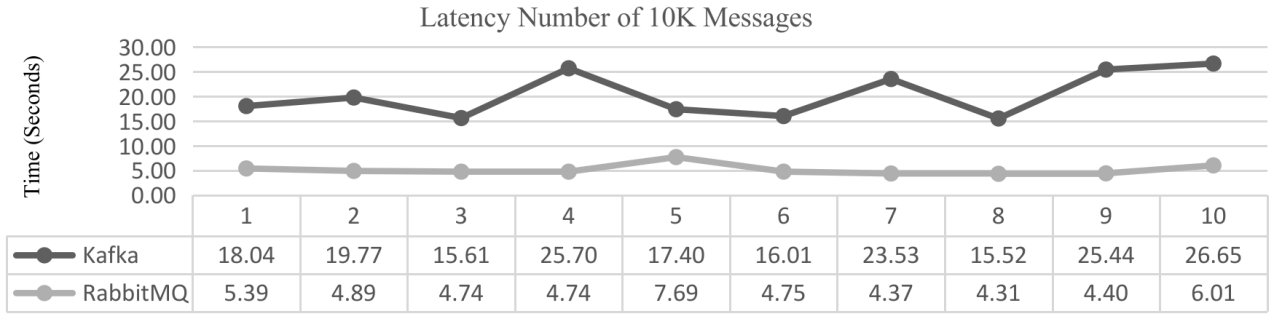


	1	2	3	4	5	6	7	8	9	10
Kafka	43.96	40.60	51.04	32.25	46.01	49.80	36.07	51.91	33.09	31.38
RabbitMQ	98.42	111.54	117.69	120.40	67.72	114.57	123.83	127.41	124.57	87.81

Fig. 14: Experiment results throughput (msg/s) number of messages 10K.

2. Latency

The resulting latency at 10K message count shows RabbitMQ outperforming Kafka in latency, with RabbitMQ maintaining very low delays between 4.31 and 7.69 seconds. In contrast, Kafka exhibited a much higher latency of 3 to 5 times, jumping up to 26.65 seconds by the 10th test. These results are shown in Fig.. 15 below.

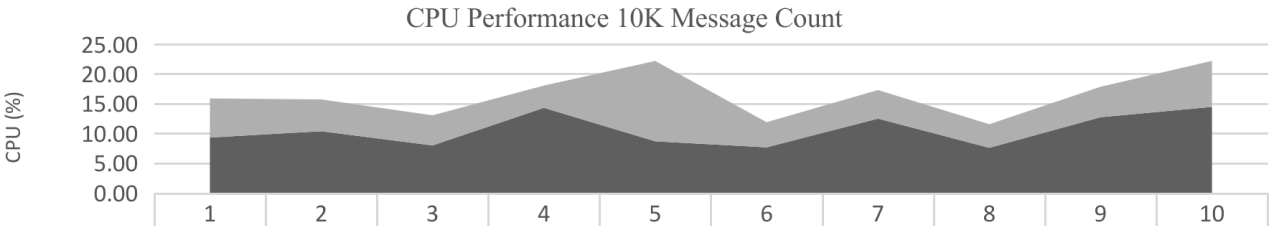


	1	2	3	4	5	6	7	8	9	10
Kafka	18.04	19.77	15.61	25.70	17.40	16.01	23.53	15.52	25.44	26.65
RabbitMQ	5.39	4.89	4.74	4.74	7.69	4.75	4.37	4.31	4.40	6.01

Fig. 15: Experiment results latency (seconds) number of messages 10K.

3. *CPU*

The CPU usage at 10K messages is shown in Fig. 16, showing Kafka has a higher CPU usage than RabbitMQ in the test with 10,000 messages. Kafka ranges from 7.60-14.47%, while RabbitMQ is at 3.74-13.49%. Kafka often uses 1.5 to 2.5 times more CPU, making RabbitMQ more efficient at CPU optimisation.



	1	2	3	4	5	6	7	8	9	10
RabbitMQ	6.56	5.34	5.06	3.74	13.49	4.22	4.80	3.96	5.11	7.72
Kafka	9.33	10.38	8.00	14.31	8.69	7.68	12.50	7.60	12.74	14.47

Fig. 16: Experiment result CPU (%) 10K message count.

4. RAM

Both message brokers allocate substantial RAM for 10,000 messages, but RabbitMQ consistently uses more, fluctuating between 81% to 96%. Kafka tends to be in the 75% to 89% range, as shown in Fig. 17. While

Table 4: Comparison result of 10K message count.

Metric	Measurement Type	Unit	Kafka	RabbitMQ
Throughput	Average	msg/s	41.61	109.40
	Standard Deviation	msg/s	8.05	19.17
Latency	Average	ms	20.37	5.13
	Standard Deviation	ms	4.51	1.04
CPU	Average	%	10.57	6.00
	Standard Deviation	%	2.72	2.89
RAM	Average	%	83.10	88.12
	Standard Deviation	%	5.41	4.38

both consume high amounts of RAM, RabbitMQ is slightly more voracious on memory usage, highlighting the importance of careful capacity planning for both brokers.

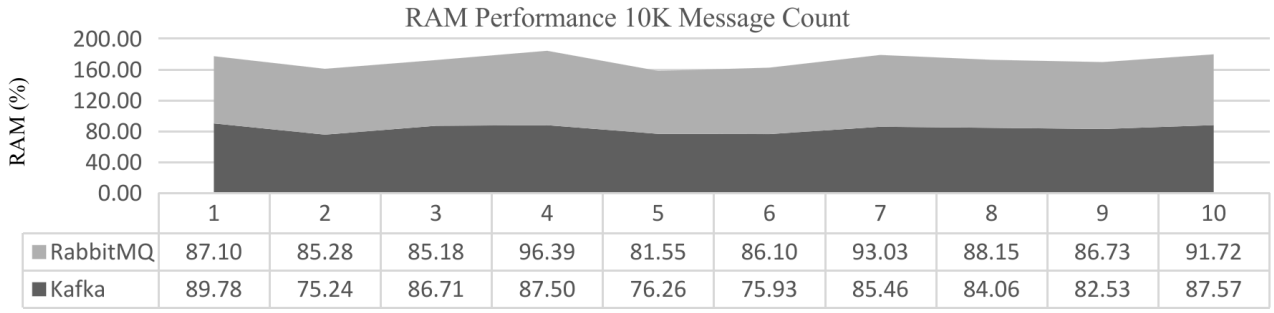


Fig. 17: Experiment result RAM (%) 10K message count.

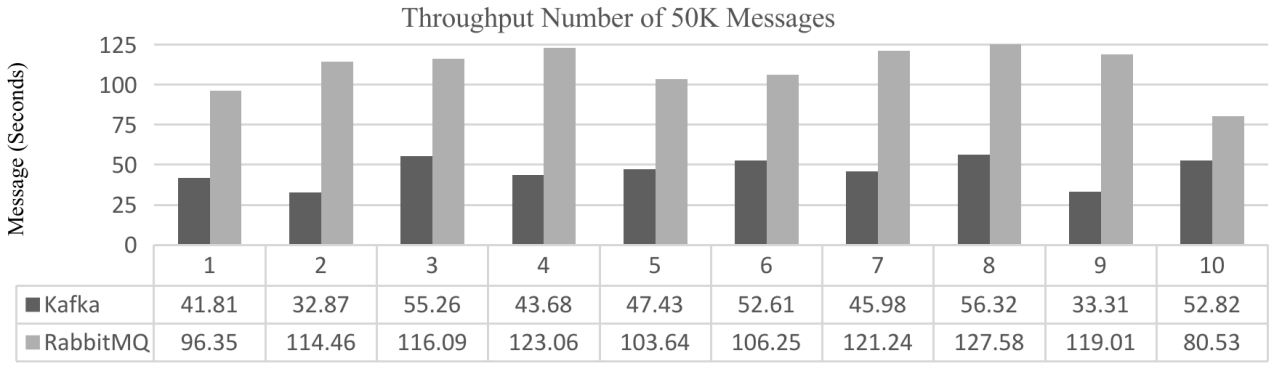
The results of comparing, whose values are shown in Table 4 below. The performance of Kafka and RabbitMQ based on a fixed message count scenario of 10,000 (10K) show that RabbitMQ provides superior performance in processing speed and CPU efficiency compared to Kafka. In terms of performance metrics, RabbitMQ recorded a significantly higher average throughput (109.40 msg/s), more than double the speed of Kafka (41.61 msg/s). This advantage is particularly striking in average latency, where RabbitMQ produced a very short delay time (5.13 ms), far below Kafka's latency (20.37 ms).

Analyzing stability through Standard Deviation (SD) further reinforces RabbitMQ's advantage in Latency. RabbitMQ's latency is very stable (SD 1.04 ms) compared to Kafka (SD 4.51 ms), demonstrating consistent and reliable latency results. Although RabbitMQ's throughput (SD 19.17 msg/s) is more variable than Kafka's (SD 8.05 msg/s), the significant difference in average throughput and latency suggests that RabbitMQ's performance advantage is likely statistically significant. In terms of resource efficiency, RabbitMQ excels in CPU usage, recording an average of 6.00% compared to 10.57% on Kafka, where the stability of CPU usage is relatively the same for both (SD $\sim 2.7 - 2.8$). Although Kafka is more efficient in average RAM usage (83.10% vs. 88.12% for RabbitMQ), RabbitMQ's advantages in throughput, latency, and CPU efficiency make it a more effective choice for scenarios measured by high message processing volumes.

4.2.2. Evaluation Results Number of Messages: 50K

1. Throughput

The results showed RabbitMQ outperformed Kafka in throughput on tests with 50,000 messages. RabbitMQ peaked at 127 msgs/sec (test 8) and 123 msgs/sec (test 4), while Kafka only ranged from 32-56 msgs/sec. RabbitMQ processed about 2 to 3 times more messages per second, as shown in Fig. 18.

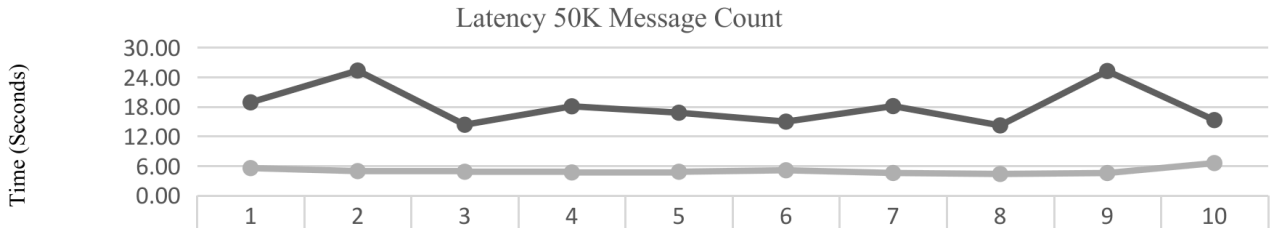


	1	2	3	4	5	6	7	8	9	10
Kafka	41.81	32.87	55.26	43.68	47.43	52.61	45.98	56.32	33.31	52.82
RabbitMQ	96.35	114.46	116.09	123.06	103.64	106.25	121.24	127.58	119.01	80.53

Fig. 18: Experiment results throughput (msg/s) number of messages 50K.

2. *Latency*

Latency generated with 50K messages, showing RabbitMQ generating low latency in the range of 4-7 seconds for 50,000 messages. On the other hand, Kafka shows 3 to 5 times longer. These results can be seen in Fig. 19 below.

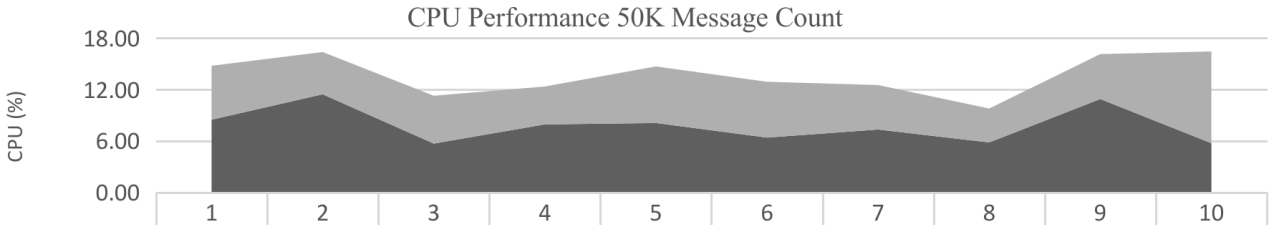


	1	2	3	4	5	6	7	8	9	10
Kafka	18.86	25.31	14.32	18.06	16.76	14.95	18.10	14.18	25.22	15.26
RabbitMQ	5.52	4.91	4.78	4.64	4.78	5.08	4.53	4.33	4.52	6.54

Fig. 19: Experiment results latency (s) number of messages 50K.

3. *CPU*

CPU usage on Kafka is higher than RabbitMQ on a test with 50,000 messages. Kafka ranged from 5.68-11.43%, while RabbitMQ was at 3.96-10.71%. Kafka uses 1.5 to 2 times more CPU, while RabbitMQ is more effective in CPU usage. These results can be seen in Fig. 20 below.



	1	2	3	4	5	6	7	8	9	10
RabbitMQ	6.29	4.93	5.58	4.42	6.60	6.51	5.19	3.96	5.24	10.71
Kafka	8.47	11.43	5.68	7.92	8.08	6.38	7.32	5.81	10.88	5.72

Fig. 20: Experiment results CPU (%) number of messages 50K.

4. RAM

RAM usage can be seen in Fig. 21. While both message brokers allocate substantial RAM for 50,000 messages, Kafka generally uses more, fluctuating between 86% to 95%. RabbitMQ tends to be in the 76% to 96% range, showing little efficiency. Kafka consistently had a slightly larger memory footprint across most tests.

Table 5: Comparison results of 50K message count.

Metric	Measurement Type	Unit	Kafka	RabbitMQ
Throughput	Average	msg/s	46.21	110.82
	Standard Deviation	msg/s	8.44	14.32
Latency	Average	ms	18.10	4.96
	Standard Deviation	ms	4.12	0.65
CPU	Average	%	7.77	5.94
	Standard Deviation	%	2.06	1.89
RAM	Average	%	89.98	86.84
	Standard Deviation	%	3.57	5.11

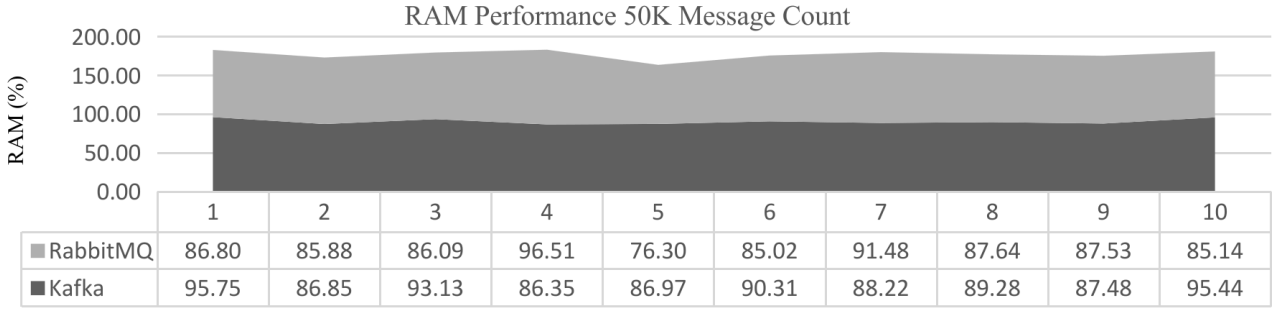


Fig. 21: Experiment results RAM (%) number of messages 50K.

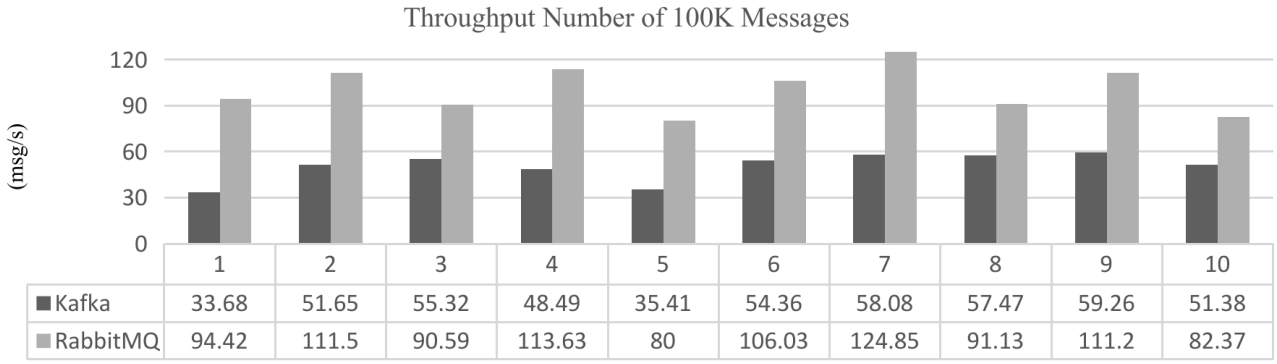
The results of comparing, whose values are shown in Table 5 below. The performance of Kafka and RabbitMQ based on a scenario of 50,000 (50K) messages consistently show that RabbitMQ maintains significantly superior performance in processing speed and CPU efficiency compared to Kafka. In terms of performance metrics, RabbitMQ recorded a much higher average throughput (110.82 msg/s), more than double the speed of Kafka (46.21 msg/s). This advantage is most significant in average latency, where RabbitMQ produces a very short delay time (4.96 ms), substantially lower than Kafka's latency (18.10 ms).

Analyzing stability through Standard Deviation (SD) further strengthens RabbitMQ. RabbitMQ's latency is very stable (SD 0.65 ms) compared to Kafka (SD 4.12 ms), which shows consistent and reliable latency results. Although RabbitMQ's throughput (SD 14.32 msg/s) is more variable than Kafka's (SD 8.44 msg/s), the significant difference in average latency and throughput suggests that RabbitMQ's performance advantage is likely statistically significant. In terms of resource efficiency, RabbitMQ outperforms Kafka in average CPU usage (5.94% vs. 7.77%), with very similar stability. Meanwhile, Kafka is slightly more efficient in average RAM usage (86.84% vs. 89.98% for RabbitMQ), with better consistency (SD 3.57% vs. 5.11%). However, RabbitMQ's significant advantages in latency, throughput, and CPU efficiency make it a more effective choice for high-volume message processing scenarios.

4.2.3. Evaluation Results Message Quantity: 100K

1. Throughput

The results show that RabbitMQ has a greater throughput than Kafka by almost two orders of magnitude, as shown in Fig. 22. Where RabbitMQ consistently outperforms Kafka in terms of throughput by an average of 100 and 50, RabbitMQ's stability is quite good despite minor fluctuations in the 5th and 10th experiments.

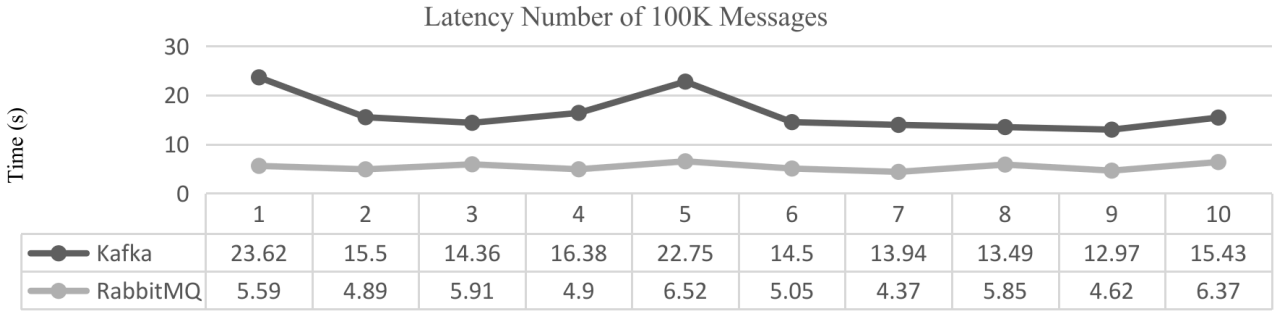


	1	2	3	4	5	6	7	8	9	10
Kafka	33.68	51.65	55.32	48.49	35.41	54.36	58.08	57.47	59.26	51.38
RabbitMQ	94.42	111.5	90.59	113.63	80	106.03	124.85	91.13	111.2	82.37

Fig. 22: Experiment results throughput (msg/s) number of 100K messages.

2. *Latency*

RabbitMQ not only has higher transmission efficiency but also lower latency than Kafka, as shown in Fig. 23. RabbitMQ consistently has much lower latency than Kafka, only around 4-6 seconds, while Kafka ranges from 13-23 seconds. Kafka experienced larger fluctuations, especially in the 1st (23.62 s) and 5th (22.75 s) experiments. Therefore, RabbitMQ is more stable and efficient in terms of message completion time.

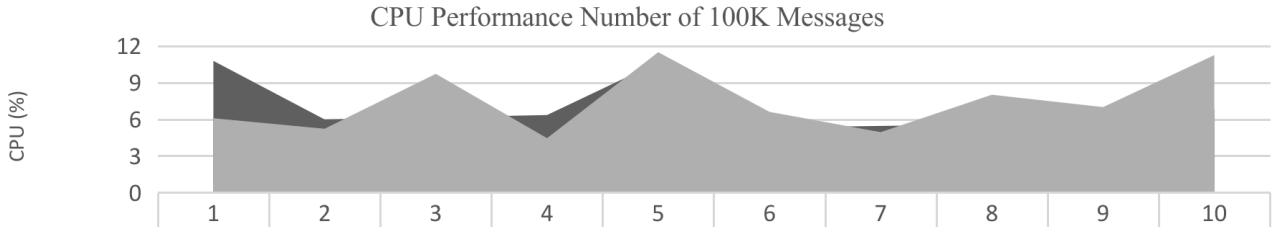


	1	2	3	4	5	6	7	8	9	10
Kafka	23.62	15.5	14.36	16.38	22.75	14.5	13.94	13.49	12.97	15.43
RabbitMQ	5.59	4.89	5.91	4.9	6.52	5.05	4.37	5.85	4.62	6.37

Fig. 23: Experiment results latency (s) number of 100K messages.

3. CPU

CPU utilisation on Kafka and RabbitMQ does not differ much between the two systems, as shown in Fig. 24. On average, Kafka ranged from 4-10% with the highest experiments being on the 1st (10.77%) and 5th (10.66%). Whereas in RabbitMQ, the CPU fluctuation is greater, ranging from 4.45% to 11.48%. The 5th (11.48%) and 10th (11.24%) experiments show high CPU load, which could indicate intensive processing during large buffers or burst traffic. So, in the CPU usage experiments with 100k messages, Kafka tends to be lower and consistent across multiple trials than RabbitMQ.



	1	2	3	4	5	6	7	8	9	10
Kafka	10.77	5.98	6.16	6.34	10.66	5.31	5.45	5.57	4.12	6.81
RabbitMQ	6.08	5.22	9.71	4.45	11.48	6.6	4.94	8	6.99	11.24

Fig. 24: Experiment results CPU (%) number of 100K messages.

4. *RAM*

The RAM usage is shown in Fig. 25. shows that RabbitMQ is slightly more RAM-efficient on average, but Kafka is more consistent and stable in RAM usage in the range of 83-96 MB, although it spikes in the 4th (95.57 MB) and 10th (91.93 MB) experiments. Kafka showed resource-aware characteristics, as its RAM

Table 6: Comparison results of 100K message count.

Metric	Measurement Type	Unit	Kafka	RabbitMQ
Throughput	Average	msg/s	50.51	117.67
	Standard Deviation	msg/s	9.05	14.92
Latency	Average	ms	16.29	4.38
	Standard Deviation	ms	3.77	0.75
CPU	Average	%	6.72	7.78
	Standard Deviation	%	2.23	2.56
RAM	Average	%	88.40	88.54
	Standard Deviation	%	3.55	5.78

usage did not go up and down in extremes. RabbitMQ was more variable than Kafka, especially in the 5th experiment, with a low of 73.13 MB and a high of 96.83 MB in the 4th experiment.

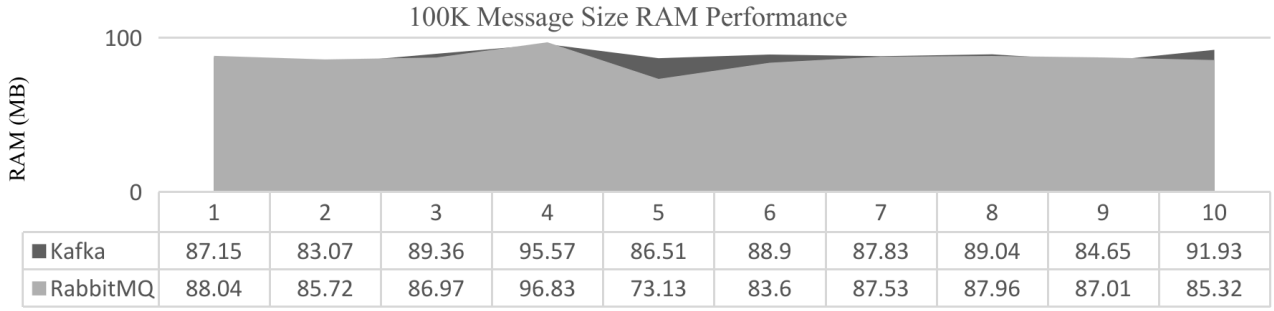


Fig. 25: Experiment results RAM (%) number of messages 100K.

The results of comparing, whose values are shown in Table 6 below. The performance of Kafka and RabbitMQ for a scenario with 100,000 (100K) messages consistently confirm that RabbitMQ maintains superior performance in processing speed and latency compared to Kafka. In terms of performance metrics, RabbitMQ recorded a significantly higher average throughput (117.67 msg/s), more than double the speed of Kafka (50.51 msg/s). RabbitMQ's advantage is particularly significant in average latency, resulting in a very low delay time (4.38 ms), compared to Kafka's much higher latency (16.29 ms).

Analyzing stability through Standard Deviation (SD) reinforces these findings on Latency. RabbitMQ latency is very stable (SD 0.75 ms) compared to Kafka (SD 3.77 ms), demonstrating consistent and reliable latency results. The large difference in average Latency and Throughput, despite Kafka's slightly better Throughput stability (SD 9.05 msg/s vs. 14.92 msg/s), suggests that RabbitMQ's performance advantage is likely statistically significant. In terms of resource efficiency, Kafka is slightly superior in average CPU usage (6.72% vs. 7.78% for RabbitMQ). However, Kafka achieves this CPU efficiency at the expense of significantly lower throughput. RAM usage for both is relatively similar (RabbitMQ: 88.54% vs. Kafka: 88.40%). Overall, RabbitMQ's significant advantages in Latency and Throughput make it a more effective choice for scenarios involving very high message volume processing.

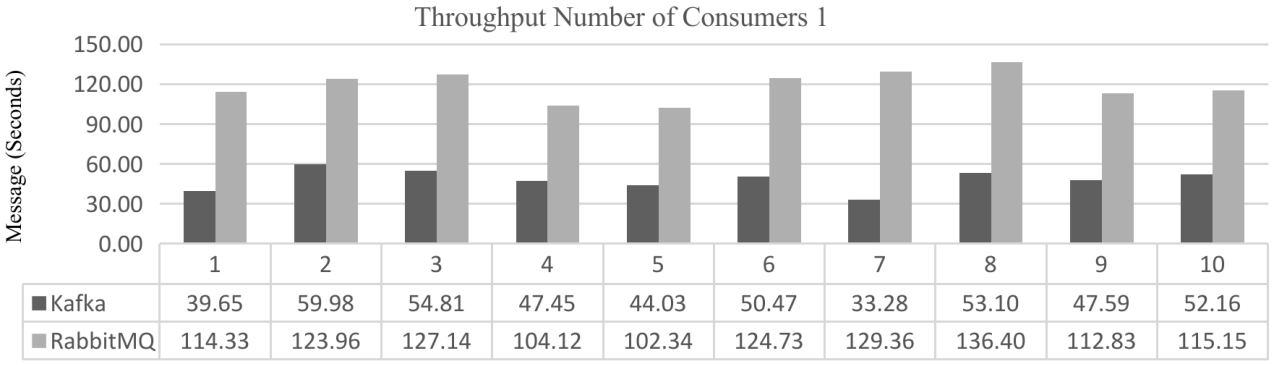
4.3. Scenario C

The evaluation results of scenario C include experiments on the number of consumers with the aim of measuring the scalability of the consumer system, where the variables used are the number of consumers, 1, 5 and 10, in the same number of messages, namely 100K messages with a size of 1KB.

4.3.1. Results of Evaluation of the Number of Consumers: 1

1. Throughput

The results showed that RabbitMQ outperformed Kafka in throughput on a single-consumer configuration. RabbitMQ peaked at 136 msgs/sec (test 8), while Kafka was only 60 msgs/sec (test 2), indicating RabbitMQ processes about 2 to 3 times more messages per second, as shown in Fig. 26.

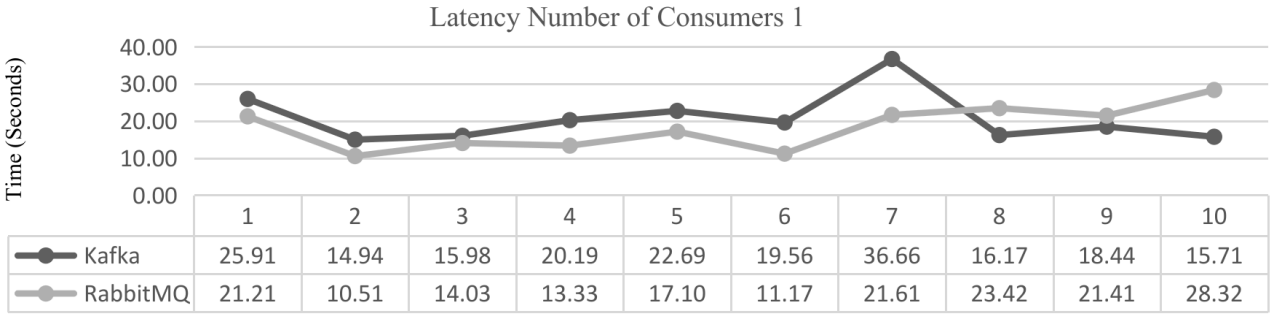


	1	2	3	4	5	6	7	8	9	10
Kafka	39.65	59.98	54.81	47.45	44.03	50.47	33.28	53.10	47.59	52.16
RabbitMQ	114.33	123.96	127.14	104.12	102.34	124.73	129.36	136.40	112.83	115.15

Fig. 26: Experiment results throughput (msg/s) number of consumers 1.

2. *Latency*

The latency of the number of consumers 1 shows that RabbitMQ generally has a lower and more stable latency than Kafka in the single-consumer configuration. RabbitMQ fluctuates between 10.51 and 28.32 seconds, while Kafka shows spikes of up to 36.66 seconds (test 7). RabbitMQ's generally 10-30% lower latency implies its efficiency for fast response with a single consumer, as shown in Fig. 27.

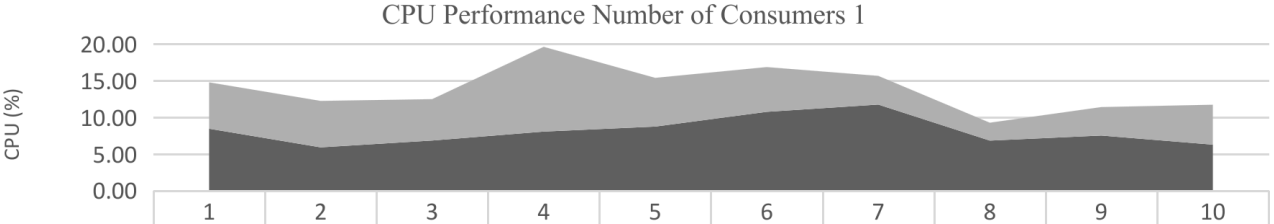


	1	2	3	4	5	6	7	8	9	10
Kafka	25.91	14.94	15.98	20.19	22.69	19.56	36.66	16.17	18.44	15.71
RabbitMQ	21.21	10.51	14.03	13.33	17.10	11.17	21.61	23.42	21.41	28.32

Fig. 27: Experiment results latency (s) number of consumers 1.

3. CPU

CPU performance at the number of consumers 1 indicates that Kafka generally has higher CPU usage than RabbitMQ in single-consumer configurations. Kafka ranges from 5.91-11.74%, while RabbitMQ is at 2.44-11.55%. Kafka often uses 1.5 to 2 times more CPUs. The results are shown in Fig. 28.



	1	2	3	4	5	6	7	8	9	10
RabbitMQ	6.32	6.33	5.64	11.55	6.64	6.10	3.91	2.44	3.88	5.44
Kafka	8.45	5.91	6.84	8.06	8.74	10.75	11.74	6.83	7.52	6.28

Fig. 28: CPU experiment results (%) number of consumers 1.

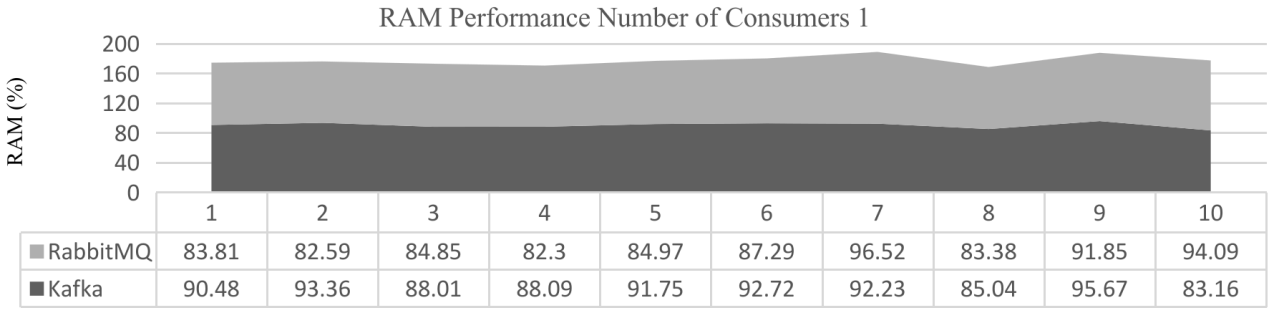
4. *RAM*

The RAM Performance of Number of Consumers 1 is shown in Fig. 29. Kafka generally has slightly higher RAM usage than RabbitMQ on a single consumer configuration. Kafka fluctuates between 83.16 MB

Table 7: Comparison results of the number of consumers 1.

Metric	Measurement Type	Unit	Kafka	RabbitMQ
Throughput	Average	msg/s	48.25	119.04
	Standard Deviation	msg/s	7.76	11.09
Latency	Average	ms	20.63	18.21
	Standard Deviation	ms	6.61	5.88
CPU	Average	%	8.11	5.83
	Standard Deviation	%	1.90	2.43
RAM	Average	%	90.05	87.17
	Standard Deviation	%	3.92	5.14

and 95.67 MB, while RabbitMQ ranges from 82.3 MB to 96.52 %. Kafka tends to use about 2% to 10% more RAM, indicating a slightly larger memory footprint even with a single consumer.



	1	2	3	4	5	6	7	8	9	10
RabbitMQ	83.81	82.59	84.85	82.3	84.97	87.29	96.52	83.38	91.85	94.09
Kafka	90.48	93.36	88.01	88.09	91.75	92.72	92.23	85.04	95.67	83.16

Fig. 29: RAM experiment results (%) number of consumers 1.

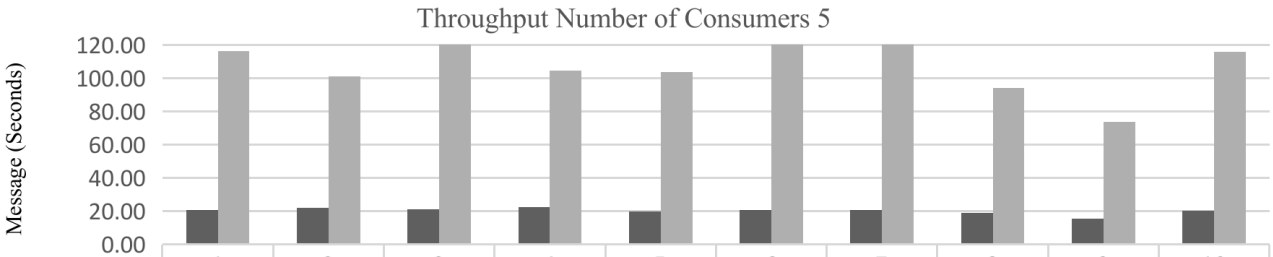
The results of comparing, whose values are shown in Table 7 below. The performance of Kafka and RabbitMQ in a single consumer scenario show that RabbitMQ provides superior performance in terms of processing speed and CPU efficiency compared to Kafka. In terms of performance metrics, RabbitMQ recorded a significantly higher average throughput (119.04 msg/s), more than double the speed of Kafka (48.25 msg/s). In terms of average latency, RabbitMQ is slightly better (18.21 ms) than Kafka (20.63 ms). Analyzing stability through Standard Deviation (SD), RabbitMQ shows better stability in Throughput (SD 11.09 msg/s vs. 7.76 msg/s in Kafka) and slightly better in Latency (SD 5.88 ms vs. 6.61 ms in Kafka). The large differences in average Throughput and CPU efficiency indicate that RabbitMQ's performance advantage is likely statistically significant.

In terms of resource efficiency, RabbitMQ outperforms Kafka in average CPU usage (5.83% vs. 8.11%), although Kafka's CPU consistency (SD 1.90%) is slightly better than RabbitMQ's (SD 2.43%). Meanwhile, RabbitMQ is slightly more economical in terms of average RAM usage (87.17% vs. 90.05% on Kafka). Overall, for scenarios where there is only one active consumer, RabbitMQ provides better performance in throughput, latency, and CPU efficiency, making it a more effective choice for single-consumer configurations.

4.3.2. Results of Evaluation of Number of Consumers: 5

1. *Throughput*

The results showed that the use of a number of 5 consumers showed that RabbitMQ outperformed Kafka in throughput in a five-consumer configuration. RabbitMQ peaked at up to 129 msgs/sec (test 7), while Kafka only reached 22 msgs/sec (test 4), suggesting RabbitMQ processed about 5 to 8 times more messages per second, as shown in Fig. 30.

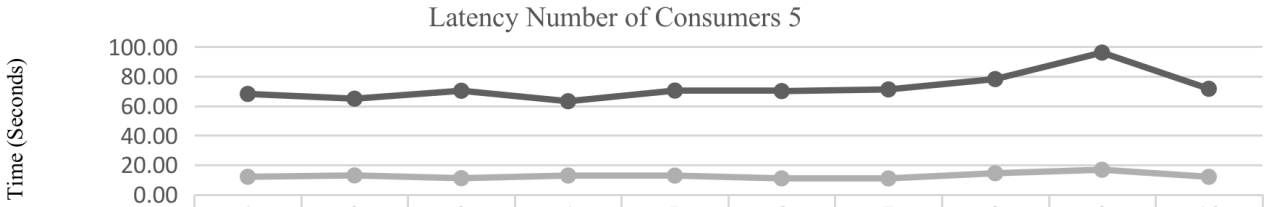


	1	2	3	4	5	6	7	8	9	10
Kafka	20.55	21.92	20.95	22.44	19.90	20.78	20.55	18.90	15.61	20.41
RabbitMQ	116.29	101.13	121.89	104.77	103.85	120.41	129.31	94.36	73.66	116.23

Fig. 30: Throughput experiment results (msg/s) number of consumers 5.

2. *Latency*

Based on the results of the experiment, RabbitMQ delivered messages with much less delay and stability than Kafka when tested with five consumers. RabbitMQ generally keeps latency below 17 seconds, moving from 10.83 seconds to 16.68 seconds. In contrast, Kafka faced a delay of 5 to 8 times slower, even shooting up to 96.04 seconds on the 9th test. This performance underscores the advantages of RabbitMQ for applications that are highly sensitive to response speed on multi-consumer architectures, the results of which are shown in Fig. 31.

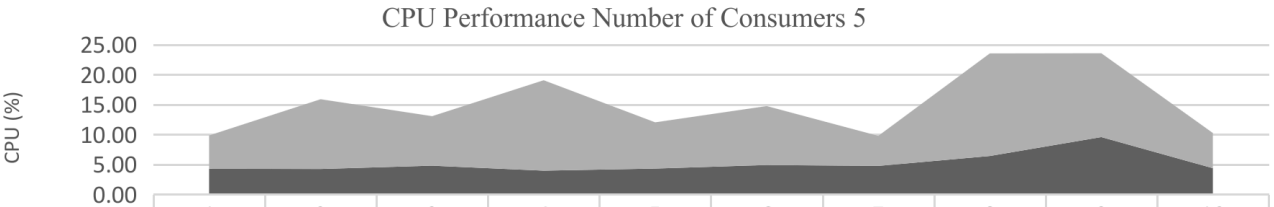


	1	2	3	4	5	6	7	8	9	10
Kafka	68.05	64.82	70.20	63.10	70.31	69.97	71.09	78.11	96.04	71.53
RabbitMQ	11.92	12.84	10.97	12.76	12.71	10.84	10.83	14.33	16.68	11.90

Fig. 31: Results of the latency experiment(s) number of consumers 5.

3. *CPU*

CPU usage patterns between Kafka and RabbitMQ at multi-consumer loads. RabbitMQ, although it showed some significant spikes (e.g., 17.13% in test 8 and 15.10% in test 4), tends to have a higher CPU footprint overall than Kafka, which maintains more stable and low usage in the range of 3.93% to 9.54%. This contrast suggests that with five consumers, the Kafka architecture may be more efficient at managing compute loads, while RabbitMQ shows more intense periods of CPU activity. The results are shown in Fig. 32 below.



	1	2	3	4	5	6	7	8	9	10
RabbitMQ	5.57	11.65	8.27	15.10	7.72	9.85	5.02	17.13	14.00	5.88
Kafka	4.24	4.21	4.76	3.93	4.28	4.87	4.72	6.38	9.54	4.34

Fig. 32: CPU experiment results (%) number of consumers 5.

4. RAM

Table 8: Comparison results of the number of consumers 5.

Metric	Measurement Type	Unit	Kafka	RabbitMQ
Throughput	Average	msg/s	20.20	108.19
	Standard Deviation	msg/s	1.89	16.25
Latency	Average	ms	72.32	12.58
	Standard Deviation	ms	9.26	1.81
CPU	Average	%	5.12	10.02
	Standard Deviation	%	1.69	4.28
RAM	Average	%	86.67	87.02
	Standard Deviation	%	2.05	5.13

The RAM usage on both message brokers, RabbitMQ consistently shows a greater need. RabbitMQ numbers fluctuate between 81% 96%, while Kafka tends to be in the range of 82% to 90%. This difference, although only ranging from 3% to 10% in each test, indicates a larger memory footprint on RabbitMQ when scaling consumers. The results can be seen in Fig. 33.

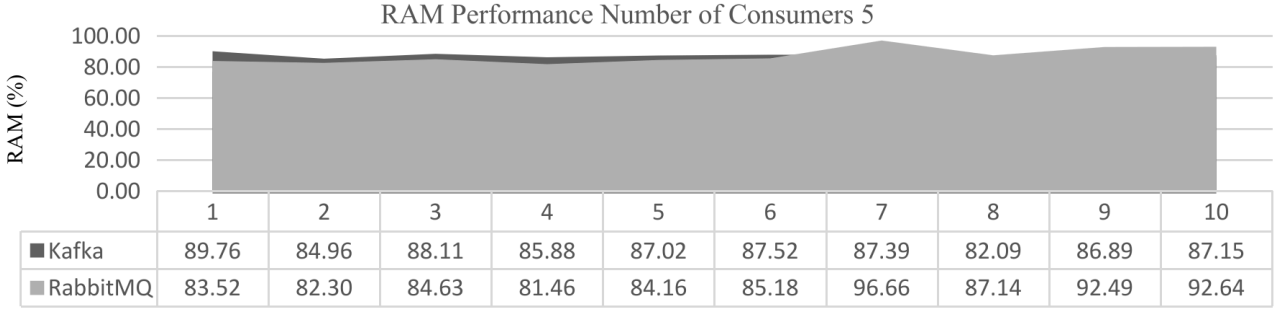


Fig. 33: RAM experiment results (%) number of consumers 5.

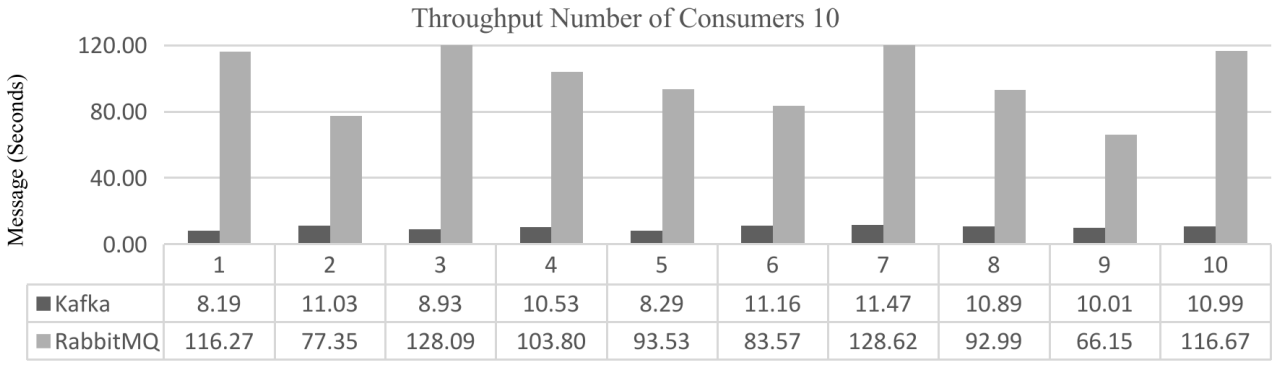
The results of comparing the performance of Kafka and RabbitMQ in a scenario with five simultaneous consumers show that RabbitMQ maintains significantly superior performance in terms of processing speed and latency compared to Kafka, despite increased CPU consumption, The comparison of performance between Kafka and RabbitMQ for sending 1 KB messages shows significant differences in most metrics whose values are shown in Table 8 below. In terms of performance metrics, RabbitMQ recorded a vastly superior average throughput (108.19 msg/s), more than five times Kafka's speed (20.20 msg/s). RabbitMQ's advantage is particularly striking in terms of average latency, resulting in a much lower delay time (12.58 ms) compared to Kafka's very high latency (72.32 ms).

Analyzing stability through Standard Deviation (SD), RabbitMQ latency is very stable (SD 1.81 ms) compared to Kafka (SD 9.26 ms), indicating that latency on RabbitMQ is much more consistent, which is very important in real-time systems. The significant difference in average Latency and Throughput suggests that RabbitMQ's performance advantage is likely statistically significant. In terms of resource efficiency, there is a reversal in the CPU trend: Kafka outperforms RabbitMQ in average CPU usage (5.12% vs. 10.02%). The increase in RabbitMQ's CPU consumption (10.02%) is likely due to the higher workload of distributing messages to five consumer queues. However, Kafka achieves this CPU efficiency at the expense of significantly lower throughput. RAM usage is relatively similar for both (Kafka: 86.67% vs. RabbitMQ: 87.02%).

4.3.3. Evaluation Results Number of Consumers: 10

1. Throughput

The results of the study showed throughput with a Number of Consumers of 10. RabbitMQ significantly outperforms Kafka in throughput on a single consumer configuration. RabbitMQ achieves values of up to 128 msgs/sec (test 7), while Kafka only 11 msgs/sec (test 7), which means RabbitMQ processes 8 to 14 times more messages per second. The results are shown in Fig. 34.

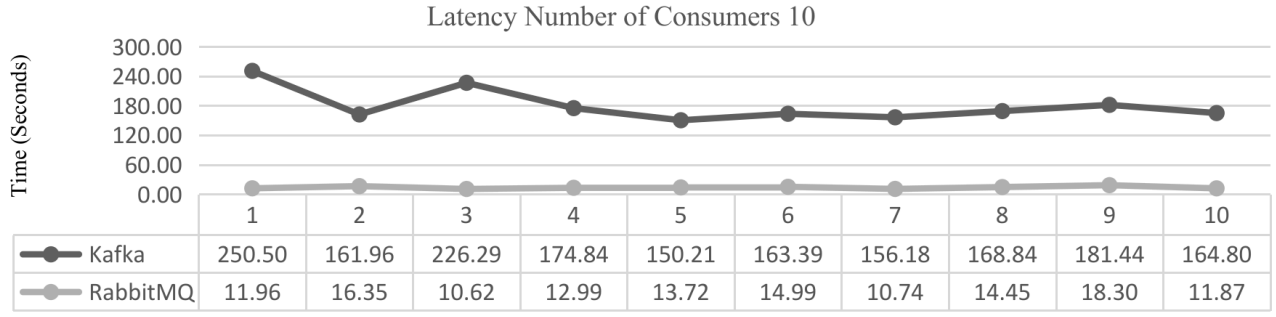


	1	2	3	4	5	6	7	8	9	10
Kafka	8.19	11.03	8.93	10.53	8.29	11.16	11.47	10.89	10.01	10.99
RabbitMQ	116.27	77.35	128.09	103.80	93.53	83.57	128.62	92.99	66.15	116.67

Fig. 34: Throughput experiment results (msg/s) number of consumers 10.

2. *Latency*

The results of the experiment showed that RabbitMQ outperformed Kafka in latency at 10 consumer configurations. RabbitMQ maintains a very low delay of between 10.62 and 18.30 seconds, while Kafka jumps from 150 to 250 seconds, as shown in Fig. 35.

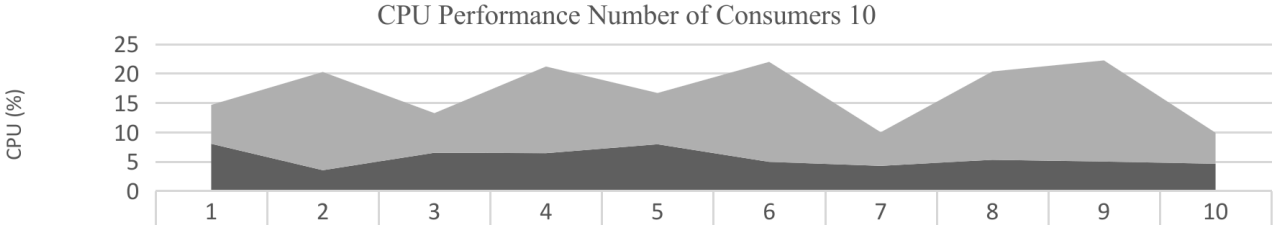


	1	2	3	4	5	6	7	8	9	10
Kafka	250.50	161.96	226.29	174.84	150.21	163.39	156.18	168.84	181.44	164.80
RabbitMQ	11.96	16.35	10.62	12.99	13.72	14.99	10.74	14.45	18.30	11.87

Fig. 35: Results of the latency experiment(s) number of consumers 10.

3. *CPU*

The CPU performance at the number of 10 consumers shown in Fig. 38, shows that RabbitMQ uses significantly higher CPUs than Kafka in the 10 consumer configuration. RabbitMQ achieved spikes of up to 17.23% (test 9) and 17.03% (test 6), while Kafka maintained a much more stable CPU usage and was low below 9%. This significant difference, where RabbitMQ often uses 3 to 5 times more CPUs, indicates CPU efficiency that is far superior to Kafka's in handling very high multi-consumer loads, as illustrated in Fig. 36.



	1	2	3	4	5	6	7	8	9	10
RabbitMQ	6.63	16.73	6.75	14.76	8.73	17.03	5.73	15.05	17.23	5.28
Kafka	8.01	3.51	6.48	6.41	7.93	4.93	4.25	5.28	4.98	4.61

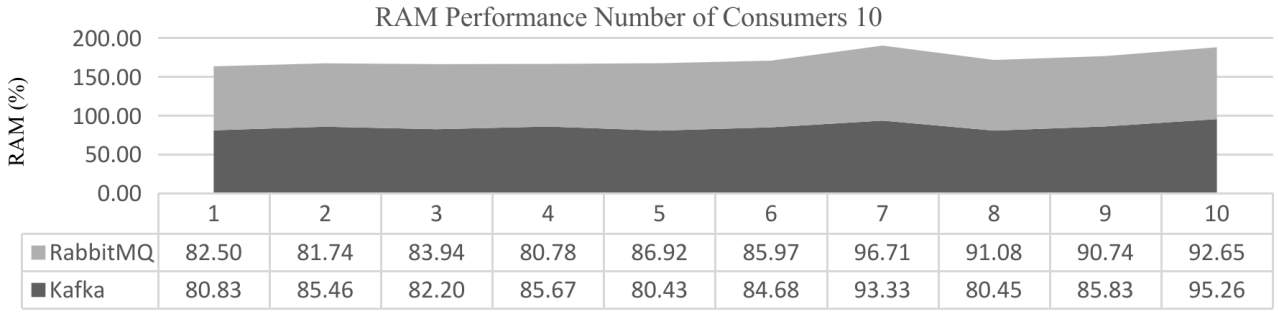
Fig. 36: CPU experiment results (%) number of consumers 10.

4. RAM

The RAM usage for consumer 10 is shown in Fig. 37. RabbitMQ has slightly higher RAM usage than Kafka. RabbitMQ fluctuates between 81%-96%, while Kafka (ignoring anomalies) is in the range of 80% to 95%. While both require substantial memory allocation, RabbitMQ tends to be a bit more RAM-hungry at these high multi-consumer loads, so careful memory capacity planning is essential for both brokers.

Table 9: Results of comparison of number of consumers 10.

Metric	Measurement Type	Unit	Kafka	RabbitMQ
Throughput	Average	msg/s	10.15	100.70
	Standard Deviation	msg/s	1.24	21.55
Latency	Average	ms	19.29	13.60
	Standard Deviation	ms	38.27	2.48
CPU	Average	%	5.64	11.39
	Standard Deviation	%	1.52	5.16
RAM	Average	%	85.95	87.30
	Standard Deviation	%	4.87	5.30



	1	2	3	4	5	6	7	8	9	10
RabbitMQ	82.50	81.74	83.94	80.78	86.92	85.97	96.71	91.08	90.74	92.65
Kafka	80.83	85.46	82.20	85.67	80.43	84.68	93.33	80.45	85.83	95.26

Fig. 37: Results of RAM (MB) experiment number of consumers 10.

The results of comparing, whose values are shown in Table 9 below. The performance of Kafka and RabbitMQ in a scenario with ten simultaneous consumers show that RabbitMQ maintains superior performance in processing speed, despite higher CPU load, while Kafka shows difficulty in maintaining throughput as the number of consumers increases in Table 9. In terms of performance metrics, RabbitMQ recorded a significantly superior average throughput (100.70 msg/s), nearly ten times the speed of Kafka (10.15 msg/s). This advantage continues in average latency, where RabbitMQ produces a lower delay time (13.60 ms) compared to Kafka's latency (19.29 ms).

Analyzing stability through Standard Deviation (SD), RabbitMQ latency is very stable (SD 2.48 ms) compared to Kafka's extremely high volatility (SD 38.27 ms). Kafka's extremely high SD latency value indicates that latency becomes highly unpredictable and unreliable in this multi-consumer configuration. The significant differences in average Throughput and Latency, along with Kafka's instability, suggest that RabbitMQ's performance advantage is likely statistically significant. In terms of resource efficiency, Kafka excels in average CPU usage (5.64% vs. 11.39% on RabbitMQ), as well as having better CPU stability (SD 1.52% vs. 5.16% on RabbitMQ). Kafka's high CPU efficiency is likely due to its very low throughput. RAM usage is relatively similar for both, although Kafka is slightly more efficient (85.95% vs. 87.30% for RabbitMQ).

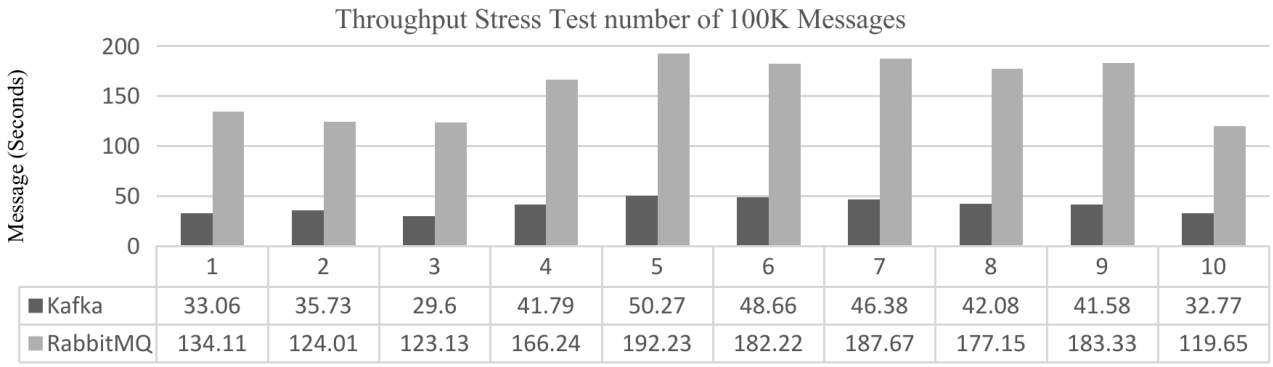
4.4. Scenario D

The results of the evaluation of scenario D include a Stress Test measurement experiment with the aim of evaluating stability and latency during overload, where the variable used is a time of 10s to send a message with a number of 100K. The results of the evaluation of scenario D include a Stress Test measurement experiment with the aim of evaluating stability and latency during overload, where the variable used is a time of 10s to send a message with a number of 100K.

4.4.1. Stress Test Evaluation Result: 100K

1. Throughput

   The results showed that RabbitMQ consistently outperformed Kafka in throughput. RabbitMQ peaked at up to 192 msgs/sec (test 5), while Kafka only reached 50 msgs/sec (test 5), indicating that RabbitMQ processed about 3 to 6 times more messages per second under pressure, as shown in Fig. 38.

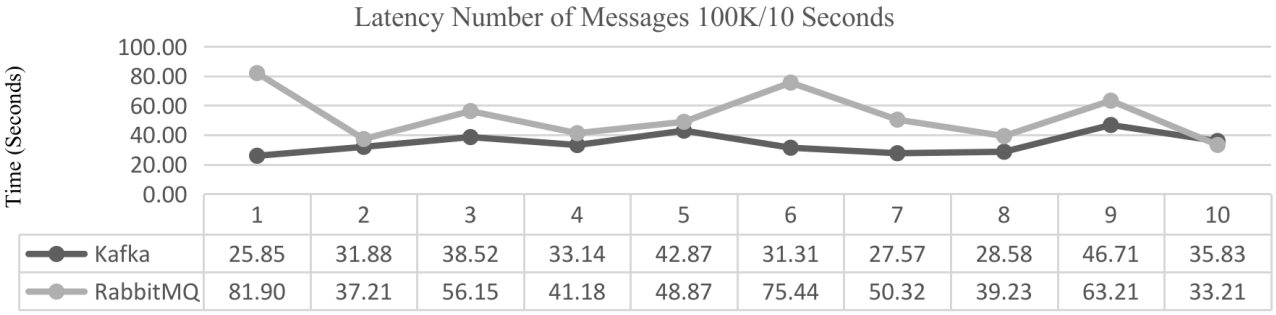


	1	2	3	4	5	6	7	8	9	10
Kafka	33.06	35.73	29.6	41.79	50.27	48.66	46.38	42.08	41.58	32.77
RabbitMQ	134.11	124.01	123.13	166.24	192.23	182.22	187.67	177.15	183.33	119.65

Fig. 38: Throughput experiment results (msg/s) stress test number of 100K messages.

2. *Latency*

Based on the experiment, Kafka showed a lower and more stable value than RabbitMQ during a stress test of 100K messages in 10 seconds. Kafka fluctuated between 25 and 47 seconds, while RabbitMQ showed greater variation and extreme spikes of up to 82 seconds on the first test. Kafka's latency advantage (10-50% lower) under these high loads implies its efficiency in managing the flow of fast messages, in contrast to the more overloaded RabbitMQ in burst conditions, as shown in Fig. 39.

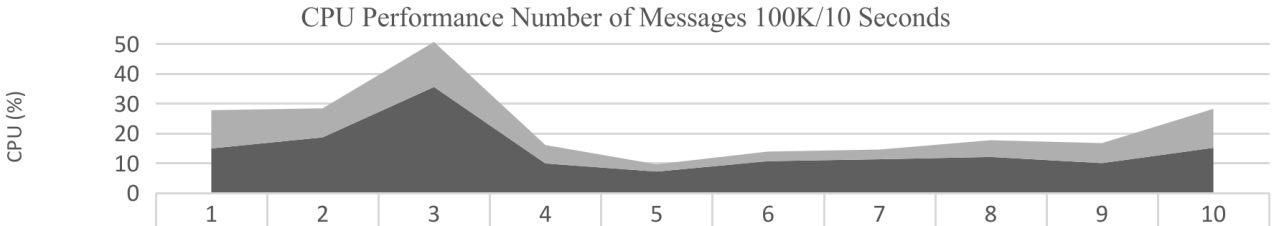


	1	2	3	4	5	6	7	8	9	10
Kafka	25.85	31.88	38.52	33.14	42.87	31.31	27.57	28.58	46.71	35.83
RabbitMQ	81.90	37.21	56.15	41.18	48.87	75.44	50.32	39.23	63.21	33.21

Fig. 39: Results of latency experiment(s) stress test number of messages 100K/10 seconds.

3. *CPU*

CPU usage with the 100K/10 seconds message count referred to in Fig. 40, shows that Kafka consistently uses CPU higher than RabbitMQ during a 100K message stress test in 10 seconds, especially at peak loads. Kafka peaked CPU usage of up to 35.51% on the 3rd test, while RabbitMQ was only 15.11% on the same test. This extreme surge underscores that 100K messages substantially increase the computational load on both brokers.



	1	2	3	4	5	6	7	8	9	10
RabbitMQ	12.82	9.73	15.11	6.18	2.43	3.22	3.25	5.61	6.71	13.06
Kafka	14.92	18.63	35.51	9.88	7.16	10.65	11.29	12.06	9.99	15.13

Fig. 40: CPU experiment results (%) stress test number of messages 100K.

4. RAM

The results of the experiment, shown in Fig. 41, show that Kafka and RabbitMQ had very similar and stable RAM usage during a stress test of 100K messages in 10 seconds. Kafka fluctuates between 79.85 MB and 86.88 MB, while RabbitMQ ranges from 79.73 MB to 88.44 MB, showing a marginal difference of less

Table 10: Results of stress test comparison of 100K/10 seconds message.

Metric	Measurement Type	Unit	Kafka	RabbitMQ
Throughput	Average	msg/s	40.19	158.97
	Standard Deviation	msg/s	7.13	30.03
Latency	Average	ms	34.23	52.67
	Standard Deviation	ms	6.77	16.49
CPU	Average	%	14.52	7.81
	Standard Deviation	%	8.07	4.58
RAM	Average	%	83.63	83.94
	Standard Deviation	%	1.87	2.45

than 1%. This indicates that both brokers have similar memory footprints under high loads, making RAM efficiency not a major differentiating factor for these use cases, but both require substantial memory allocation.

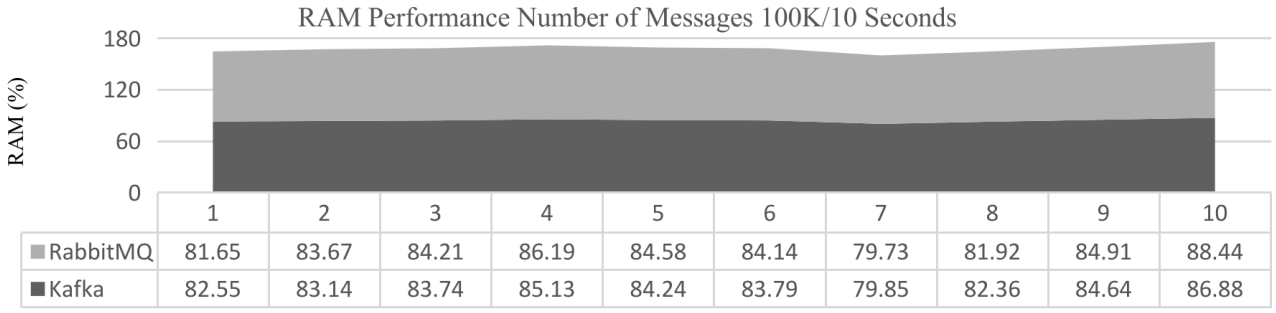


Fig. 41: RAM experiment results (%) stress test number of messages 100K.

Stress test results comparing Kafka and RabbitMQ in a scenario of sending 100,000 messages over 10 seconds show that RabbitMQ delivers significantly superior performance in terms of processing capabilities and core resource efficiency. In terms of performance metrics, RabbitMQ recorded a much higher average throughput (158.97 msg/s), nearly four times the speed of Kafka (40.19 msg/s). Although RabbitMQ's average latency (52.67 ms) is higher than Kafka's (34.23 ms), RabbitMQ's significant advantage in throughput indicates that it is capable of handling a much denser message load.

Analyzing stability through Standard Deviation (SD), RabbitMQ shows higher volatility in both Throughput (SD 30.03 msg/s vs. 7.13 msg/s on Kafka) and Latency (SD 16.49 ms vs. 6.77 ms on Kafka). However, the large difference in average Throughput, along with better resource efficiency, indicates that RabbitMQ's overall performance advantage is statistically significant in its ability to handle high load pressure. In terms of resource efficiency, RabbitMQ showed a much lower average CPU usage (7.81%) compared to Kafka (14.52%), and was also more stable (SD 4.58% vs. 8.07% on Kafka). RAM usage was relatively similar for both, with only a slight difference. RabbitMQ's high CPU efficiency, despite its higher latency, indicates that it uses computational resources more optimally to process very large message volumes, whose values are shown in Table 10 below.

5. Conclusion

The results of the experiment reveal different performance characteristics between the two brokers. RabbitMQ demonstrated superior throughput and lower latency for individual message processing in most scenarios, handling high volumes efficiently. Specifically, for 100 KB messages, RabbitMQ achieved 117.67 msgs/s compared to Kafka's 12.08 msgs/s, with a latency of 4.38 ms versus 83.27 ms. However, Kafka showed stronger performance with 10 KB messages (105.24 msgs/s vs. 48.37 msgs/s), indicating its efficiency in batch data processing. In stress tests, RabbitMQ significantly outperformed Kafka with a throughput of 158.97 msgs/s compared to 40.19 msgs/s. Although RabbitMQ showed better CPU efficiency, both brokers experienced high RAM consumption, with RabbitMQ being slightly more memory intensive. These findings suggest that RabbitMQ is more suitable for

applications that require high-volume individual message delivery with low latency, while Kafka excels at handling large-scale data streams with high durability requirements.

This study has several limitations. First, the experiment was conducted in a controlled environment with specific hardware and network configurations that may not represent all production scenarios. Second, the variations in message size, volume, and number of consumers tested are still limited and do not cover scenarios with more complex network topologies. Third, aspects of security, operational costs, and long-term system maintenance have not been explored. Fourth, testing was conducted over a relatively short period, and long-term performance with dynamic traffic patterns has not been evaluated. Further research needs to address these limitations in several ways: (1) testing with various hardware configurations, including cloud-native and multi-region architectures; (2) exploring hybrid deployment scenarios that combine both brokers to leverage their respective strengths; (3) comprehensive security evaluation, including authentication mechanisms, data encryption, and regulatory compliance; (4) testing with realistic traffic patterns that reflect actual production workloads, including daily, weekly, and seasonal variations; (5) analysis of total cost of ownership (TCO) and return on investment (ROI) at various deployment scales; and (6) industry-specific case studies such as fintech, e-commerce, and IoT to provide practical guidance on selecting a message broker based on application requirements.

CRediT Authorship Contribution Statement

M. R. Ramadan: Conceptualization, Methodology, Writing – Original Draft. **A. M. Umam:** Writing – Review & Editing, Formal analysis. **A. Asmani:** Writing – Review & Editing. **R. M. Ijtihadie:** Writing – Review & Editing, Supervision.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data Availability

The data used to support the findings of this study are available from the corresponding author upon request.

Declaration of Generative AI and AI-assisted Technologies in The Writing Process

The authors used generative AI to improve the writing clarity of this paper. They reviewed and edited the AI-assisted content and take full responsibility for the final publication.

References

[1] M. Steen and A. S. Tanenbaum, “A brief introduction to distributed systems,” *Computing*, vol. 98, no. 10, pp. 967–1009, Oct. 2016, doi: 10.1007/s00607-016-0508-7.

[2] R. Laigner, Y. Zhou, M. A. V. Salles, Y. Liu, and M. Kalinowski, “Data Management in Microservices: State of the Practice, Challenges, and Research Directions,” *CoRR*, 2021, [Online]. Available: https://arxiv.org/abs/2103.00170

[3] M. Albano, L. L. Ferreira, L. M. Pinho, and A. R. Alkhawaja, “Message-oriented middleware for smart grids,” *Computer Standards & Interfaces*, vol. 38, pp. 133–143, 2015, doi: https://doi.org/10.1016/j.csi.2014.08.002.

[4] T. Rausch, S. Dustdar, and R. Ranjan, “Osmotic Message-Oriented Middleware for the Internet of Things,” *IEEE Cloud Computing*, vol. 5, no. 2, pp. 17–25, 2018, doi: 10.1109/MCC.2018.022171663.

[5] S. T and S. N. K, “A study on Modern Messaging Systems- Kafka, RabbitMQ and NATS Streaming,” *CoRR*, 2019, [Online]. Available: http://arxiv.org/abs/1912.03715

[6] T. P. Raptis and A. Passarella, “A Survey on Networked Data Streaming With Apache Kafka,” *IEEE Access*, vol. 11, no. , pp. 85333–85350, 2023, doi: 10.1109/ACCESS.2023.3303810.

[7] M. Rostanski, K. Grochla, and A. Seman, “Evaluation of highly available and fault-tolerant middleware clustered architectures using RabbitMQ,” in *2014 Federated Conference on Computer Science and Information Systems*, 2014, pp. 879–884. doi: 10.15439/2014F48.

[8] A. E. Bagaskara, S. Setyorini, and A. A. Wardana, “Performance Analysis of Message Broker for Communication in Fog Computing,” in *2020 12th International Conference on Information Technology and Electrical Engineering (ICITEE)*, 2020, pp. 98–103. doi: 10.1109/ICITEE49829.2020.9271733.

[9] R. Goel, “Evaluating Message Brokers: Performance, Scalability, and Suitability for Distributed Applications,” 2024, pp. 62–65. doi: 10.5923/j.ajca.20241105.02.

[10] P. Dobbelaere and K. S. Esmaili, “Kafka versus RabbitMQ: A comparative study of two industry reference publish/subscribe implementations: Industry Paper,” in *Proceedings of the 11th ACM International Conference on Distributed and Event-Based Systems*, in DEBS '17. Barcelona, Spain: Association for Computing Machinery, 2017, pp. 227–238. doi: 10.1145/3093742.3093908.

[11] G. Hesse, C. Matthies, and M. Uflacker, "How Fast Can We Insert? An Empirical Performance Evaluation of Apache Kafka," in *2020 IEEE 26th International Conference on Parallel and Distributed Systems (ICPADS)*, 2020, pp. 641–648. doi: 10.1109/ICPADS51040.2020.00089.

[12] S. Henning and W. Hasselbring, "Benchmarking scalability of stream processing frameworks deployed as microservices in the cloud," *Journal of Systems and Software*, vol. 208, p. 111879, 2024, doi: https://doi.org/10.1016/j.jss.2023.111879.

[13] M. S. H. Chy, M. A. R. Arju, S. M. Tella, and T. Cerny, "Comparative Evaluation of Java Virtual Machine-Based Message Queue Services: A Study on Kafka, Artemis, Pulsar, and RocketMQ," *Electronics*, vol. 12, no. 23, 2023, doi: 10.3390/electronics12234792.

[14] T. P. Raptis, C. Cicconetti, and A. Passarella, "Efficient topic partitioning of Apache Kafka for high-reliability real-time data streaming applications," *Future Generation Computer Systems*, vol. 154, pp. 173–188, 2024, doi: https://doi.org/10.1016/j.future.2023.12.028.

[15] A. Ćatović, N. Buzađija, and S. Lemes, "Microservice development using RabbitMQ message broker," *Sci. Eng. Technol.*, vol. 2, no. 1, pp. 30–37, Apr. 2022.

[16] T. Kaciuczyk, T. Korga, and J. Smołka, "Functional and performance analysis of selected message brokers in a distributed application," *jcsi*, vol. 14, pp. 19–25, Mar. 2020.

[17] S. Dyjach and M. Plechawska-Wójcik, "Efficiency comparison of message brokers," *Journal of Computer Sciences Institute*, vol. 31, pp. 116–123, 2024, doi: 10.35784/jcsi.6084.

[18] M. A. Spohn, "On MQTT scalability in the Internet of Things: Issues, solutions, and future directions," *J. Electron. Electric. Eng.*, p. 4, Oct. 2022.

[19] A. Detti, L. Funari, and N. Blefari-Melazzi, "Sub-Linear Scalability of MQTT Clusters in Topic-Based Publish-Subscribe Applications," *IEEE Transactions on Network and Service Management*, vol. 17, no. 3, pp. 1954–1968, 2020, doi: 10.1109/TNSM.2020.3003535.

[20] I. Linkov *et al.*, "Resilience stress testing for critical infrastructure," *International Journal of Disaster Risk Reduction*, vol. 82, p. 103323, 2022, doi: https://doi.org/10.1016/j.ijdrr.2022.103323.

[21] Y. Guo *et al.*, "Advances in Full-Link Stress Testing Technology for Cloud-Native Information System," in *2024 IEEE 4th International Conference on Information Technology, Big Data and Artificial Intelligence (ICIBA)*, 2024, pp. 1661–1665. doi: 10.1109/ICIBA62489.2024.10868076.

[22] G. Fu, Y. Zhang, and G. Yu, "A Fair Comparison of Message Queuing Systems," *IEEE Access*, vol. 9, no. , pp. 421–432, 2021, doi: 10.1109/ACCESS.2020.3046503.

[23] G. Ortiz, A. Bazan-Muñoz, W. Lamersdorf, and A. Garcia-de-Prado, "Evaluating the integration of Esper complex event processing engine and message brokers," *PeerJ Comput. Sci.*, vol. 9, p. e1437, July 2023.