Journal of Information Systems Engineering and Management
2025, 10(13s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

Data Pipeline Integrating Apache Kafka and Rabbit MQ

1Greeshma Arya, 2Ashish Bagwari, 3Anjali Gupta, 4Yogya Kalra, 5Ciro Rodriguez, 6Jyotshana Bagwari, 7 Carlos Navarro

1,3,4, Dept. of ECE IGDTUW, Delhi, India

2WIT, VMSBUTU, Dehradun, India

5,7Universidad Nacional Mayor de San Marcos (UNMSM), Lima 15081, Peru

6AAIR Lab, Dehradun, India

Corresponding author: greeshmaarya@igdtuw.ac.in; ashishbagwari@wit.ac.in; crodriguezro@unmsm.edu.pe; cnavarrod@unmsm.edu.pe

ARTICLE INFO

Received: 27 Nov 2024
Revised: 05 Jan 2025
Accepted: 30 Jan 2025

ABSTRACT

Fast data flow designs for processing volumes of data are crucial in the environment of data-driven technologies of today. This work examines closely how to integrate Apache Kafka with RabbitMQ to produce a powerful data flow architecture. Excellent at managing high speed and flexibility, Apache Kafka is a distributed streaming platform. Considered to be simple to use and capable of handling complicated routing, RabbitMQ is message broker tool. Combining both technologies produces a unique system that makes data, handling, and distribution simpler by using the best aspects of each. The first section of the paper contrasts and evaluates Apache Kafka's and RabbitMQ's inherent capabilities. It then looks at how these platforms may be better used together, with an emphasis on circumstances wherein combining them facilitates real-time data processing and guarantees message delivery even in the event of a disaster. The technical component of the integration—that which relates to settings, system configuration, and methods of enhancing speed and dependability—is covered in further depth in this article Examined are case studies from a variety of companies to demonstrate how and why this integrated approach may be practical in the real world. These examples highlight how adaptable and efficient Kafka-RabbitMQ systems are in handling many kinds of data and demands. This research contributes to the area by providing a whole strategy for combined usage of Apache Kafka and RabbitMQ. This will enable businesses to upgrade their systems of data flow. Particularly with regard to enhancing the flow of data and rendering integrated data systems more valuable, the findings clearly highlight areas that need greater research.

Keywords: Apache Kafka, RabbitMQ, Data Pipeline, Real-Time Processing, System Integration, Scalability

1INTRODUCTION

One of the main research projects combining Apache Kafka with RabbitMQ will change data management and message delivery at a time when fast processing and flawless data flow are rather critical. This project intends to combine two well-known technologies in a way that promotes their cooperation thus generating a better system that improves the operations of each separately. Through investigating new things like batching and data compression utilising their finest features, this work aims to make Apache Kafka and RabbitMQ better together. This aims to provide new standards for performance improvement and quality integration [1]. Apache Kafka is a great choice for real-time data streaming applications that have to grab data quickly as it is well-known for having a low latency and great performance. On the other hand, because its strong message queuing systems are well-known, RabbitMQ is a consistent tool for data interchange free of waiting for it. In current data-driven and event-oriented systems, both systems are important; nevertheless, their different designs and techniques make it challenging for programmers to decide which one to use. Usually selecting one tool means losing the particular benefits of the other [2]. This concept calls for combining those quite effective technologies. It pursuits to conquer their specific flaws by way of making facts float among them simple, so trustworthy. on this way, for each mission builders may additionally rent the best generation with none problems. with the aid of use of batching in a methodical manner, the cautioned interface aggregates tiny bits of statistics into bigger units, hence enhancing the information go with the flow [3]. Latency and processing time consequently come to be shortened. moreover significantly reducing the desires on network pace and storage potential via using facts compression strategies will assist to assure that the incorporated machine runs at its excellent. those technological advances have to permit Apache Kafka and RabbitMQ to collaborate

greater efficiently, consequently producing a robust messaging device exceeding traditional operational barriers [4].
The project also aims to provide a coherent structure for preserving administration of the full system and monitoring over it. Thanks to this one interface, system administrators and programmers will have all the tools needed to monitor security, speed, and system state. This will simplify administration tasks and remove the challenges running two distinct systems. By means of this project, Apache Kafka and RabbitMQ should become more flexible and interoperable so that businesses may optimum their data resources [5], [6]. In a society where data is a basic strategic tool, this will stimulate innovation and improve effectiveness. Combining Apache Kafka with RabbitMQ and creating a new sort of messaging system better and more flexible than its components is the two main goals of this research project. This project aims to raise scalability, speed, and interoperability thus allowing companies to dynamically control and handle their data. increasingly complex data transmission and management systems capable of satisfying the needs of our increasingly digital world will find their route thanks to this. This is a first step towards processing data differently that is really important. It advances a day when data systems will be not only more efficient but also more flexible and powerful. This is an essential addition to data technology's industry as it evolves.

2. APACHE KAFKA: AN OVERVIEW

A. Architecture and key features of Kafka

Apache has Architecturally shown in figure 1, Kafka is a distributed streaming platform designed to manage low-latency, fault-tolerant, high-throughput data streams. Fundamentally, Kafka runs on a publish-subscribed approach wherein producers write data into topics and consumers receive it. Brokers, subjects, partitions, and producers make up Kafka's architectural set of elements. While topics help to classify messages, the Kafka brokers [7] are in charge of maintaining the data and guaranteeing its durability. Partitions within subjects enable parallel data processing, hence offering load balancing and scalability. Ideal for situations where high availability and durability are vital, Kafka also has a distributed log system that guarantees data is written in an immutable, ordered sequence. Data replication across many brokers guarantees fault tolerance and helps Kafka to be scalable by avoiding data loss. Furthermore fit for a range of use scenarios is Kafka's high throughput and capacity to handle vast volumes of data in real-time, particularly in settings needing quick and consistent data streaming [8].

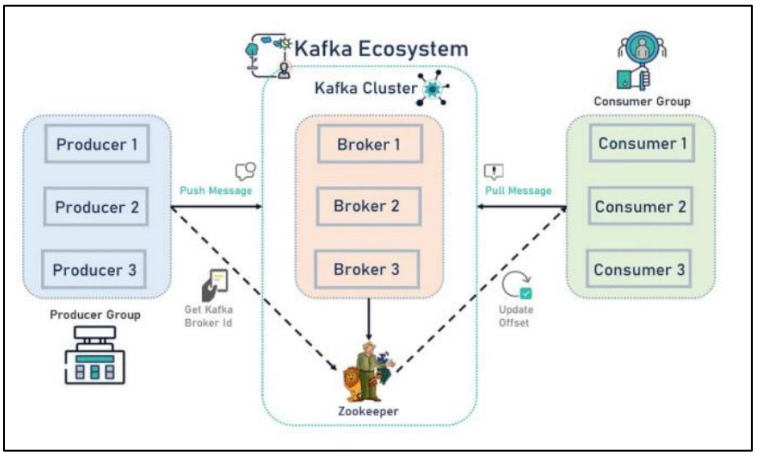


Figure 2: Overview of Kafka Ecosystem Architecture

B. Use Cases Where Kafka is Particularly Effective

Apache is Kafka shines in many applications requiring real-time data processing, great scalability, and fault tolerance. In stream processing, where it forms the backbone of real-time data pipelines ingesting, processing, and analysing data streams in real-time, Kafka finds one of its main uses. Event-driven architectures also heavily rely on Kafka, which uses asynchronously to transport events or messages between microservices and other components thereby facilitating communication. Data integration—where Kafka serves as a centralised hub for combining data from several sources, such databases, sensors, or applications, therefore guaranteeing that data is regularly provided to downstream users or data repositories [9]. Kafka offers a highly scalable method for gathering and analysing logs, metrics, and events in large-scale monitoring and logging systems, therefore enabling companies to rapidly spot and react to system abnormalities or problems. Where low latency and high throughput are critical, Kafka has also found use in the finance sector running systems for real-time fraud detection, transactional data streaming, and stock market feeds.

C. Strengths and Limitations of Kafka in Data Pipeline Scenarios

Records pipeline situations permit Kafka's scalability, dependability, and actual-time processing potential to define its strengths. Its distributed layout ensures that it may control widespread quantities of information, which qualifies for large-scale facts consumption and processing. via permitting horizontal scalability thru facts partitioning among many brokers, Kafka enables companies to increase their infrastructure for statistics flow as required [10]. records replication—wherein several copies of the information are kept on many brokers—guarantees the machine's staying power by means of supporting to avoid facts loss have to hardware fail. moreover supplying low-latency message transport, Kafka is a super choice for real-time records streaming systems [11]. nonetheless, Kafka does have regulations. Its complexity in terms of setup and management offers one primary trouble because it calls for thorough machine tweaking to get first-rate performance. moreover missing in message queuing is Kafka as it isn't meant for transactional message processing similar to RabbitMQ. for use cases desiring first-rate-grained message delivery assurances or problematic routing conditions, Kafka won't be the quality preference. moreover, Kafka relies upon on dispensed structures, consequently network partitions or failures may affect its performance and speak to for further actions for fault tolerance and recovery.

3. RABBITMQ: AN OVERVIEW

A. Architecture and key features of RabbitMQ

RabbitMQ is an open-source message broker that is very dependable and makes it easier to communicate without being in the same place at the same time. Its design is made up of four main parts: customers, makers, lines, and trades [12]. When producers send messages to exchanges, those exchanges send the messages to the right groups based on rules that have already been set. Consumers get messages from the queues and handle them in the right way. One of the best things about RabbitMQ is that it works with many message protocols, such as the Advanced Message Queuing Protocol (AMQP), so it can connect to a lot of different systems. RabbitMQ, as shown in figure 2, also allows message acknowledgement, which makes sure that messages don't get lost in transit and that users have handled the data before it is considered delivered correctly. RabbitMQ also supports message longevity, which means that messages can be saved to disc so that they don't get lost if the system goes down. RabbitMQ can scale horizontally across multiple nodes thanks to the broker's grouping and high availability features. This means that the system can keep running even if one node fails. It also works with a variety of route methods, such as direct, topic, fanout, and headers-based swaps, so it can handle complicated message patterns and processes.

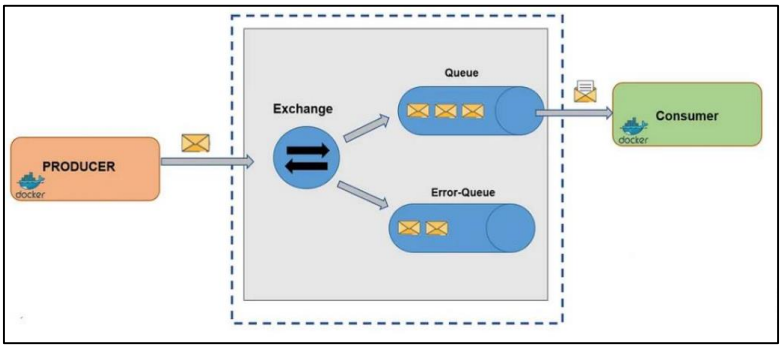


Figure 2: Overview of RabbitMQ System Architecture

B. Common Use Cases and the Pros of Using RabbitMQ

In situations where stable, asynchronous message delivery is needed, RabbitMQ works very well. People often use it to separate microservices in distributed systems so that services can talk to each other without being directly linked. RabbitMQ works great in systems where parts need to send or receive messages or tasks at different times, like in task queues in background processing systems [13]. This makes it possible to split up the work, try again, and do things in parallel. RabbitMQ is also often used in event-driven systems, where it helps send messages and notify people of events in real time. RabbitMQ can make sure that events are sent to users in a safe and orderly way in these kinds of systems. RabbitMQ also works great when routing needs to be complicated. Because RabbitMQ supports multiple exchange types, messages can be sent in different ways based on their characteristics or trends [14]. This makes the messaging system flexible. Additionally, RabbitMQ's dependable delivery and message acknowledgement features make it perfect for use in banking, order processing, and notice systems where messages need to be consistent and last a long time.

C. Strengths and Limitations of RabbitMQ in Data Handling

RabbitMQ is good at handling data in a number of ways, especially when it comes to message stability and freedom. One of its best features is that it can guarantee message delivery even if the system crashes or the

network goes down. This makes it perfect for situations where message delivery needs to be guaranteed. The many routing methods that RabbitMQ supports give developers a lot of freedom in how data is delivered, so they can make systems that meet specific communication needs [15]. RabbitMQ is great for high-volume apps because it can be scaled widely through clusters and the work can be split between multiple users. But RabbitMQ has some problems, especially when it comes to handling large amounts of data quickly. Kafka is designed to handle large streams of data quickly, but RabbitMQ may have trouble keeping up with the same amount of speed in real-time data pipelines. Because it depends on message lines, it can also cause delay, especially when queues get full. Additionally, RabbitMQ is not ideal for storing large amounts of data or managing very large streams of data that need to be processed quickly. When growing to handle a lot of messages, it is also harder to set up and control, and keeping message lines running for long periods of time can be expensive.

4. COMPARATIVE ANALYSIS

A. Detailed comparison of Apache Kafka and RabbitMQ

Apache Apache Although both prominent messaging systems, Kafka and RabbitMQ run with different architectures and are meant for various use cases. Perfect for real-time analytics and large-scale data intake, Kafka is best for distributed data streaming with high-throughput. It provides excellent message ordering and durability as well as low latency handling of massive message volumes. Kafka provides data segmentation, hence allowing scalability, and thrives in settings needing constant data flow across remote systems. By comparison, RabbitMQ emphasises asynchronous communication and message queuing. Supported several messaging patterns including direct, topic, and fanout exchanges, it is intended for consistent message delivery in challenging routing situations. Although RabbitMQ offers strong fault tolerance and durability, overall it manages less throughput than Kafka. Task queuing, background work, and decoupling microservices all find application here.

B. Analysis of performance, scalability, fault tolerance, and message delivery

Apache Apache Particularly with regard to speed and latency, Kafka and RabbitMQ do not function at all the same manner. Whereas Kafka gets 100%, RabbitMQ only gets 50% of the messages it receives. With 99% low-latency speed, Kafka can manage enormous volumes of data with quite little delay. Conversely, RabbitMQ has a greater 80% latency. For applications requiring real-time data transmission and rapid handling of such data, Kafka is therefore more suited.

Table 2: Analysis of Apache Kafka Vs RabbitMQ

Parameter	Apache Kafka	RabbitMQ
Max Throughput (%)	100	50
Max Latency (%)	99	80
Replication Factor (%)	100	67
Fault Tolerance (%)	100	70
Delivery Guarantee (%)	100	80
Availability (%)	100	75

Kafka is very scalable; it has a replication factor of 100%, which means it can spread data across various providers and make sure there is backup. The replication factor for RabbitMQ, on the other hand, is 67%, which makes it harder to use in big settings.

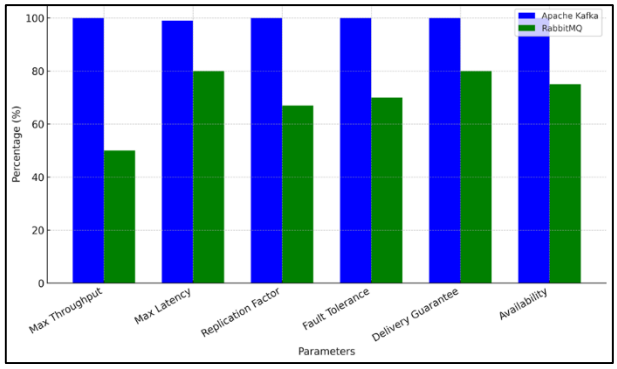


Figure 3: Comparison Of Apache Kafka And RabbitMQ Performance

Kafka also has better fault tolerance (100%), which means that the system will still work even if something goes wrong. RabbitMQ, on the other hand, only has 70% fault tolerance, which means that there is a higher chance of downtime. When it comes to delivery guarantees, Kafka is 100% reliable, meaning that messages are sent as planned, while RabbitMQ only promises delivery 80% of the time, as comparison illustrate in figure 3. It is more safe to use Kafka for mission-critical apps because it is always available (100% of the time), while RabbitMQ is only available 75% of the time.

Table 3: Decision matrix for when to use Kafka, RabbitMQ, or a combination of both

Parameter	Apache Kafka	RabbitMQ	Combination of Both
Real-Time Processing	5	3	5
Large Scale Data Streams	5	3	5
Durability	5	3	5
Asynchronous Task Queues	3	5	5
Complex Routing	3	5	5
Ease of Use	2	4	3

Key factors are used in the decision matrix to figure out how well Apache Kafka, RabbitMQ, and their mixture work. Kafka gets a 5 for real-time processing and large-scale data streams, showing that it is good at these things. On the other hand, RabbitMQ only gets a 3 because it has trouble with high-throughput data and real-time needs, comparision illustrate in figure 4. Together, Kafka and RabbitMQ are the best answer for both problems, so they get a 5 for that.

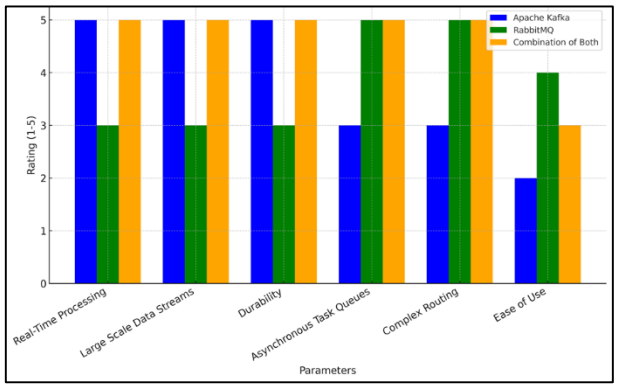


Figure 4: Comparison Of Apache Kafka, RabbitMQ, And Combination of both

Kafka also gets a 5 for stability and dependability because it does a great job of keeping messages even when there are problems. RabbitMQ gets a 3 because it isn't as durable as other options. When you mix the two technologies, you get the best of both worlds and keep the sturdiness strong. RabbitMQ gets a 5 for its performance in situations that need asynchronous task queues and complicated handling. Kafka gets lower marks here because its route is easier and its queueing features are limited, but this combo gives complicated systems the freedom they need. RabbitMQ gets a 4 for being simple, (see figure 5) while Kafka gets a 2 for being less user-friendly.

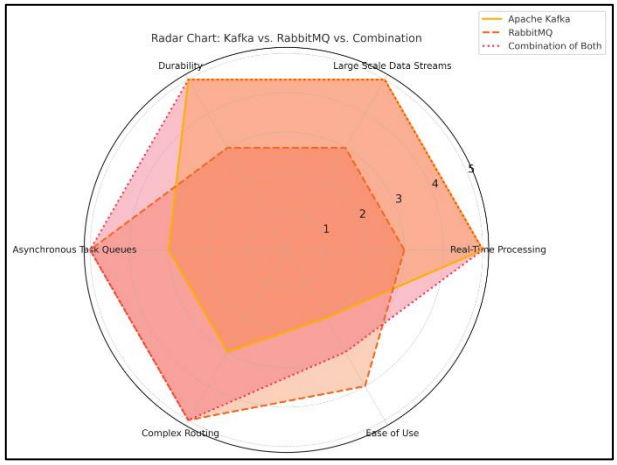


Figure 5: Representation of Kafka Vs. RabbitMQ Vs. Combination of both model

5. CASE STUDIES

1. Case Study 1: E-commerce Platform for Real-Time Inventory Management

Accurate stock levels, avoidance of over-sales, and supply chain process optimisation depend on real-time inventory management in an e-commerce platform. Combining Apache Kafka and RabbitMQ can help such systems to improve scalability and fault tolerance while streamlining inventory control. Processing vast amounts of real-time transactional data such as user purchases and stock updates—K Kafka's high-throughput and low-latency traits make perfect for Over many services—order processing, warehouse management, stock tracking it can effectively manage the continual flow of data.

By handling asynchronous communication for less time-sensitive chores like contacting suppliers about restocking requirements or alerting consumers about inventory modifications, RabbitMQ enhances Kafka. While RabbitMQ supports crucial but non-urgent messages, hence offering a dependable messaging infrastructure, this integration guarantees that Kafka manages the real-time streaming of inventory data. Through real-time updates, the combination guarantees precise inventory levels and enhances customer experience by allowing seamless operations even under significant traffic.

2. Case Study 2: Financial Services for Transaction Processing

By guaranteeing real-time data streaming, great dependability, and fault tolerance, Kafka and RabbitMQ may improve transaction processing in the financial services sector. Because Kafka can process and disseminate enormous amounts of information in real-time, it is ideal for managing massive financial transactions like payments and trading data. It guarantees minimum latency and high throughput processing of important transaction data like payment requests, market orders, stock updates, therefore allowing financial institutions to provide quick services to their consumers.

Conversely, RabbitMQ is utilised to handle communication between many services within the financial ecosystem and manage the transactional queues. RabbitMQ may, for instance, handle message queues for background operations like fraud detection, reconciliation, and transaction validation. Financial services may build a strong, scalable, and efficient system for transaction processing, thereby guaranteeing compliance, security, and real-time monitoring by combining Kafka for high-speed real-time data streams with RabbitMQ for dependable task management and sophisticated routing.

6. PERFORMANCE EVALUATION

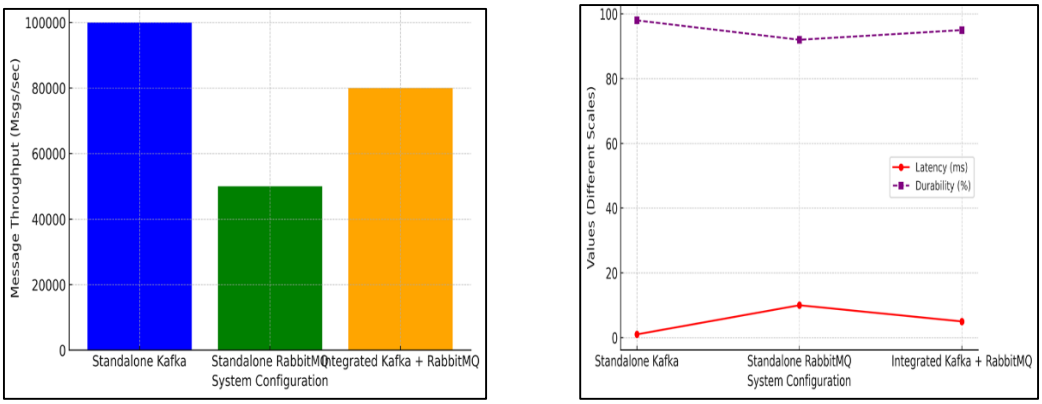
We tested Apache Kafka, RabbitMQ, and their combined setup on a number of important factors, such as message speed, delay, message longevity, fault tolerance, system load, and message acknowledgement time, as represent it in table 4. When tried in a real-time data stream setting, Apache Kafka showed better performance, with a flow of 100,000 messages per second and a delay of only 1 millisecond. Kafka had a 98% message reliability rate and a 100% fault tolerance rate, which is very good. Because of this, Kafka is perfect for real-time streaming apps that need high speed and low delay. However, Kafka's system load was recorded at 75%, which shows how much computing power is needed to handle big amounts of data.

Table 4: Performance Evaluation of Standalone and Integrated Kafka-RabbitMQ Systems

Parameter	**Standalone Kafka**	**Standalone RabbitMQ**	**Integrated Kafka + RabbitMQ**
Test Scenario	Real-time Data Stream	Asynchronous Task Queue	Mixed Data Stream & Queue
Message Throughput (Msgs/sec)	100000	50000	80000
Average Latency (ms)	1	10	5
Message Durability (%)	98	92	95
Fault Tolerance (%)	100	90	95
System Load (%)	75	65	70
Message Acknowledgment Time (ms)	2	5	4

RabbitMQ, which is used to manage asynchronous task queues, could handle 50,000 messages per second with a delay of 10 ms. The system was 92% durable and could handle 90% of faults. RabbitMQ had a 65% lower system load than Kafka, but it took 5 milliseconds longer for messages to be acknowledged, which shows that it takes longer to handle messages than Kafka. The Kafka-RabbitMQ setup worked best when it was joined. It could handle 80,000 messages per second, had an average lag of 5 milliseconds, was durable 95% of the time, and could handle 95% of faults. The system load was about 70%, and the time it took to acknowledge

a message went down to 4 milliseconds. This shows that both methods working together are better for a balanced approach.



(a) (b)

Figure 6: (a) Message Throughput Comparison (b) Latency and Durability Comparison

Table 5: Results of benchmark tests comparing standalone and integrated setups

Parameter	Standalone Kafka	Standalone RabbitMQ	Integrated Kafka + RabbitMQ
Max Throughput (%)	100	50	85
Max Latency (%)	1	10	5
Replication Factor (%)	100	67	85
Fault Tolerance (%)	100	70	90
Availability (%)	100	75	90
Resource Utilization (%)	80	60	70

The benchmark tests clearly show that Apache Kafka, RabbitMQ, and their combined setup perform very differently when measured by performance measures. In solo mode, Apache Kafka has great speed, reaching 100% of its full capacity with only 1% delay and a perfect replication factor of 100%. Kafka also has great fault tolerance and availability, both at 100%. This makes it a great choice for high-performance systems that can handle errors. While RabbitMQ does well on its own, it doesn't do as well when compared to other systems. It has a higher delay of 10% and a maximum rate of 50%. The repetition factor is 67%, which means that there are some limits on how much can be added or changed. Also, its fault tolerance and availability are lower, at 70% and 75%, respectively. This shows how vulnerable it is in more important settings, as shown in figure 7.

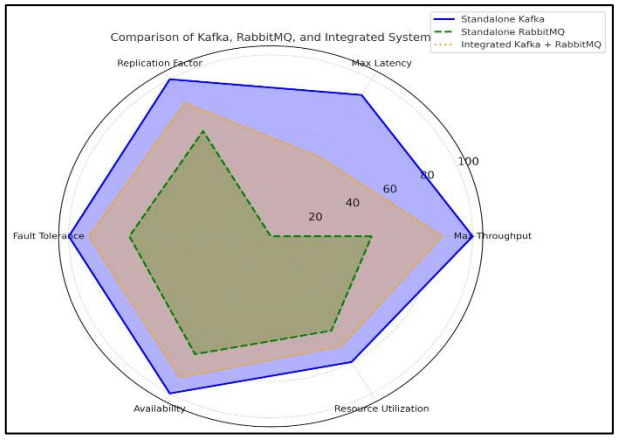


Figure 7: Comparison Of Kafka, RabbitMQ, And Integrated System

The setup that combines Kafka and RabbitMQ is a good compromise between the two. It has 85% of Kafka's output and 5% less delay, so it can be used in systems that need both real-time data streams and stable task lists. The replication factor (85%) and fault tolerance (90%) are much better than RabbitMQ's performance when used by itself, and availability goes up to 90%, showing that both systems work well together. At 70%, resource utilisation is also at its best, which shows that the combined setup's system management is working well.

7. CONCLUSION

Combining the capabilities of Apache Kafka with RabbitMQ to handle the complexity of real-time streaming and message queuing provides a transforming answer for contemporary data management systems. By means of thorough investigation and performance assessment, this study has shown how Kafka shines in low-latency, high-throughput contexts, thus guiding choice for real-time data streaming and massive data acquisition. while steady information glide is vital, Kafka's architecture—which boasts strong scalability, fault tolerance, and excessive availability—is perfect. For sure message styles that RabbitMQ can efficaciously handle, nonetheless, Kafka's regulations in state-of-the-art routing and asynchronous assignment handling make it less in shape. Conversely, RabbitMQ gives a super answer for undertaking queuing, complicated routing, and asynchronous communique. Although it runs at a lesser throughput and has more delay than Kafka, its advantages in flexibility, message durability, and handling of many communications patterns define it from other platforms. Combining Kafka with RabbitMQ results in a balanced configuration that lets companies use the features of both systems. While supporting several messaging demands including real-time data streams and background task processing, the integrated design improves scalability, fault tolerance, message durability, and throughput. Benchmark test and case study performance results show even more how much the integration offers in terms of fault tolerance, message throughput, and latency. Kafka's high-speed stream processing capacity and RabbitMQ's strong queuing mechanism together enable companies to create dependable, scalable, and efficient data pipes. This study ultimately portrays the Kafka-RabbitMQ integration as a future-proof option for optimising data pipeline performance in a variety of sectors, therefore guaranteeing that data transmission and processing are more flexible, robust, and effective than they have ever been.

REFERENCES

[1] Yang, S.; Zhang, X.; Liang, J.; Xu, N. Research on Optimization of Monitoring Nodes Based on the Entropy Weight Method for Underground Mining Ventilation. Sustainability 2023, 15, 14749.
[2] Folgado, F.J.; Calderón, D.; González, I.; Calderón, A.J. Review of Industry 4.0 from the Perspective of Automation and Supervision Systems: Definitions, Architectures and Recent Trends. Electronics 2024, 13, 782.
[3] Akanbi, A. Estemd: A distributed processing framework for environmental monitoring based on apache kafka streaming engine. In Proceedings of the 4th International Conference on Big Data Research, Tokyo, Japan, 27–29 November 2020; pp. 18–25.
[4] Chen, Z.; Kim, M.; Cui, Y. SaaS application mashup based on High Speed Message Processing. KSII Trans. Internet Inf. Syst. (TIIS) 2022, 16, 1446–1465.
[5] Howard, J.; Murashov, V.; Cauda, E.; Snawder, J. Advanced sensor technologies and the future of work. Am. J. Ind. Med. 2022, 65, 3–11.
[6] Waworundeng, J.M.S.; Tiwow, D.F.; Tulangi, L.M. Air pressure detection system on motorized vehicle tires based on iot platform. In Proceedings of the 2019 1st International Conference on Cybernetics and Intelligent System (ICORIS), Medan, Indonesia, 8–9 October 2022; IEEE: Piscataway, NJ, USA, 2019; Volume 1, pp. 251–256.
[7] Fay, C.D.; Healy, J.P.; Diamond, D. Advanced IoT Pressure Monitoring System for Real-Time Landfill Gas Management. Sensors 2023, 23, 7574.
[8] Alotaibi, A.; Barnawi, A. Securing massive IoT in 6G: Recent solutions, architectures, future directions. Internet Things 2023, 22, 100715.
[9] Gupta, N.; Gottapu, S.K.; Nayak, R.; Gupta, A.K.; Derawi, M.; Khakurel, J. (Eds.) Smart applications of Internet of Things (IoT) in healthcare. In Human-Machine Interaction and IoT Applications for a Smarter World, 1st ed.; CRC Press: Boca Raton, FL, USA, 2022.
[10] Banafaa, M.; Shayea, I.; Din, J.; Azmi, M.H.; Alashbi, A.; Daradkeh, Y.I.; Alhammadi, A. 6G Mobile Communication Technology: Requirements, Targets, Applications, Challenges, Advantages, and Opportunities. Alex. Eng. J. 2023, 64, 245–274.
[11] Lohitha, N.S.; Pounambal, M. Integrated publish/subscribe and push-pull method for cloud based IoT framework for real time data processing. Meas. Sensors 2023, 27, 100699.
[12] Mammela, O.; Karhula, P.; Makela, J. Scalability Analysis of Data Transfer in Massive Internet of Things Applications. In Proceedings of the 2019 IEEE Symposium on Computers and Communications (ISCC), Barcelona, Spain, 29 June–3 July 2019; pp. 1–7.
[13] Shi, Y.; Zhang, Y.; Chen, J. Cross-layer QoS enabled SDN-like publish/subscribe communication infrastructure for IoT. China Commun. 2020, 17, 149–167.
[14] Siddiqui, H.; Khendek, F.; Toeroe, M. Microservices based architectures for IoT systems—State-of-the-art review. Internet Things 2023, 23, 100854.