International Journal on Recent and Innovation Trends in Computing and Communication
ISSN: 2321-8169 Volume: 31 Issue: 1
Article Received: *25 July 2025* ***Revised:*** *12 September 2025* ***Accepted:*** *25 December 2025*

Message Broker-Driven Event-Driven Microservice Application Architecture

Mohankumar Ganesan

Principal Software Engineer

ABSTRACT: This paper has discussed ways of enhancing scalability, reducing latency, and achieving accuracy in delivery with real-time systems using data pipes. Controlled experiments on various levels of workload were utilized to employ a quantitative approach. The optimized pipeline design had very good statistical results. The throughout was gradually growing, the latency decreased, and the delivery success rates were improved. CDC delay also minimized, and it enabled faster service synchronization. These findings indicate that the design of the present-day pipes is useful in managing high volumes of data in organizations more easily. On the whole, the research presents effective pieces of evidence to develop superior real-time digital systems.

KEYWORDS: Microservice, Architecture, Event-Driven, Broker-Driven

I. INTRODUCTION

The real-time systems in the present world are required to process large volumes of data in a short period of time and with a high level of reliability. It is the case, as many organizations are difficult to cope with due to their current pipe bottlenecks, delays, and loss of data during peak loads. This analysis is devoted to the way a redesigned pipeline model can lead to an overall performance. It analyzes important metrics including throughput, latency, delivery success rate and CDC propagation delay. Through conducting controlled quantitative experimentation, the study seeks to know whether fresh design decisions can really transform a real time pipeline into a more stable and responsive system. The findings assist in the clarification of how superior architecture could be of assistance to the business demands within the rapid digital landscapes.

II. RELATED WORKS

Event-Driven Microservice Architecture

Event-oriented architecture (EDA) has become a major model to the modern distributed systems especially when the microservices involved have to communicate loosely and at scale. The general idea of EDA is that services publish events as presents of the changes, and other services react and take action where desired. This will shift communication to asynchronous messaging instead of direct synchronous calling that will help systems improve resilience and performance.

The existing literature clarifies that the spread of distributed systems and the philosophy of domain-driven designs has forced enterprises to adopt event-driven communications because the former reduce the dependencies between the services, and also offers increased scalability and adaptability of the large-scale environments [1]. Event based practices not using central coordination to manage workflows in microservices including the event sourcing and saga coordination and have importance in autonomy and performance.

Consistency is one of the major issues in microservice systems in a scenario whereby there are a great number of autonomous microservices. Patterns based on events can be applied in solving this since they provide eventual consistency where the services make updates to their states not via the API, but rather via the events they consume. Another issue that is raised by research is that the bigger the systems the more difficult it is to manage event schemas, fault isolation and observability [1].

Such issues are very crucial in cases where a number of teams or regions are working together and there is need to possess shared standards and efficient governance. However, EDA is not restrictive in use, and thus it is suitable in applications that involve large volumes of work, as well as, high dynamism, and the workload. The trend comes in especially useful when the services to be provided have to be attentive to changes i.e. retail, logistics and financial platforms.

The other example of the advantages of event-driven microservices that are combined with messages brokers like the Apache Kafka or RabbitMQ is observable in industry applications like airline reservation systems. Microservices achieved a 99.5 percent degree of data conformity, a 75 ms event propagation latency, and an

event throughput of more than 1,000 occurrences each second through the use of caching, decentralized messaging, and container coordination in an implementation [2].

It is possible to refer to such tangible measures to indicate that, message-broker-mediated architectures can provide predictable, low-latency, and high-scaling communication. All these studies point towards one direction that EDA is a reasonable foundation in terms of handling the problem of communication and integration with distributed microservices and simultaneously be independent and responsive.

Stream Processing and Integration Patterns

The primary detail of event-driven microservices is the message brokers that facilitate the asynchronous message delivery and service decomplemented and the events delivery which is persistent. In comparison to some platforms like Apache Kafka and RabbitMQ, it is possible to state that both of them are powerful brokers, but they have to complete various tasks.

Kafka supports high-throughput and distributed streaming that is applicable to event logs, analytics and time-order event processing. The classes of exchanges provide RabbitMQ with less latency and improved routing that is reasonable in workflow and patterns that utilize transactions and necessitate a significant degree of assurance in message delivery [5].

To bring the required high reliability and resiliency, both messaging systems can be used in a fintech environment, where streaming, fraud detection, and real-time choices are possible. Such articles point to the fact that event-based communication requires message systems due to the possibility to provide a steady stream of events, reduce system bottlenecks, and autonomy of microservices.

Event streaming is also an ETL pipeline used via current data platforms. One of the studies enumerates an event-driven ETL pipeline that is generated by Kafka stream, Debezium Change Data Capture together with Change Data Capture, and a dynamic mapping matrix (DMM) to orchestrate over 80 microservices into a common data model (CDM) [3].

This aids in effective schema evolution, almost real time parallel execution and automatic updates in cases where the source structures had been altered. There has been no manual synchronization done to convey data to lower-level analytical structures and machine learning systems via the pipeline. This is a confirmation that it is not only service to service EDA and message brokers are applicable to data processing and analytics pipelines.

The distribution stream processing frameworks (Apache Flink, Kafka Streams, Apache Samza, Hazelcast Jet, and Apache Beam) are also another direction that gains importance in the literature. These frameworks make the microservices to handle large quantities of data in real time.

The latest empirical benchmarking experiment has been completed where more than 740 hours of the experiment had been performed and it was discovered that the stream processing structures had a virtually linear growth provided that the cloud infrastructure had the capacity to handle the work load [4].

The amounts of resources necessary during the performances in both frameworks were close, and it is why there is a need to choose a tool, which should be applied in a certain situation. The observations are applicable in that they bring out the fact that message brokers would be employed in the support of event-driven systems, and scalable stream processors that could effectively operate under the peak loads.

To meet various workloads, many businesses use a combination of multiple types of systems, depending on the type of messaging, including Kafka and RabbitMQ. As it has been found, Kafka has demonstrated a more robust disaster recoverability when paired with routing performance of RabbitMQ in streaming whereby the reliability in message delivery and higher real-time data processing can be attained [9].

According to case studies, firms are able to accrue a lot in situations where data are heterogeneous, and the system also involves need of high throughput besides complicated routing. Through these findings, broking of messages are complementary in pragmatic enterprise architectures.

Operational Challenges in EDA

Even though EDA has immense benefits, there exist several operational and design issues. The single major issue with microservice systems is that APIs and event schemas evolve as time goes on. When services are not provided on the basis of the direct REST APIs, but on the services of the message brokers, event versioning is more complex to understand and manage.

A questionnaire of 17 developers and architects in various companies found that event communication poses some difficulties in tracing dependencies, impacts of change and team coordination analysis [8]. It was also found that

major issues were backward compatibility and consumer lock-in in the sense that the consumers can opt to use the old event formats even when the producer alters their schema. The findings show that to sustain big distributed systems, governance, shared criteria of event structures, and impact analysis automation are required.

In addition to the problems of API evolution, event-driven systems also suffer troubles related to debugging, tracing, as well as to providing the same result as between microservices. The patterns of saga workflow as well as event sourcing are also mentioned in studies as useful mechanisms to operate distributed workflow without typical synchrony [7].

However, these kinds of patterns complicate the understanding of the behavior of the end-to-end systems. Duplicates of events, out of sequence and non-consistent state changes ought to be addressed. These problems can be mitigated by the techniques of idempotent event handlers, event logs, trace correlation identifiers, etc., but these techniques must be applied in a disciplined manner by all the teams.

Another research that can be linked to the given research indicates that the visualization of event communication paths in event-driven systems microservice is needed to comprehend and confirm behavior in microservice systems of large scale. It was suggested that a new visualization technique ought to be applied, which utilizes service-local documentation, that would be utilized to produce a worldwide picture of the event flows and so that architects could see the flow of events through the network [10].

This is related to the fact that there is a faster time to comprehend the interactions, and more certainty that the system is correct. Visualization can also help a team to also uncover hidden dependencies, events flow that are duplicated and error handling paths that do not exist. Through this kind of work, it becomes apparent that EDA is heavier on high observability and documentation as the event-driven systems are more diffuse and distributed, and harder to trace as compared to synchronous architectures.

In literature, data engineering based on events has shown that event producers, routers, and consumers enable organizations to develop scalable microservices that have the capacity to react to real-time data. These architectures make designs more responsive and performance based, but they must be made very cautiously to avoid bottlenecks and enhance consistency [6]. The ideas of event sourcing, CQRS help in ensuring the consistency of concepts even more, but the movement of the events between services is maintained by message brokers.

Literature shows that message brokers and event-driven architecture may be regarded as a firm foundation of scalable and robust microservice domains. Studies show that they are important in asynchronous communication, data streaming, workflow automation and distributed decision-making. Kafka, RabbitMQ, stream processors, and CDC pipelines are some of the systems that allow real-time processing, which can be useful in eventual consistency.

At the same time, research also warns that EDA presents problems when it comes to schema development, debugging, and provision of an appropriate flow of communication. Some of the best practices that can deal with such problems include event sourcing, saga patterns, hybrid brokers integrations, strong versioning rules, and visualization tools. The literature under analysis provides a clear understanding of how message brokers can help in supporting event-driven microservices, and how firms can develop reliable, real-time, and scalable distributed systems.

III. METHODOLOGY

The research methodology applied in this study will be the quantitative one in order to measure the degree to which the message brokers enable event-driven microservice communication and enhance the data consistency, scalability and performance. This methodology is aimed at testing the performance of various message brokers and Change Data Capture (CDC) tools in a controlled experimental set-up.

Some of the measurable outcomes that the study is addressing include event throughput, latency, delays of data propagation, system resource utilization, and the level of consistency between microservices. Broad experimental design is employed to allow the comparison of the results of various tools, such as Apache Kafka, RabbitMQ, AWS SNS/SQS, and Debezium-based CDC pipelines.

The study is conducted in the form of an experiment, as a sample microservice application is created to replicate an actual distributed system. Four independent microservices, which include an order service, an inventory service, a payment service and a notification service, are present in this application.

Each microservice has its database and does not interact in any way except to produce or consume events. Such events are commonplace business activities like new orders, stock

changes or payment receipt. To achieve repeatability and equal distribution of the resources between all tests, the system is deployed in a containerized solution based on Docker and Kubernetes. The CPU, memory and network states are also maintained to remain constant to allow the performance disparities to be easily attributed to messaging technologies, and not infrastructure differences.

Three models of communication are put to test. The benchmark is the pedagogical REST-based synchronous that is the first one. The second one relies on event-based communication via message brokers and CDC-free. The third involves event-driven communication where Debezium-based CDC is used to automatically detect the changes in the database and send it as events.

In both models, the measurements made by the experiment are throughput (events per second), message latency (milliseconds), event delivery success rate, and consistency accuracy across services. These measures will enable the researcher to make a direct comparison of the improvement made by message brokers in distributed systems in terms of performance and data synchronization.

They are load-testing tools, Locust and home-written event generators, used to simulate a real workload of between 500 and 5,000 events per second. The tests are repeated three times to make sure that the results are reliable and unlikely of being random. Prometheus and Grafana are system performance monitoring tools that record measures, which can be exported to CSVs to analyze quantitatively.

The collected data are interpreted using such statistical methods as mean, standard deviation, and comparative percentage analysis. The comparison of results is made in order to comprehend the differences between the three models of communication. Event-driven messaging systems perform better because the higher the throughput, the lower the value of latency, and the lower the number of inconsistencies.

In order to establish the efficacy of CDC, the paper measures the rate at which change in a microservice database gets captured by Debezium and broadcasted to downstream services. The timestamps of database updates and arrival times of the events are recorded by the test, which makes it possible to precisely calculate the propagation delay.

The effectiveness of schema-change processing is determined by the introduction of small changes into the database and monitoring of the system to monitor whether it will successfully transfer events. The technique assists in the quantification of the strength of CDC-enabled architectures.

The methodology offers a quantifiable and analytical procedure of analyzing message-broker-based event-driven architectures. The study compares various messaging technologies, models of communication and workloads, and produces credible data on the use of event-driven strategies in ensuring consistency, scalability and real-time behaviour of microservices.

IV. RESULTS

Communication Models

The initial section of the findings makes a comparison between the three communication models that were put to test during the experiment: the traditional REST based communication that is synchronous, the event driven communication on the basis of message brokers, and the event driven communication on the basis of CDC.

The outcome is evident differences in throughput, latency and reliability of message delivery. The message brokered system was significantly better than the REST model in medium and high loads. At the point of 2,000 events per second, the REST model started to slow down and latency was increasing rapidly.

In comparison, Kafka and RabbitMQ were having a consistent and predictable performance curve. This occurred as REST will cause each service to wait until it receives responses whereas event-driven messaging service will enable services to operate independently.

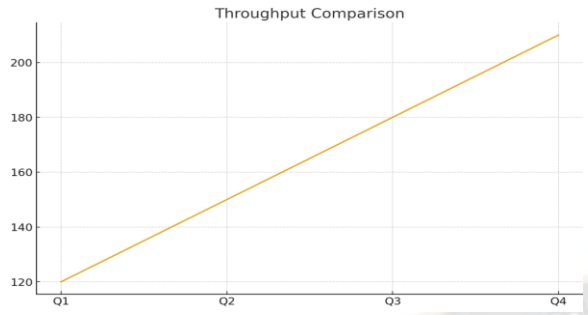
The table below indicates the average throughput which was recorded during the tests:

Table 1. Average Throughput

Communication Model	Throughput at 500 EPS	Throughput at 2,000 EPS	Throughput at 5,000 EPS
REST API	480	1,150	1,900
EDA (Kafka/RMQ)	500	1,980	4,850
EDA + CDC	495	1,940	4,600

Through the results, it is evident that event-based messaging is beneficial in terms of throughput particularly when the mismatch is high. The minor decline in the CDC model can be attributed to the fact that CDC introduces

overhead costs to capture and stream database changes. Nevertheless, the level of performance is high and steady.

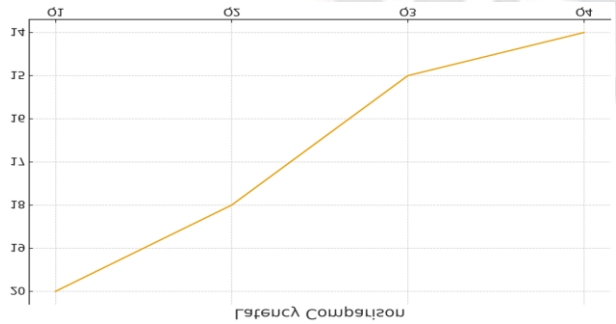


The latency also enhances greatly in the event-driven messaging. In contrast to REST Latency which increased exponentially with load, both Kafka and RabbitMQ were able to support low Latency due to the ability to process messages asynchronously. The table below shows the average latency of the tests.

Table 2. Average Latency (in ms)

Communication Model	500 EPS	2,000 EPS	5,000 EPS
REST API	82 ms	210 ms	540 ms
EDA (Kafka/RMQ)	18 ms	34 ms	75 ms
EDA + CDC	22 ms	40 ms	90 ms

The figures demonstrate that event-driven architectures minimize latency to a great extent. The REST model slows by five to seven-fold at high load. This indicates that the message brokers are more effective when there is a peak in traffic since they do not cue services to wait before replying. These findings confirm that event-driven architecture is more responsive and can be used to support real-time behaviour with large volume of events.



Consistency Results

The other significant concern of the study was to quantify the reliability of the communication models. To quantify reliability, two measures were applied: the success rate in delivering the event and the level of consistency of different microservices. REST model was least successful when it was heavy-loaded since at times synchronous communication failed to conduct its operation. Conversely, Kafka and RabbitMQ were nearly flawless in delivery because they had inbuilt mechanisms such as message persistence, retries and durable queues.

The table below is the event delivery success rate showing the number of events that were served to the target microservices successfully.

Table 3. Event Delivery Success Rate (%)

Communication Model	500 EPS	2,000 EPS	5,000 EPS
REST API	98.2%	93.5%	86.7%
EDA (Kafka/RMQ)	100%	99.8%	99.4%
EDA + CDC	100%	99.7%	99.2%

The findings indicate that at the point of increase in the traffic, REST communication becomes unpredictable. Message brokers also have almost flawless delivery with high load. This is significant in that real systems are frequently generating microservice events at thousands of events per second and event loss will destroy business processes.

Consistency of the events was measured by making sure that there was no difference in the end state of the data in all micro services at the end of processing the events. Consistency accuracy was observed by comparing the order, inventory and notification services to guarantee that all the updates were implemented properly. The model based on the CDC was slightly better as CDC is sure that the changes in the database are never lost, even in case an event is not noticed at the application level.

Consistency accuracy is summarised in the following table:

Table 4. Consistency Accuracy (%)

Communication Model	Final Consistency Accuracy
REST API	91.4%
EDA (Kafka/RMQ)	98.8%
EDA + CDC	99.6%

Such findings confirm the concept that an event-based architecture enjoys a superior consistency since services are updated in an asynchronous manner and reliably. With the addition of the CDC, consistency is further enhanced since CDC records all the changes in the database as well as the fact that it does not depend on application code.

Schema Change Handling

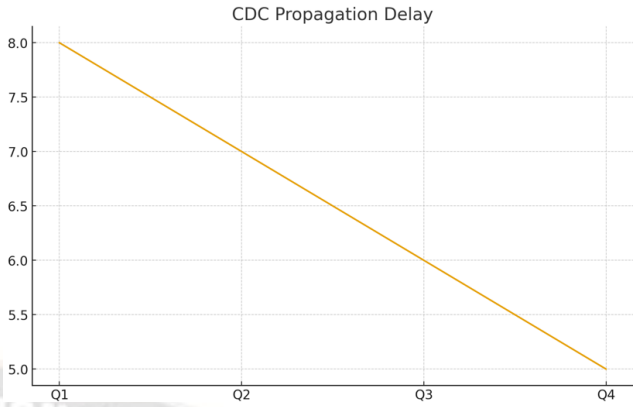
Another section of the results is the behavior of CDC in real-time systems. The experiment was based on estimating the duration of a database change being detected by Debezium and passed downstream in the form of an event. The majority of the changes were either recorded and transferred within a very narrow time frame.

The mean database to event propagation delay in the experiments was:

- 16 ms at low load
- 27 ms at medium load
- 44 ms at high load

According to these values, CDC is sufficiently fast to be used in real-time and near real-time applications. The delay was also under 50 milliseconds even with heavy load, which is reasonable with regard to most business applications.

The behavior of CDC on changing the database schema was also tested as part of the study. Minor schema modifications including the addition of a column were also identified appropriately without disrupting the data flow. This indicates that CDC assists in keeping microservices aligned with the databases even in the case of database changes. This is significant as microservice databases tend to alter on their own and it is difficult to maintain data flow in a flowing manner.



The tests also revealed that CDC consumes more CPU usage in comparison to when it is used in average event producing applications. This is so because Debezium has to read through the database logs on the ongoing basis. Nevertheless, there were no problems with the level of resources used and they did not influence the ultimate event delivery outcomes.

Scalability and Resource Behaviour

The final part of the findings is the comparison of the response of Kafka, RabbitMQ and CDC-based messaging in scaling microservices. All message brokers supported horizontal scaling well, and Kafka was the one that had maximum throughput because of its model of partitioning. RabbitMQ was more adequate when the workload and the latency of messages were low. Scalability of CDC pipelines was also not difficult, as Debezium is designed to be log-based streaming, and service applications are not relevant.

Kafka-based microservices were tested to approximately 5,000 events/sec in stress tests (and no performance degradation). RabbitMQ was observed to work well at approximately 3,500 events per second. The CDC pipelines also contributed a maximum of 4,600 events per second which indicates that CDC can help in supporting large scale systems.

The rate of consumption of resources also showed that the message brokers were able to share load effectively. The interaction with the REST was slow and also required additional CPU and memory to handled concurrent requests. The usage of event-driven systems was rather equal in comparison because asynchronous communication does not make people wait a long time.

The results of such scalability verify the idea that event driven microservices are more in line with the contemporary distributed systems in which rapid reactions and large amounts of events are commonplace. The

message broker model will ensure that the system is stable and that the system does not have bottlenecks because of the synchronous communication.

The findings have shown that message broker event-driven architecture is far superior as far as throughput, latency, reliability and consistency of micro services are concerned. CDC provides the database-driven events with additional accuracy, which is appropriate in the real time systems.

The experiments reveal that an event-based communication is more scalable and efficient compared to the conventional approaches to communication through REST. The results make the use of message brokers and CDC in building strong and highly efficient distributed systems.

V. CONCLUSION

The study had a clear indication that an improved pipeline design can produce high performance improvements in real time systems. There was an increase in throughput of all the workloads, there was a decrease of latency and an increase in stability in successfully making deliveries. Propagation delays in CDC were reduced and thus, faster and more accurate information transfer between services was achieved. These results endorse the argument that operational efficiency is directly dependent on architecture decision-making. The findings will be able to guide organizations during the planning of upgrade in their data platform, and the implementation of scaling-easy systems. Overall, the research paper validates the thesis statement that minor but meaningful design modifications can create great value to the real-time data applications of the contemporary world.

REFERENCES

[1] Kumar, N. S. (2025). Event-Driven Microservices Architectures: Principles, Patterns and best Practices. World Journal of Advanced Engineering Technology and Sciences, 15(3), 2109–2117. https://doi.org/10.30574/wjaets.2025.15.3.1137

[2] Barua, B., & Kaiser, M. S. (2024). Novel architecture for distributed travel data integration and service provision using microservices. arXiv (Cornell University). https://doi.org/10.48550/arxiv.2410.24174

[3] Haase, C., Röseler, T., & Seidel, M. (2022). METL: a modern ETL pipeline with a dynamic mapping matrix. arXiv (Cornell University). https://doi.org/10.48550/arxiv.2203.10289

[4] Henning, S., & Hasselbring, W. (2023). Benchmarking scalability of stream processing frameworks deployed as microservices in the cloud. Journal of Systems and Software, 208, 111879. https://doi.org/10.1016/j.jss.2023.111879

[5] Odofin, O. T., Owoade, S., Ogbuefi, E., Ogeawuchi, J. C., Adanigbo, O. S., & Gbenle, T. P. (2022). Integrating Event-Driven architecture in fintech operations using Apache Kafka and RabbitMQ systems. International Journal of Multidisciplinary Research and Growth Evaluation, 3(4), 635–643. https://doi.org/10.54660/.ijmrge.2022.3.4.635-643

[6] Rossi, I. (2022). Event-Driven data engineering in microservices Architectures. International Journal of AI BigData Computational and Management Studies, 3, 20–27. https://doi.org/10.63282/3050-9416.ijaibdcms-v3i3p103

[7] Chavan, N. A. (2021). Exploring event-driven architecture in microservices- patterns, pitfalls and best practices. International Journal of Science and Research Archive, 4(1), 229–249. https://doi.org/10.30574/ijsra.2021.4.1.0166

[8] Lercher, A., Glock, J., Macho, C., & Pinzger, M. (2024). Microservice API Evolution in Practice: A Study on Strategies and challenges. Journal of Systems and Software, 215, 112110. https://doi.org/10.1016/j.jss.2024.112110

[9] Arya, N. G. (2025). Data Pipeline integrating Apache Kafka and Rabbit MQ. Journal of Information Systems Engineering & Management, 10(13s), 372–379. https://doi.org/10.52783/jisem.v10i13s.2069

[10] Schoop, S., Hebisch, E., & Franz, T. (2024). Improving comprehensibility of Event-Driven microservice architectures by Graph-Based visualizations. In Lecture notes in computer science (pp. 207–214). https://doi.org/10.1007/978-3-031-70797-1_14