The Impact of Messaging Queues on Real-Time Data Exchange in Large-Scale applications

Rajesh Kotha

Senior Software Engineer at Kroger

Abstract:
Large-scale applications rely on real-time data interchange to remain reliable, scalable, and perform well. Messaging queues provide the necessary solution to keep data flowing even when high loads or component failures occur, enabling asynchronous communication between services. This paper investigates the concept of messaging queues; it describes a few types, such as publish-subscribe and point-to-point systems, with examples taken from cloud-based solutions like Amazon SQS and RabbitMQ, Apache Kafka, among other systems. It studies how they affect fault tolerance, scalability, and system performance, relating real-world examples from the financial, e-commerce, and Internet of Things sectors. The study also explores the general role of messaging queues in enabling modern networked systems and determines their importance in developing technologies. The conclusion stresses the role messaging queues already play within real-time systems and provides recommendations for further investigations, particularly about decreasing latency and increasing security in distributed environments.

Keywords: Message queues, real-time message exchange, distributed systems, scalability, RabbitMQ, Apache Kafka, fault tolerance, asynchronous message exchange, Large-Scale applications, and microservices

1. Introduction

Most modern applications generate millions of data points in real time; hence, there is great demand for practical ways for these systems to exchange information [2]. Messaging queues are the critical infrastructure that needs to be implemented to handle message exchange between services or applications. This ensures that the data flow is asynchronously controlled, which plays a vital role in keeping the integrity and speed of such extensive systems intact. Messaging queues decouple producers from the consumers of the data to enable various parts of an application to communicate without direct dependency on one another's availability or performance. While the organization continuously improves its ability to cope with increased data demand, messaging queues have become necessary in large-scale application developments to allow real-time data exchange while continuing operations on connected systems. This paper examines the various types of messaging queues and their applications in driving real-time data exchange

2. Literature Review

Messaging queues were among the oldest building blocks for a distributed system, allowing different components to communicate asynchronously. Messaging queues are middleware that indirectly enables applications to communicate through passing messages via a queue; thus, it decouples the sender and receiver. This makes the system scalable because different services can run independently of others' speeds and operation availability, which is an essential requirement of a large-scale application.

Various works have pointed out messaging queues' role in assuring better performance and reliability for distributed systems. Messaging queues reduce system downtime by buffering messages when there is high

traffic so that the services are not overwhelmed; these messages are not lost when a service is down [6]. This property is particularly critical in real-time systems where loss or delay of information may lead to catastrophic results. Another example of real-time data streams in large applications' handling of the messaging queue concerns fault tolerance and message persistence using RabbitMQ and Apache Kafka.
Various messaging queues provide different feature sets for specific needs. For instance, RabbitMQ implements AMQP and is very powerful in message routing, finding wide usage in enterprise applications. On the contrary, Kafka is designed for high-throughput distributed environments and does an excellent job in the real-time management of data streams [5]. Both systems show the diversity of messaging queues to support real-time communication across scalable applications. Cloud-based message queues, such as Amazon SQS and Google Cloud Pub/Sub, have become increasingly adopted, enhancing even more scalability by providing distributed managed services. This is important because it will allow an organization to easily handle fluctuating workloads and ensure real-time data exchange remains efficient even when increasing demands exist.

3. Problem Statement

In large-scale applications, emergent challenges of complex real-time data exchange among multiple services have started to appear. This involves traditional means of communication, such as direct API calls and synchronous messaging, resulting in bottlenecks in the core or latency issues, or both, under heavy loads. This concern reduces the scalability of performance-critical apps. A service interruption or delay could make the system unstable since data flow cannot be organized. A robust and scalable approach, such as messaging queues, is needed to meet this challenge, which can easily handle asynchronous communication and ensure smooth data transfer.

4. Solutions

Large systems use message queues for asynchronous, decoupled service communication and real-time data transfer. They eliminate data-flow bottlenecks and sustain application performance during service failures or high traffic. To solve the problem, one must understand messaging queue types, how to use them, and how they affect system architecture and design.

A. Types of Messaging Queues

Message queue features allow several real-time data interchange use cases. An adequate messaging queue is needed to manage data volume with low latency and exemplary dependability.

a. Point-to-Point (P2P) Messaging Queues

Point-to-point messaging routes messages from one producer to one consumer [12]. Once a message is consumed, it is removed from the queue. This type of queue should be used when application design must guarantee that no more than one receiver processes each message. There are two types of point-to-point messaging queues: Fire – and – forget messaging (one way) and request – response messaging. One example application of point-to-point messaging queue is Amazon Simple Queue Service-SQS, which supports P2P reliable and scalable messaging for distributed applications.

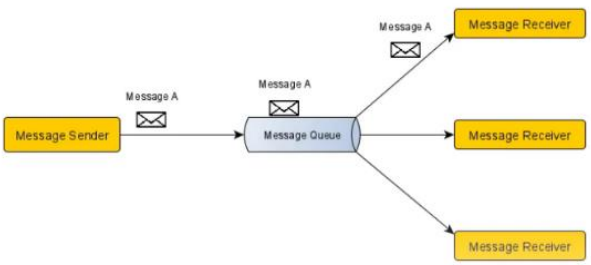

Fig. 1. Illustration of point to point messaging queue. Adapted from [12]

Use Case 1:

P2P queues are helpful in systems whereby each task should be processed by one worker or service [12]. For example, a job processing application could place tasks into a queue and then be picked up by one worker at a time. Publish-Subscribe Messaging Queues

In the Publish-subscribe messaging queue, multiple consumers can subscribe to a single topic. All subscribers will then receive a copy of each message. This pattern is suitable when the same data must be distributed to multiple services or systems in real time. Google Cloud Pub/Sub and Apache Kafka are popular systems that apply this pattern.

Use Case 2:

Pub/Sub queues will be suitable for real-time analytics and monitoring systems where multiple consumers should get the same messages to react at once. Examples of such consumers include and are not limited to dashboards, data processors, and alerting systems.

b. Message Brokers with Routing Capabilities

Some messaging systems, such as RabbitMQ, also offer queuing capabilities and advanced routing features. Using message brokers, messages may be dynamically routed to various queues based on specific rules or patterns. This can be utilized to facilitate the flexible distribution of messages over multiple services. This includes publish – subscribe messaging model. A publish-subscribe approach is utilized when a message creator requires sending an identical message to multiple recipients at once [7]. By using this pattern of communications, a message is published by the message broker to a subject matter, where it is sorted, stored, processed, and sent to subscribers.

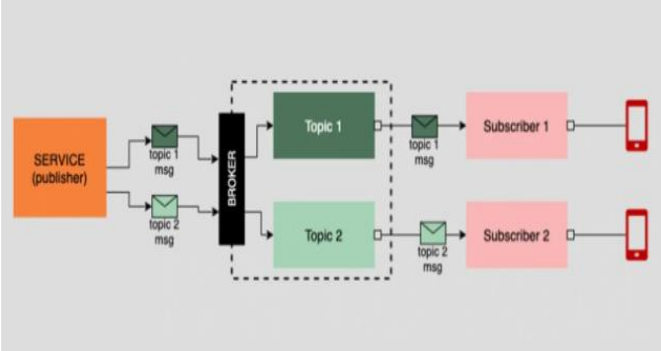

Fig. 2. Illustration of publish-subscribe model. Adapted from [7].

Use Case 1:

A typical use case could be that, in some retail systems where it takes customer orders and processes them, it would send messages based on the product type of an order. For example, some orders will go one way to a service for physical products, while other orders will go to a digital service for e-products.

B. Key Components of Messaging Queues in Large-Scale Applications

Large-scale applications require several critical pieces if one wants to realize the full potential of messaging queues. These are:

a. Asynchronous Processing

One of the critical benefits of messaging queues is that they provide asynchronous processing. For example, data producers, such as microservices, do not need to wait for consumers to process a message before going ahead and continuing their operations. In turn, messages can simply reside in a queue while waiting for an available consumer to realize or process them. This kind of asynchrony helps bottleneck pressure and allows large volumes of real-time data to be handled without overwhelming a particular service.

b. Load balancing

Messaging queues at runtime can distribute the load over multiple consumers. Even during high traffic, the system would still be responsive. During sudden spikes in traffic, additional consumers can be spun up to process the backlog messages, assuring horizontal scaling. It is vital for large-scale applications whose traffic patterns may be pretty unpredictable. It may vary based on external factors, such as user demand or the system load.

c. Fault Tolerance and Reliability

Large-scale applications are bound to have their constituent services fail occasionally. Messaging queues ensure fault tolerance by persisting messages until they have succeeded in processing [1]. For example, if a consumer fails, the message will stay in the queue, waiting for another available consumer or being picked up once it becomes available. This way, no data is lost; in case of failure, the system goes back to its previous state without losing important information. An example is how Kafka ensures that messages are durable- aside from writing messages to disk, it replicates them over multiple servers, making it highly fault-tolerant.

d. Message Acknowledgment and Redelivery

Maintaining data integrity is the reason most messaging queues provide an acknowledgment mechanism by which a consumer needs to acknowledge the successful receipt and processing of the message. If the consumer does not recognize the message in a given amount of time, the messaging queue can deliver the same message to another free consumer. Again, this avoids any message loss during consumer failures or crashes. Examples include RabbitMQ, which has this feature to help maintain consistency and reliability in data.

C. Use of Messaging Queues in Real-Time Data Exchange

Messaging queues help in real-time data exchange by allowing services to communicate smoothly with each other even when the environment is vast and distributed. Following are some ways in which messaging queues contribute to the facilitation of real-time data exchange:

a. Handling of Volume of Data

In large-scale applications, data exchanged between services can be pretty huge. Messaging queues work at buffering this data to prevent services from being overwhelmed by spikes in traffic [11]. Kafka is good for real-time streaming data and has high-throughput abilities. For example, a financial trading application could queue millions of transactions and get them processed in real time without losing a single transaction.

b. Real-Time Event Processing

Messaging queues often find their deployment in event-driven architectures, where systems are supposed to take some real-time action against events [11]. In such architectures, the publishing services will emit events into a queue to which subscribers subscribe to act upon when these events become available. For instance, an e-commerce platform might use a messaging queue to notify various systems when customers place orders, triggering real-time updates to inventory, shipping, and customer service systems.

c. Decoupling Microservices

In a microservice architecture, messaging queues can establish loose coupling among these services. Each service can interact with the others irrespective of whether the services are up or not and regardless of their performance. Such decoupling is essential in large-scale applications where services will be independently deployed and, as such, would not face failures or updates simultaneously. Messaging queues, when used, ensure that even when one service goes down, the other services continue the process without any problem.

D. Real-World Examples of Messaging Queues in Action

Be it ride-sharing or media streamers, several big applications have already tried messaging queues for real-time data exchange. Some examples are:

a. Uber

Apache Kafka was used in Uber to process billions of messages a day with real-time analytics, ride requests,

and matching between riders and drivers [10]. This helped deal with the huge volume of data generated by Uber users and services, thus allowing scalability and responsiveness even under very high loads.

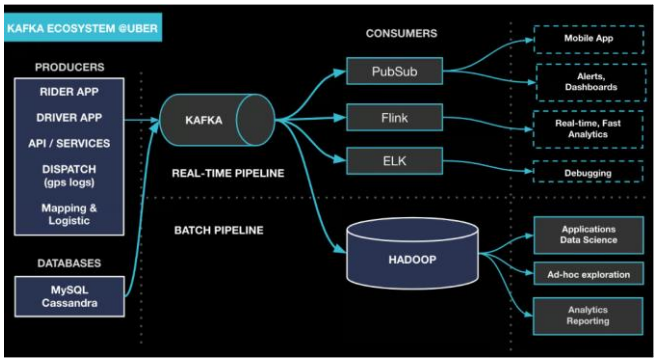

Fig. 3. Kafka Ecosystem at Uber. Adapted from [10]

b. Netflix

For example, Netflix combines messaging queues and stream processing to handle real-time data around users' viewing habits and recommendations. Kafka supports Netflix's data streams through its processing and distribution of a variety of services, hence creating a seamless user experience [9].

E. Challenges and Considerations

Whereas messaging queues introduce many advantages, they also come with challenges that must be considered.

a. Message Ordering

Some applications have a requirement where message order is critical. While some messaging queues, such as Kafka, can provide strong assurances in message processing orders, others cannot [5]. The developer must carefully select a messaging queue that meets the particular ordering requirements for their application.

b. Latency

Messaging queues reduce bottlenecks within systems; however, due to the nature of messages queuing and being handled asynchronously, latency is definitely introduced [5]. In real-time systems that have requirements for low latency, this lag has to be minimal; proper configuration is needed, and thus, optimization of the messaging queue is required.

c. Security

Security is very much crucial in wide-scale distributed systems for the safety of the data being exchanged [11]. Messaging queues need proper protection from unauthorized access, data breaches, or manipulation of messages. End-to-end encryption and strict access controls are two solutions that can mitigate these types of threats.

5. Impact

Messaging queues significantly affect large-scale applications as they can make the system much more scalable, fault-tolerant, and high-performing. Using a messaging queue effectively decouples an application's producer and consumer parts, hence letting services work independently. This reduces dependencies that often cause bottlenecks in the real-time exchange of data. Consequently, the system's overall resiliency is greatly improved because failures in one part do not disrupt the workflow.

Second, the efficiency at which message queues can handle large volumes of data allows large applications to handle sudden traffic spikes without performance degradation. To put this into perspective, Kafka can handle millions of messages per second. RabbitMQ can handle millions of messages per second to ensure that

real-time applications such as e-commerce platforms or financial trading systems work seamlessly under high-load conditions.

Messaging queues further enhance fault tolerance because of the assurance they offer regarding the reliability of message delivery. Messages in queues do not get deleted but remain there, waiting for system uptime so their processing may be completed. This enhances system reliability and reduces the risk of data loss.

Message queues generally allow for more flexible, scalable, and resilient architectures that enable real-time communication for industries ranging from financial services to logistics, where performance and reliability are paramount. Also, messaging queues find applications in many industries and real-time data exchange applications involving asynchronous communications between services.

6. Uses

Messaging queues have various uses. On an e-commerce site, for instance, messaging queues could be utilized in order processing: inventory systems, payment gateways, and shipping services are decoupled to ensure that all components function well without delays or failures across the system.

Messaging queues also allow high-frequency trading and payment processing in financial services, where large transactions must be dealt with reliably in real time. Kafka and RabbitMQ provide rapid execution of trades and avoid bottlenecking when processing data on payments [11]. The messaging queue also forms the backbone of microservices-based architectures where the services operate independently and communicate without tight coupling. This further enables horizontal scaling of services whereby, when instances increase, the work distribution can be highly effective and helpful for large-scale, cloud-based applications.

Another vital use case involves IoT systems, where there is a need to collect data from thousands of devices, process it, and analyze it in real-time [4]. Message queues ensure the continuity and reliability of such a flow from edge devices to central processing for analysis, thus making them essential in applications like smart cities and health monitoring.

7. Scope

These messaging queues can be applied to extensive applications related to finance, e-commerce, health care, telecommunications, and many others. Since the infrastructures are increasingly complex in the digital world, the need for a fault-tolerant, scalable, real-time communication system is felt more than ever. Messaging queue has acquired importance in microservices architecture, enabling decoupling and asynchronous communication necessary for smooth scaling and flexible deployment over distributed systems.

Messaging queues are at the heart of the cloud-native environment for handling these unpredictable loads in modern applications [8]. As more and more companies move towards the cloud, demand for resilient messaging infrastructure will keep growing to maintain large-scale systems capable of processing large portions of real-time data while ensuring high system availability.

Messaging queue applicability spreads from conventional industries to the latest ones, including IoT, machine learning, and artificial intelligence [3]. Since many of these technologies depend on a quick and efficient data flow between devices, sensors, and processors, messaging queues would become instrumental in managing real-time data exchange applications like autonomous vehicles, smart cities, or predictive analytics. In general, areas where messaging queues apply, are vast, and their use is becoming a must for any decent, modern data-driven application, regardless of the sector.

8. Conclusion

Messaging queues provide the fundamental building block for large-scale applications that can exchange data in a reliable, scalable, and real-time manner. They decouple services from one another and communicate through asynchronous interaction, whereby their presence helps improve system performance and reduces

bottlenecks under huge loads while being fault tolerant. Key messaging queue technologies, like Kafka, RabbitMQ, and cloud-based solutions, have become indispensable in finance, e-commerce, and IoT, where adequate data flow is desirable and indispensable. So, with the further evolution of more complex and large systems, messaging queues will remain crucial in making an application handle volume of real-time data in a manner that is not only practical but sustainable. Possible future research may be reducing the minimum latency introduced by the messaging queues for real-time applications depending on stringent timing, such as autonomous systems or high-frequency trading platforms.

References

1. I. Álvarez, A. Ballesteros, M. Barranco, D. Gessner, S. Djerasevic, and J. Proenza, "Fault Tolerance in Highly Reliable Ethernet-Based Industrial Systems," Proceedings of the IEEE, vol. 107, no. 6, pp. 977–1010, Jun. 2019, doi: https://doi.org/10.1109/JPROC.2019.2914589.
2. R. A. Ariyaluran Habeeb, F. Nasaruddin, A. Gani, I. A. Targio Hashem, E. Ahmed, and M. Imran, "Real-time big data processing for anomaly detection: A Survey," International Journal of Information Management, vol. 45, pp. 289–307, Apr. 2019, doi: https://doi.org/10.1016/j.ijinfomgt.2018.08.006.
3. E. Al-Masri et al., "Investigating Messaging Protocols for the Internet of Things (IoT)," IEEE Access, vol. 8, pp. 94880–94911, 2020, doi: https://doi.org/10.1109/access.2020.2993363.
4. P. Ferrari, A. Flammini, E. Sisinni, S. Rinaldi, D. Brandao, and M. S. Rocha, "Delay Estimation of Industrial IoT Applications Based on Messaging Protocols," IEEE Transactions on Instrumentation and Measurement, vol. 67, no. 9, pp. 2188–2199, Sep. 2018, doi: https://doi.org/10.1109/tim.2018.2813798.
5. G. Fu, Y. Zhang, and G. Yu, "A Fair Comparison of Message Queuing Systems," IEEE Access, vol. 9, pp. 421–432, 2021, doi: https://doi.org/10.1109/ACCESS.2020.3046503.
6. "Stream Buffers vs Message Buffers vs Queues," FreeRTOS Community Forums, Dec. 06, 2019. https://forums.freertos.org/t/stream-buffers-vs-message-buffers-vs-queues/8367
7. "What is a Message Broker?" Vmware. Available: https://www.vmware.com/topics/message-brokers
8. "Cloud native high availability with IBM MQ," https://community.ibm.com/community/user/integration/people/david-ware1, "Cloud native high availability with IBM MQ," Ibm.com, Mar. 25, 2021. https://community.ibm.com/community/user/integration/blogs/david-ware1/2021/03/23/native-ha-cloud-native-high-availability.
9. keenioblog, "Multi-tenant vs. Dedicated Event Streaming - Which to Choose," Keen, Mar. 15, 2021. https://keen.io/blog/multi-tenant-vs-dedicated-event-streaming/.
10. "Disaster Recovery for Multi-Region Kafka at Uber| Uber Blog," Uber Blog, Dec. 21, 2020. Available: https://www.uber.com/blog/kafka/?uclick_id=fdba3880-f9a9-470d-b72d-db84f0b721d8
11. C. Wang, C. Gill, and C. Lu, "Real-Time Middleware for Cyber-Physical Event Processing," ACM Transactions on Cyber-Physical Systems, vol. 3, no. 3, pp. 1–25, Oct. 2019, doi: https://doi.org/10.1145/3218816.
12. T. Tran, "Point-to-Point and Publish/Subscribe Messaging model," Medium, Jan. 08, 2021. https://programmingsharing.com/point-to-point-and-publish-subscribe-messaging-model-2efc4d2b6726