System Design: The Distributed Messaging Queue

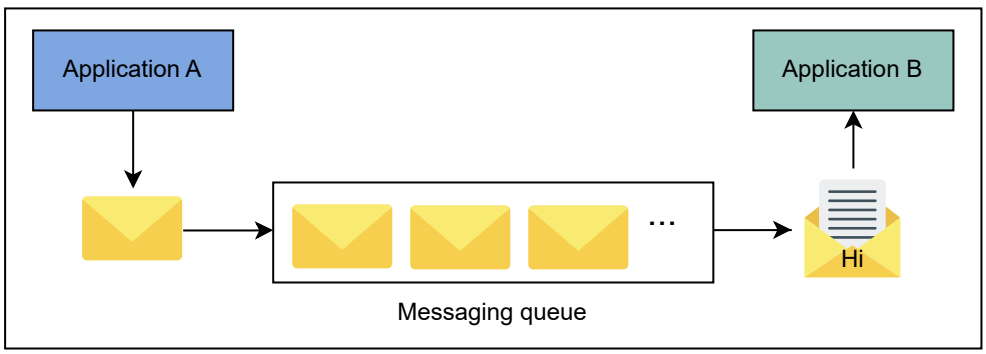
Learn about the messaging queue, why we use it, and important use cases.

We'll cover the following

What is a messaging queue?

A **messaging queue** is an intermediate component between the interacting entities known as *producers* and *consumers*. The **producer** produces messages and places them in the queue, while the **consumer** retrieves the messages from the queue and processes them. There might be multiple producers and consumers interacting with the queue at the same time.

Here is an illustration of two applications interacting via a single messaging queue:



An example of two applications interacting via a single messaging queue

Motivation

A messaging queue has several advantages and use cases.

- **Improved performance:** Improved performance: A messaging queue enables asynchronous communication between the two interacting entities, producers and consumers, and eliminates their relative speed difference. A producer puts messages in the queue without waiting for the consumers. Similarly, a consumer processes the messages when they become available. Moreover, queues are often used to separate out client-perceived latency. For example, instead of waiting for a specific task that's taking a long time to complete, the producer process sends a message, which is kept in a queue if there are multiple requests, for the required task and continues its operations. The consumer can notify us about the fate of the processing, whether a success or failure, by using another queue.
- **Better reliability:** The separation of interacting entities via a messaging queue makes the system more fault tolerant. For example, a producer or consumer can fail independently without affecting the others and restart later. Moreover, replicating the messaging queue on multiple servers ensures the system's availability if one or more servers are down.
- **Granular scalability:** Asynchronous communication makes the system more scalable. For example, many processes can communicate via a messaging queue. In addition, when the number of requests increases, we distribute the workload across several consumers. So, an applicati is in full control to tweak the number of producer or consumer processes according to its current need.

- **Easy decoupling:** A messaging queue decouples dependencies among different entities in a system. The interacting entities communicate via messages and are kept unaware of each other's internal working mechanisms.
- **Rate limiting:** Messaging queues also help absorb any load spikes and prevent services from becoming overloaded, acting as a rudimentary form of rate limiting when there is a need to avoid dropping any incoming request.
- **Priority queue:** Multiple queues can be used to implement different priorities—for example, one queue for each priority—and give more service time to a higher priority queue.

Messaging queue use cases

A messaging queue has many use cases, both in single-server and distributed environments. For example, it can be used for interprocess communication within one operating system. It also enables communication between processes in a distributed environment. Some of the use cases of a messaging queue are discussed below.

1. **Sending many emails:** Emails are used for numerous purposes, such as sharing information, account verification, resetting passwords, marketing campaigns, and more. All of these emails written for different purposes don't need immediate processing and, therefore, they don't disturb the system's core functionality. A messaging queue can help coordinate a large number of emails between different senders and receivers in such cases.
2. **Data post-processing:** Many multimedia applications need to process content for different viewer needs, such as for consumption on a mob phone and a smart television. Oftentimes, applications upload the content into a store and use a messaging queue for post-processing of

content offline. Doing this substantially reduces client-perceived latency and enables the service to schedule the offline work at some appropriate time—probably late at night when the compute capacity is less busy.

3. **Recommender systems:** Some platforms use recommender systems to provide preferred content or information to a user. The recommender system takes the user's historical data, processes it, and predicts relevant content or information. Since this is a time-consuming task, a messaging queue can be incorporated between the recommender system and requesting processes to increase and quicken performance.

How do we design a distributed messaging queue?

We divide the design of a distributed messaging queue into the following five lessons:

1. **Requirements:** Here, we focus on the functional and non-functional requirements of designing a distributed messaging queue. We also discuss a single server messaging queue and its drawbacks in this lesson.
2. **Design consideration:** In this lesson, we discuss some important factors that may affect the design of a distributed messaging queue—for example, the order of placing messages in a queue, their extraction, their visibility in the queue, and the concurrency of incoming messages.
3. **Design:** In this lesson, we discuss the design of a distributed messaging queue in detail. We also describe the process of replication of queues and the interaction between various building blocks involved in the design.
4. **Evaluation:** In this lesson, we evaluate the design of a distributed messaging queue based on its functional and non-functional requirements.

5. **Quiz:** At the end of the chapter, we evaluate understanding of the design of a distributed messages queue via a quiz.

Let's start by understanding the requirements of designing a distributed messaging queue.

Back

Memcached versus Redis

Next

Requirements of a Distributed Messaging Qu...

Mark as Completed