Back To Module Home

Distributed Systems

0% completed

Introduction to Distributed Systems

Mark Module as Completed

The Tale of Exactly-Once Semantics

Know the story behind the exactly-once semantics.

We'll cover the following

- Difference between delivery and processing
- Other delivery semantics

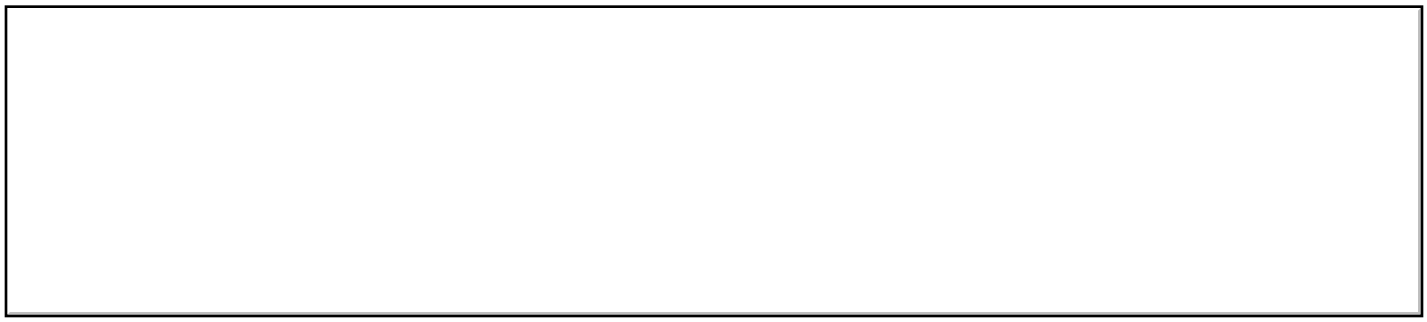
Multiple deliveries of a message#

Various nodes of a distributed system communicate with each other through the exchange of messages.

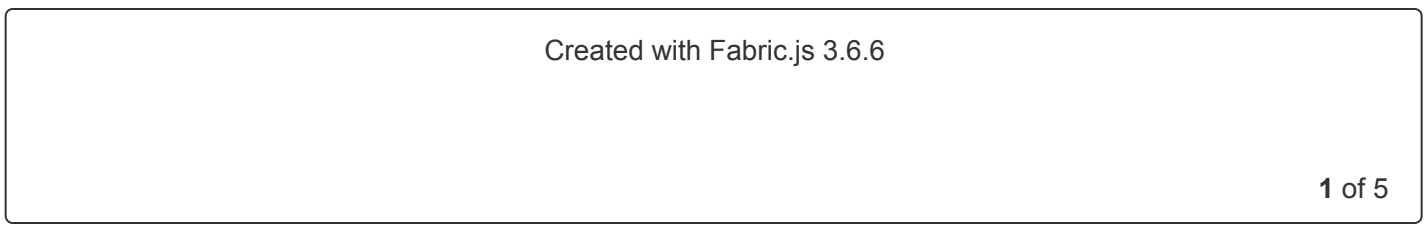
As the network is not reliable, these messages might get lost. Of course, to cope with this, nodes can retry with the hope that the network will recover at some point and deliver the message.

However, this means that the nodes may deliver messages multiple times because the sender can't know what really happens.

The following illustration shows what happens when a node doesn't deliver a message at all.



The following illustration shows a message that a node delivers twice.



This duplicate delivery of a message can create disastrous side effects.

Example consequence#

Think about what would happen if the message is supposed to signal the transfer of money between two bank accounts as part of a purchase. The bank may charge a customer twice for a product.

Avoiding multiple deliveries of a message#

To handle scenarios like the one above, we can take multiple approaches to ensure that the nodes only process a message once, even though it may be delivered multiple times. Let's see these approaches.

Idempotent operations approach#

Idempotent is an operation we can apply multiple times without changing the result beyond the initial application.

Example of idempotent operation#

An example of an idempotent operation is to add a value in a set of values. Even if we apply this operation multiple times, the operations that run after the first will have no effect, since the value will already be added to the set. Of course, we assume here that other operations cannot remove values from the set. Otherwise, the retried operation may add a value that was removed.

Example of non-idempotent operation#

An example of a non-idempotent operation is to increase a counter by one, where the

operation will have additional side effects every time it's applied.

By using idempotent operations, we can have the guarantee that even if a node delivers a message multiple times and repeats the operation, the result will be the same.

However, idempotent operations commonly impose tight constraints on the system. So, in many cases, we cannot build our system so that all operations are idempotent by nature. In these cases, we can use another approach: the de-duplication approach.

De-duplication approach#

In the de-duplication approach, we give every message a unique identifier, and every retried message contains the same identifier as the original. In this way, the recipient can remember the set of identifiers it received and executed already. It will also avoid executing operations that are executed.

It is important to note that in order to do this, we must have control on both sides of the system: sender and receiver. This is because the ID generation occurs on the sender side, but the de-duplication process occurs on the receiver side.

Example#

Imagine a scenario where an application sends emails as part of an operation. To send an email is not an idempotent operation. If the email protocol does not support de-duplication on the receiver side, we can't be sure that every email displays exactly once to the recipient.

Difference between delivery and processing#

When we think about **exactly-once semantics**, it's useful to distinguish between the notions of delivery and processing.

In the context of the above discussion, let's consider **delivery** to be the arrival of the message at the destination node, at the hardware level.

Then, we consider **processing** to be the handling of this message from the software application layer of the node.

In most cases, we care more about how many times a node processes a message, than about how many times it delivers it. For instance, in our previous email example, we were mainly interested in whether the application would display the same email twice, and not whether it would receive it twice.

As the previous examples demonstrated, it's impossible to have *exactly-once delivery* in a distributed system. However, it's still sometimes possible to have *exactly-once processing*.

In the end, it's important for us to understand the difference between these two notions, and clarify what we refer to when we talk about exactly-once semantics.

Other delivery semantics#

As a last note, it's easy to see that we can easily implement **at-most-once** delivery semantics and **at-least-once** delivery semantics.

We can achieve the *at-most-once* delivery when we send every message only one time, no matter what happens. Meanwhile, we can achieve the *at-least-once* delivery when we send a message continuously until we get an acknowledgment from the recipient.

Types of Failures

Next

Failure in the World of Distributed Sys…

Mark as Completed

Report an Issue