Log In Join

Back To Module Home

Distributed Systems

0% completed

Search Module

Introduction to Distributed Systems

Mark Module as Completed

Failure in the World of Distributed Systems

Let's see why failures occur in distributed systems, and how we can detect them.

We'll cover the following

- A perfect failure detector

We should understand that it is challenging to identify failure because of all the characteristics of a distributed system that the Difficulties Designing Distributed Systems lesson described. One of them is the asynchronous nature of the network.

One reason for failure#

The asynchronous nature of the network in a distributed system can make it very hard for us to differentiate between a crashed node and a node that is just really slow to respond to requests.

One mechanism to detect failure#

Timeouts is the main mechanism we can use to detect failures in distributed systems. Since an asynchronous network can infinitely delay messages, timeouts impose an artificial upper bound on these delays. As a result, we can assume that a node fails when it is slower than this bound. This is useful because otherwise, the assumption that the nodes are extremely slow would block the system is waiting for the nodes that crashed.

However, a timeout does not represent an actual limit. Thus, it creates the following trade-off.

Trade-off for the small timeout value#

If we select a smaller value for the timeout, our system will waste less time waiting for the nodes that have crashed.

At the same time, the system might declare some nodes that have not crashed dead, while they are actually just a bit slower than expected.

The following illustration shows this trade-off phenomenon.

Created with Fabric.js 3.6.6

1 of 5

Trade-off for the large timeout value#

If we select a larger value for the timeout, the system will be more lenient with slow nodes.

At the same time, the system will be slower in identifying crashed nodes, in some cases wasting time while waiting for them.

The following illustration shows this trade-off phenomenon.

Created with Fabric.js 3.6.6

1 of 7

This is a fundamental problem in the field of distributed systems.

Failure detector#

A failure detector is the component of a node that we can use to identify other nodes that have failed.

This component is essential for various algorithms that need to make progress in the presence of failures. There has been extensive research about failure detectors.

Properties that categorize failure detectors#

We can distinguish the different categories of failure detectors through two basic properties that reflect the trade-off:

1. Completeness

   Completeness corresponds to the percentage of crashed nodes a failure detector successfully identifies in a certain period.

2. Accuracy

   Accuracy corresponds to the number of mistakes a failure detector makes in a certain period.

A perfect failure detector#

A perfect failure detector is the one with the strongest form of *completeness* and *accuracy*. That is, it is one that successfully detects every faulty process without ever assuming a node has crashed before it actually does.

As expected, it is impossible to build a perfect failure detector in purely asynchronous systems. Still, we can even use imperfect failure detectors to solve difficult problems. One such example is the problem of consensus.

The Tale of Exactly-Once Semantics

Next

Stateless and Stateful Systems

Mark as Completed

Report an Issue