Log In Join

Back To Module Home

System Design Essentials

0% completed

Search Module

Introduction

Abstractions

Non-functional System Characteristics

- Availability
- Reliability
- Scalability
- Maintainability

Fault Tolerance

Fault Tolerance

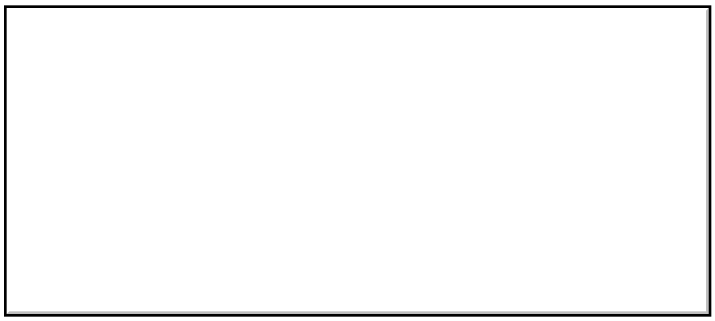
Learn about fault tolerance, how to measure it, and its importance.

We'll cover the following

What is fault tolerance?#

Real-world, large-scale applications run hundreds of servers and databases to accommodate billions of users' requests and store significant data. These applications need a mechanism that helps with data safety and eschews the recalculation of computationally intensive tasks by avoiding a single point of failure.

Fault tolerance refers to a system's ability to execute persistently even if one or more of its components fail. Here, components can be software or hardware. Conceiving a system that is hundred percent fault-tolerant is practically very difficult.



Let’s discuss some important features for which fault-tolerance becomes a necessity.

Availability focuses on receiving every client’s request by being accessible 24/7.

Reliability is concerned with responding by taking specified action on every client’s request.

Fault tolerance techniques#

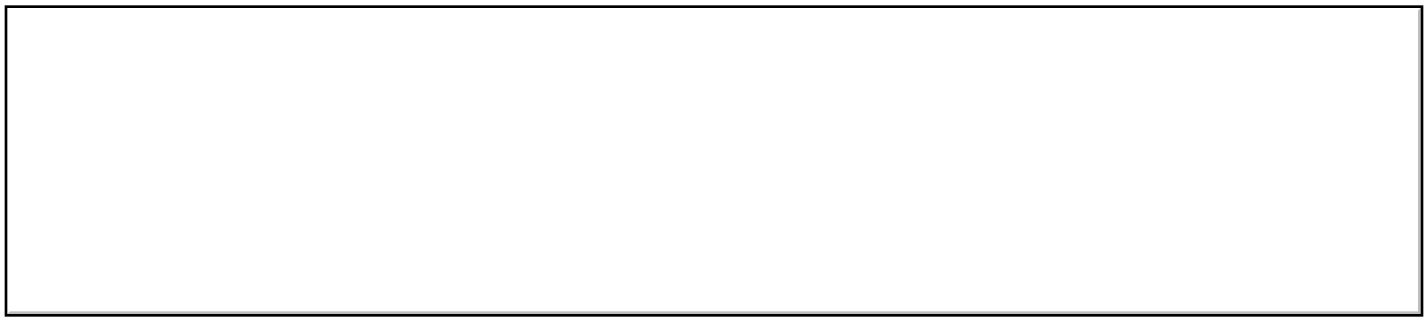
Failure occurs at the hardware or software level, which eventually affects the data. Fault tolerance can be achieved by many approaches, considering the system structure. Let’s discuss the techniques that are significant and suitable for most designs.

Replication#

One of the most widely-used techniques is **replication-based fault tolerance**. With this technique, we can replicate both the services and data. We can swap out failed nodes with healthy ones and a failed data store with its replica. A large service can transparently make the switch without impacting the end customers.

We create multiple copies of our data in separate storage. All copies need to update regularly for consistency when any update occurs in the data. Updating data in replicas is a challenging job. When a system needs strong consistency, we can synchronously update data in replicas. However, this reduces the availability of the system. We can also asynchronously update data in replicas when we can tolerate eventual consistency, resulting in stale reads until all replicas converge. Thus, there is a trade-off between both consistency approaches. We compromise either on availability or on consistency under

failures—a reality that is outlined in the CAP theorem.

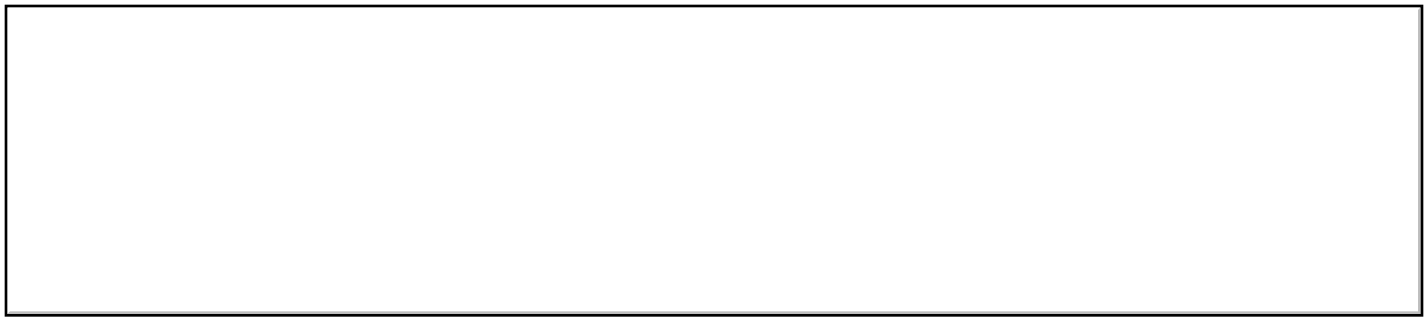


Replication-based fault tolerance

Checkpointing#

Checkpointing is a technique that saves the system's state in stable storage when the system state is consistent. Checkpointing is performed in many stages at different time intervals. The primary purpose is to save the computational state at a given point. When a failure occurs in the system, we can get the last computed data from the previous checkpoint and start working from there.

Checkpointing also comes with the same problem that we have in replication. When the system has to perform checkpointing, it makes sure that the system is in a consistent state, meaning that all processes are stopped. This type of checkpointing is known as **synchronous checkpointing**. On the other hand, checkpointing in an inconsistent state leads to data inconsistency problems. Let's look at the illustration below to understand the difference between a consistent and an inconsistent state:



Checkpointing in a consistent and inconsistent state

Consistent state: The illustration above shows no communication among the processes

when the system performs checkpointing. All the processes are sending or receiving messages before and after checkpointing. This state of the system is called a consistent state.

Inconsistent state: The illustration also displays that processes communicate through messages when the system performs checkpointing. This indicates an inconsistent state, because when we get a previously saved checkpoint, *Process i* will have a message (m_1) and *Process j* will have no record of message sending.

Maintainability

Next

Put Back-of-the-envelope Numbers in…

Mark as Completed

Report an Issue