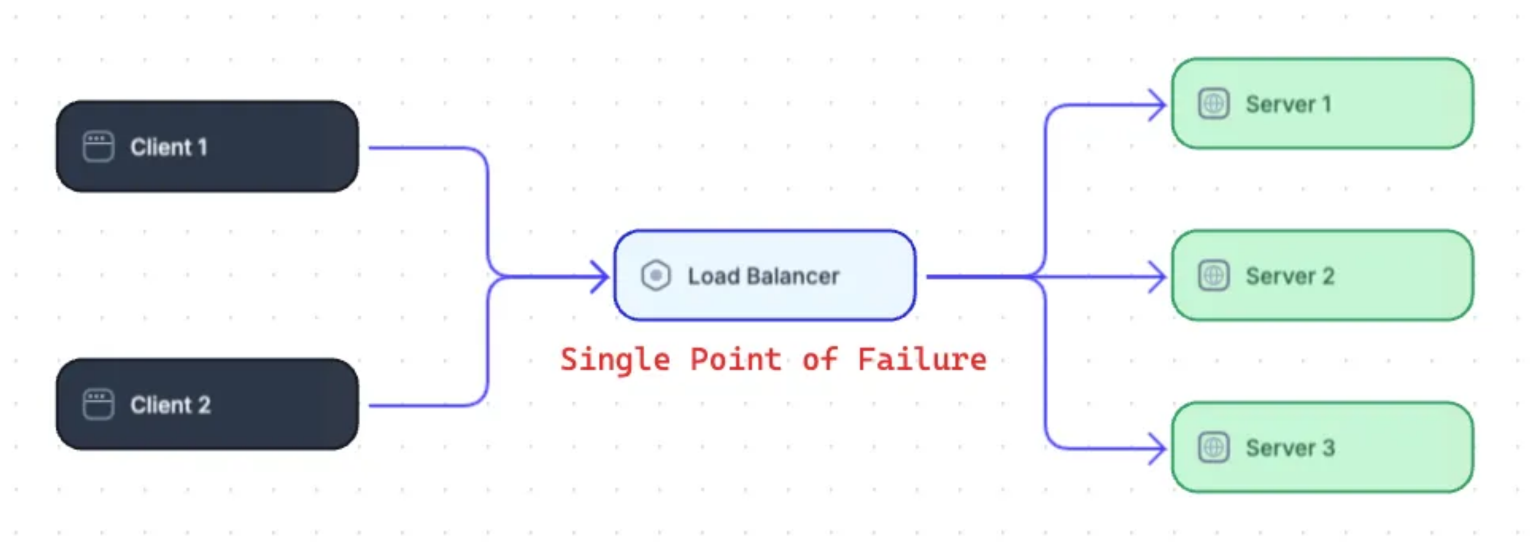
System Design: How to Avoid Single Point of Failures?

ASHISH PRATAP SINGH
OCT 09, 2024

178 15 16 Share

A **Single Point of Failure** (**SPOF**) is a component in your system whose failure can bring down the entire system, causing downtime, potential data loss, and unhappy users.



Created using Multiplayer

In the above example, if there is only one instance of the **load balancer**, it becomes a SPOF. If it goes down, clients won’t be able to communicate with the servers.

By minimizing the number of SPOFs, you can improve the overall **reliability** and **availability** of the system.

In this article, we'll explore what a SPOF is, how to identify it in a distributed system, and strategies to avoid it.

If you’re enjoying this newsletter and want to get even more value, consider becoming a paid subscriber.

As a paid subscriber, you'll unlock all **premium articles** and gain full access to all premium courses on algomaster.io.

---

1. Understanding SPOFs

A Single Point of Failure (SPOF) is any component within a system whose failure would cause the entire system to stop functioning.

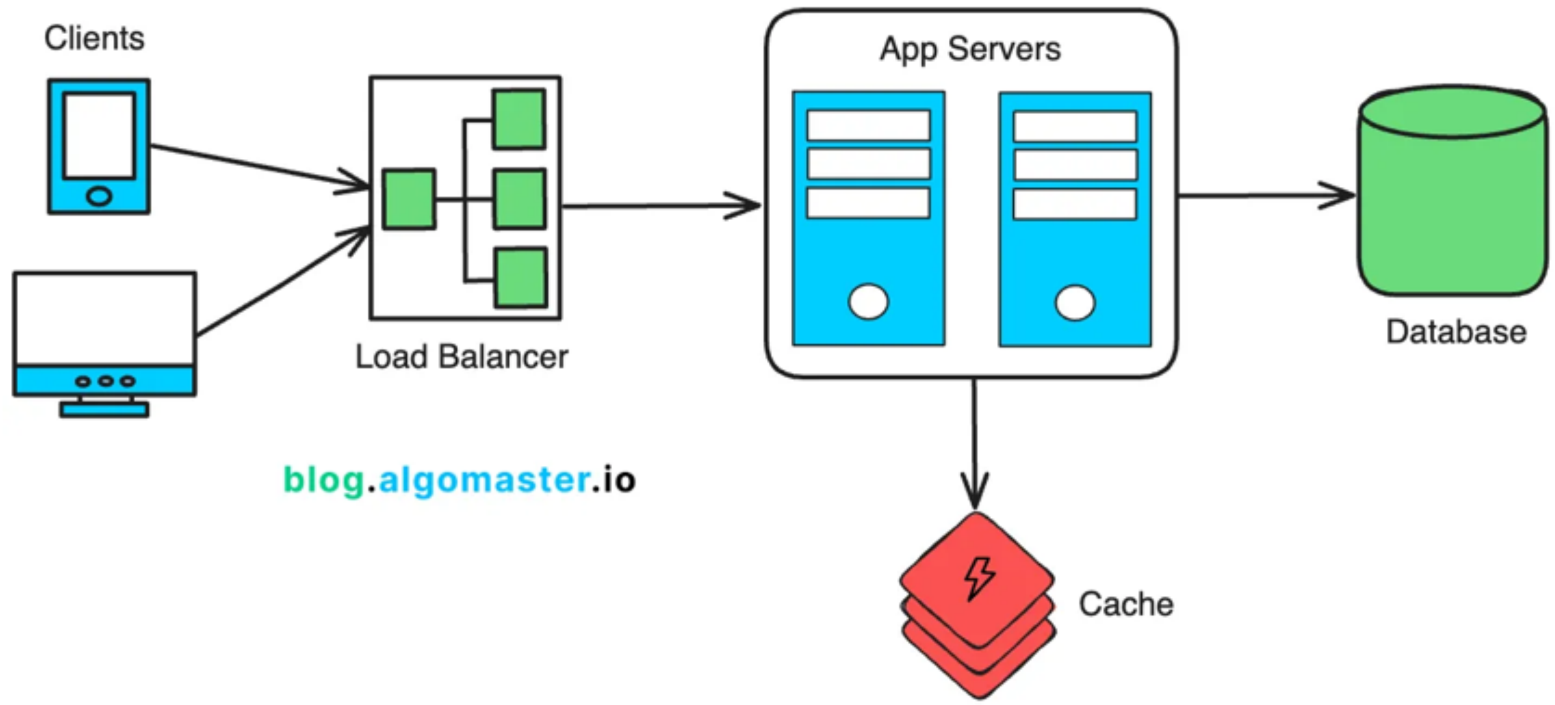
> Imagine a bridge that connects two cities. If it's the only route between them and it collapses, the cities are cut off. In this scenario, the bridge is the single point of failure.

In distributed systems, failures are inevitable. Common causes include **hardware malfunctions**, **software bugs**, **power outages**, **network disruptions**, and **human error**.

While failures can't be entirely avoided, the goal is to ensure they don’t bring down the entire system.

In system design, SPOFs can include a single server, network link, database, or any component that lacks redundancy or backup.

Let’s see an example of a system and various single points of failures in it:



Created using Multiplayer

This system has one load balancer, two application servers, one database, and one cache server.

Clients send requests to the load balancer, which distributes traffic across the two application servers. The application servers retrieve data from the cache if it's available, or from the database if it's not.

In this design, the potential SPOFs are:

- **Load Balancer**: If there is only one load balancer instance and it fails, all traffic will stop, preventing clients from reaching the application servers. To avoid this, we can add a **standby** load balancer that can takeover if the primary one fails.
- **Database**: With only one database, its failure would result in data being unavailable, causing downtime and potential data loss. We can avoid this by **replicating** the data across multiple servers and locations.
- **Cache Server**: The cache server is not a true SPOF in the sense that it doesn’t bring the entire system down. When it’s down, every request hits the database, increasing its load and slowing response times.

The **application servers** are not SPOFs since you have two of them. If one fails, the other can still handle requests, assuming the load balancer can distribute traffic effectively.

---

2. How to Identify SPOFs in a Distributed System

1. Map Out the Architecture

Create a detailed diagram of your system's architecture. Identify all components, services, and their dependencies.

Look for components that do not have backups or redundancy.

2. Dependency Analysis

Analyze dependencies between different services and components.

If a single component is required by multiple services and does not have a backup, it is likely a SPOF.

3. Failure Impact Assessment

Assess the impact of failure for each component.

Perform a "**what if**" analysis for each component.

Ask questions like, “What if this component fails?” If the answer is that the system would stop functioning or degrade significantly, then that component is a SPOF.

4. Chaos Testing

Chaos testing, also known as Chaos Engineering, is the practice of intentionally injecting failures and disruptions into a system to understand how it behaves under stress and to ensure it can recover gracefully.

Chaos engineering often involves the use of tools like Chaos Monkey (developed by Netflix) that randomly shut down instances or services to observe how the rest of the system responds.

This can help us identify components that, if they fail, would cause a significant impact on the system.

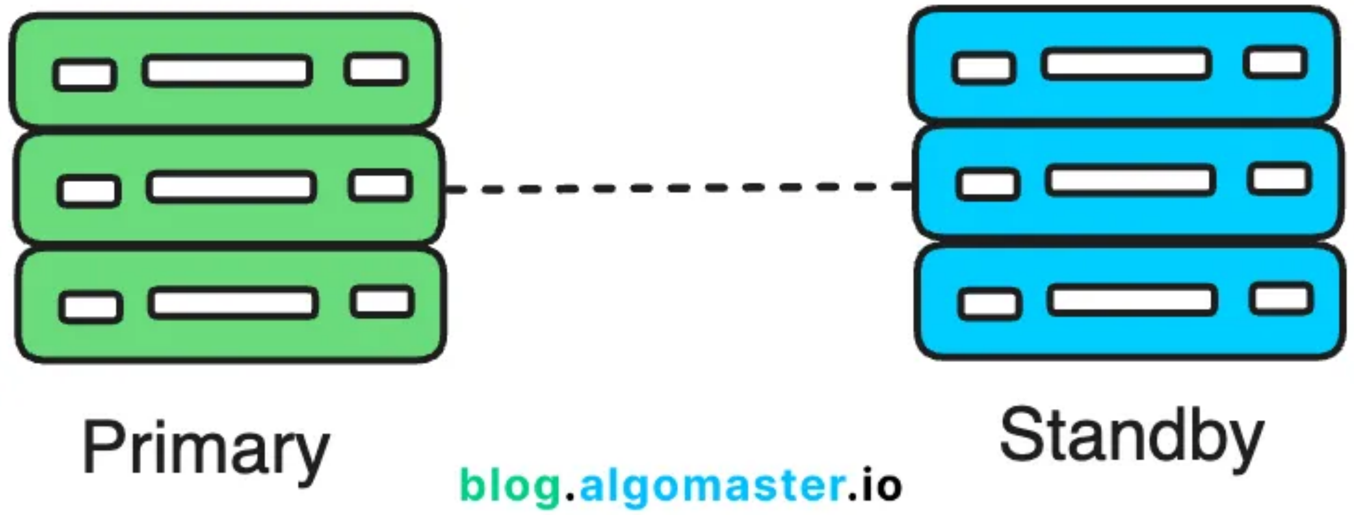
---

3. Strategies to Avoid Single Points of Failures

1. Redundancy

The most common way to avoid SPOFs is by adding **redundancy**. Redundancy means having multiple components that can take over if one fails.

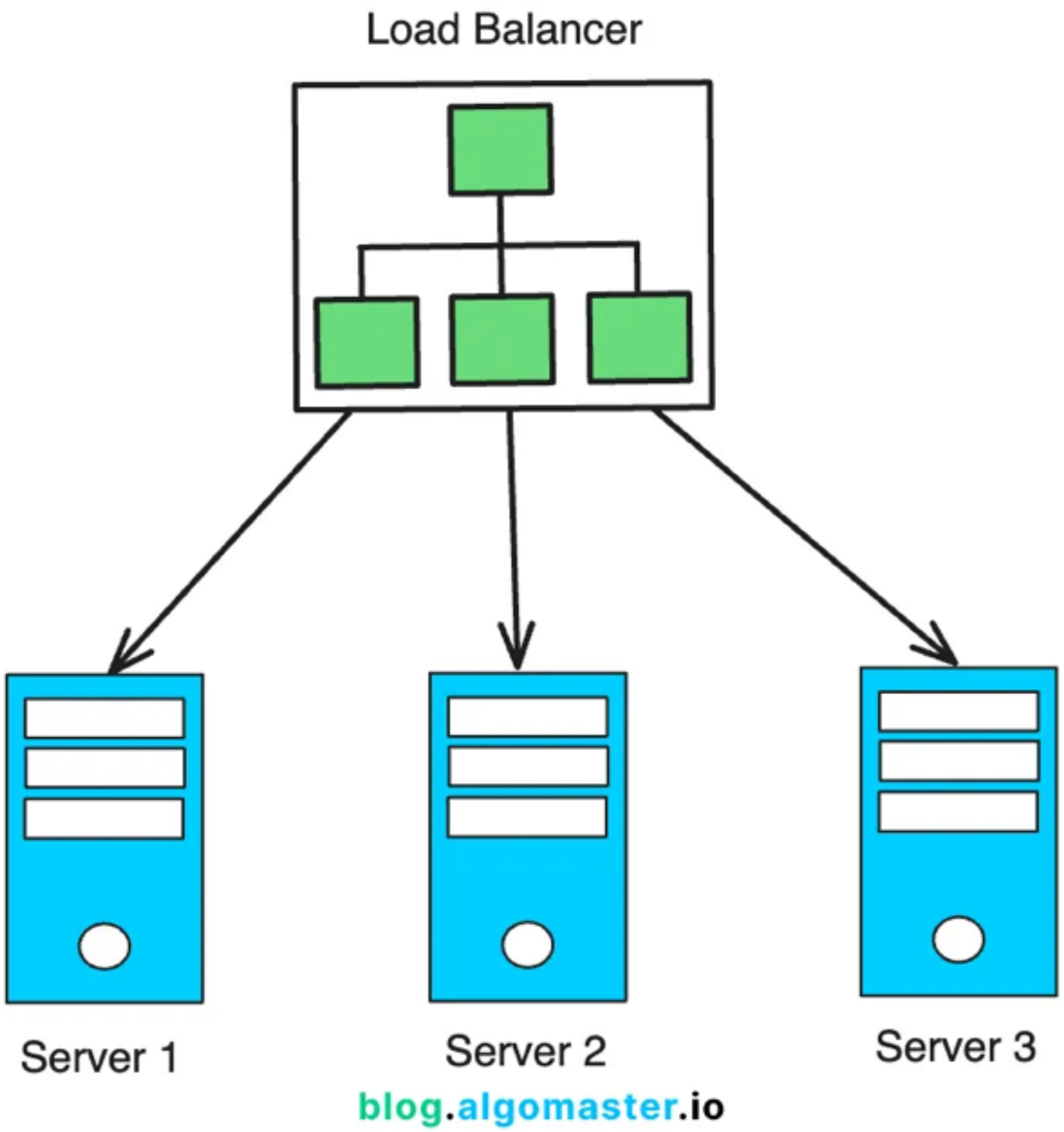
Redundant components can be either **active** or **passive**. Active components are always running. Passive (standby) components are only used as a backup when the active component fails.



Created using Multiplayer

2. Load Balancing

Load balancers distribute incoming traffic across multiple servers, ensuring no single server becomes overwhelmed.



Created using Multiplayer

They help avoid single point of failures by detecting failed servers and rerouting traffic to healthy instances.

3. Data Replication

Data replication involves copying data from one location to another to ensure that data is available even if one location fails.

- **Synchronous Replication**: Data is replicated in real-time to ensure consistency across locations.
- **Asynchronous Replication**: Data is replicated with a delay, which can be more efficient but may result in slight data inconsistencies.

4. Geographic Distribution

Distributing services and data across multiple geographic locations mitigates the risk of regional failures.

This includes using:

- **Content Delivery Networks (CDNs)** to distribute content globally, improving availability and reducing latency.

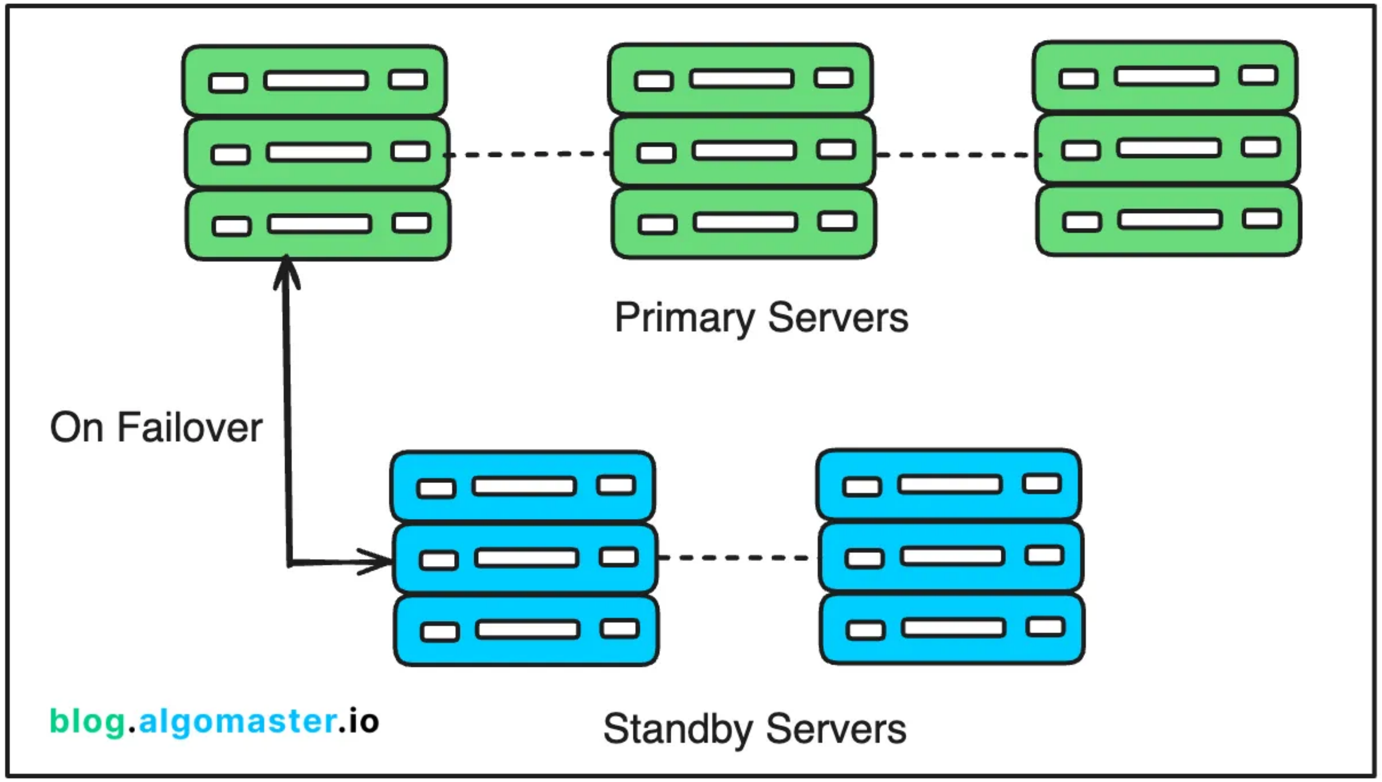
- **Multi-Region Cloud Deployments** to ensure that an outage in one region does not disrupt your entire application.

5. Graceful Handling of Failures

Design applications to handle failures without crashing.

> **Example:** If a service that provides user recommendations fails, the application should still function, perhaps with a message indicating limited features temporarily.

Implement failover mechanisms to automatically switch to backup systems when failures are detected.



Sketched using Multiplayer

6. Monitoring and Alerting

Proactive monitoring helps detect failures before they lead to major outages.

Key practices include:

- **Health Checks:** Automated tools that perform regular health checks on components.

- **Automated Alerts**: Alerts and notifications sent when a component fails or behaves abnormally.
- **Self-Healing Systems**: Systems that automatically recover from failures, such as auto-scaling to replace failed servers.

As a paid subscriber, you'll unlock all **premium articles** and gain full access to all premium courses on algomaster.io.

There are group discounts, gift options, and referral bonuses available.

Checkout my Youtube channel for more in-depth content.

Follow me on LinkedIn, X and Medium to stay updated.

Checkout my GitHub repositories for free interview preparation resources.

I hope you have a lovely day!

See you soon,
Ashish

---

178 Likes · 16 Restacks

← Previous

Next →

Discussion about this post

Discussion about this post

Comments Restacks

Write a comment...

Luv Singh 10 Oct 2024

Liked by Ashish Pratap Singh

Thanks ashish loved reading it 🤙🏻

LIKE (1) REPLY SHARE

Ranjeet Nair 9 Oct 2024

Liked by Ashish Pratap Singh

Well written article .Thanks for sharing

LIKE (1) REPLY SHARE

1 reply by Ashish Pratap Singh

13 more comments...