# Fundamentals of Distributed Systems

Understanding the concepts and characteristics of distributed systems is essential for designing fault-tolerant, highly scalable, and low-latency services.

## What is a Distributed System?

A distributed system is a network of independent software components or machines that function as a unified system for the end user. In this setup, several computers (known as nodes) communicate with one another and share their resources to complete tasks that may be too complex or time-consuming for a single machine to handle.

On other side, due to the decentralized nature of distributed systems, they can handle a high volume of requests and serve millions of users simultaneously. They operate in parallel, which means that even if one component fails, it does not impact the performance of the entire system.

For example, traditional databases stored on a single machine may experience difficulties in performing read and write operations as the amount of data grows. One way to resolve this issue is to split the database system across multiple machines. If the volume of read traffic is much higher than that of write traffic, we can also implement master-slave replication, where read and write requests are processed on separate machines.

## Scalability in a distributed system

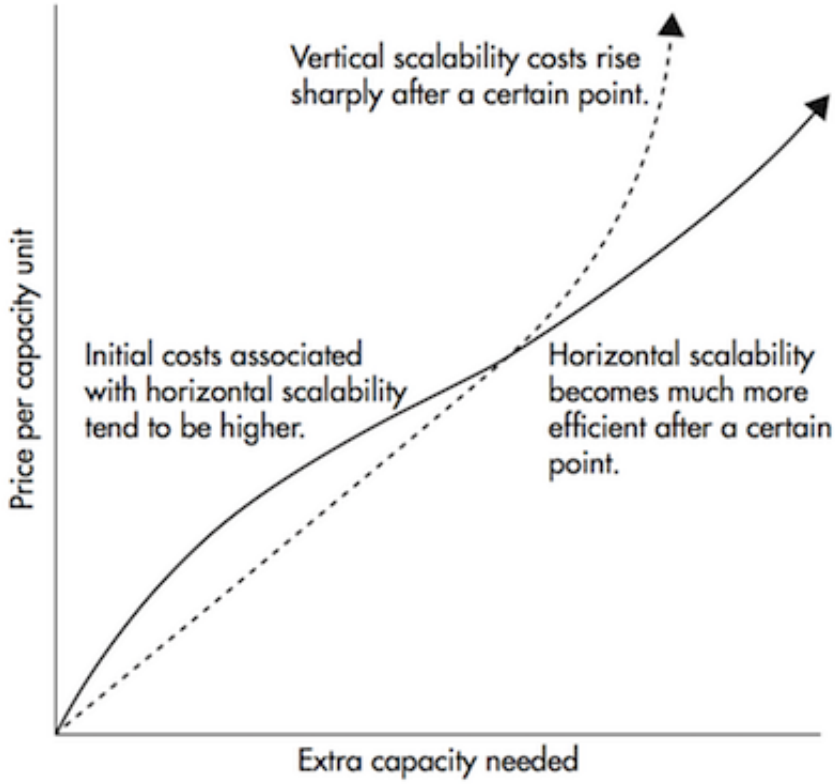
Distributed systems can continuously evolve to support growing workloads like handling a large number of user requests or a large number of database transactions. So one of the key characteristics of a distributed system is to achieve high scalability without decrease in

performance.

Traditionally, systems scale using **vertical scaling**, which involves adding more CPU, RAM, and storage to an existing server. This approach is not suitable for large-scale operations because it is expensive and prone to a single point of failure. The idea is simple: It is limited by the capacity of a single server, and scaling beyond that capacity often requires downtime.

On the other hand, **horizontal scaling** allows for indefinite scaling by adding more servers to the pool of resources. If there is any performance degradation due to large number of requests, we can simply add more machines. This will make the system fast with minimal overhead cost compared to vertical scaling.

So horizontal scaling is a common approach for distributed systems because it allows for flexibility and efficiency in handling growing workloads. The figure below describes how much a company costs to use Vertical vs Horizontal Scaling.




Good examples of horizontally scalable systems are Cassandra and MongoDB. MySQL is an example of a vertically scalable system because it

allows for easy scaling by switching to larger machines. But this process often requires downtime.

## Performance of a distributed system

There are two standard parameters to measure the performance of a distributed system:

1. Latency or response time: Delay in obtaining the response to the first request.
2. Throughput: Number of requests served in a given time.

These two factors are related to the volume of responses sent by nodes and the size of responses. It's important to note that the performance of distributed systems also depends on other factors like network load, architecture of software and hardware components, etc.

Many highly scalable services are read-heavy, which can decrease system performance. To address this issue, one solution is to use replication, which ensures high availability and fault tolerance. To further increase performance, distributed systems can use sharding by dividing the central database server into smaller servers called shards. This will improve performance by distributing the load across multiple servers.

### Scalability vs Performance

Scalability and performance are related, but they are not the same thing. Performance measures how quickly a system can process a request or perform a specific task, while scalability measures how much a system can grow or shrink.

For example, consider a system with 200 concurrent users, each sending a request every 5 seconds (on average). In this situation, the system would need to handle a throughput of 40 requests per second.

The performance of the system would determine how much time it takes to serve these 40 requests per second, while the scalability of the system

would determine how many additional users the system can handle and how many more requests it can serve without degrading the user experience.

## Reliability of a distributed system

Reliability is one of the main characteristics of any distributed system. It refers to the probability that a system will fail within a given time period. A distributed system is reliable if it can continue delivering its services even when one or more components fail. In such systems, another healthy machine can always replace a failing machine.

For example, an e-commerce store needs to ensure that user transactions are never cancelled due to a machine failure. If a user adds an item to their shopping cart, the system must not lose that data. Reliability can be achieved through redundancy i.e. if the server hosting the user's shopping cart fails, another server with an exact replica of the shopping cart should take its place. So redundancy comes at a cost, and a reliable system must be prepared to pay for this resilience by eliminating all single points of failure.

### How important Is reliability?

Bugs or outages of critical applications can lead to lost productivity and significant costs in terms of lost revenue. Even in noncritical applications, businesses have a responsibility towards their users.

Suppose a customer stored all their important data in an application. After some time, that database suddenly got corrupted and there is no mechanism to restore it from a backup. What would happen? This would likely result in a loss of trust from the customer. So the conclusion is simple: It is important for businesses to ensure the reliability and availability of their applications to avoid these types of issues and maintain customer satisfaction.

## Availability of a distributed system

Availability refers to the percentage of time that a system or service is operational under normal conditions. It is a measure of how often a system is available for use. In rough words, an application that can continue to function for several months without experiencing downtime can be considered highly available.

For example, in a company like Amazon, internal services are described in terms of the 99.9th percentile for availability. This means that 99.9% of requests are expected to be served without issue. While this only affects 1 in 1,000 requests, it is important to prioritize these requests because they are likely from the company's most valuable customers, who have made many purchases and have a lot of data on their accounts. So ensuring high availability is important for maintaining customer satisfaction and trust.

**Reliability vs Availability**

If a system is reliable, it is available. However, if it is available, it is not necessarily reliable. In other words, high reliability contributes to high availability. Still, achieving high availability even with an unreliable system is possible by minimising maintenance time and ensuring that machines are always available when needed.

For example, suppose a system has 99.99% availability for the first two years after its launch. Unfortunately, the system was launched without any security testing. Customers are happy, but they never realize that system is vulnerable to security risks. In the third year, the system experiences a series of security problems that suddenly result in low availability for long periods. This may result in financial loss to the customers.

## Other key characteristics of a distributed system

- **Manageability:** How it is easy to operate and maintain the system. If it takes a lot of time to fix system failures, availability will decrease. So early detection of faults can reduce or prevent system downtime.
- **Concurrency:** Ability of multiple components to access and update

shared resources concurrently without interference. Concurrency reduces latency and increases the throughput of distributed systems.

- **Transparency:** Allows users to view a distributed system as a single, logical device, without needing to be aware of the system architecture. This is an abstraction where a distributed system, consisting of millions of components spread across multiple computers, appears as a single system to the end user.
- **Openness:** Ability to update and scale a distributed system independently. This is about how easy it is to integrate new components or replace existing ones without affecting the overall computing environment.
- **Security:** It is crucial because users often send requests to access sensitive data managed by servers. For example, doctors may request records from hospitals, and users may purchase items through an e-commerce website. To prevent denial of service attacks and ensure security and privacy, distributed systems must use secure authentication processes to identify users.
- **Heterogeneity:** Distributed system components may have a variety of differences in terms of networks, hardware, operating systems, programming languages, and implementations by different developers.

## Types of Distributed Systems

One way to classify distributed systems is based on their architecture, which describes the way the components are organized and interact with each other. Some of the common architectures are:

**Client-Server Architecture:** This is the most traditional and widely used architecture. In this model, clients send requests to a central server, which processes and returns the requested information. An example of this architecture is an email system where client is an email software (Gmail) and the server is Google Mail Server.

Peer-to-Peer (P2P) Architecture: In P2P, there are no centralized

machines. Each component acts as an independent server and carries out its assigned tasks. Responsibilities are shared among multiple servers called peers, who collaborate to achieve a common goal. An example of this architecture is file-sharing networks like BitTorrent, where every user acts as both a client and a server. Users can upload and download files directly from each other's computers, without the need for a central server.

## Advantages of Distributed Systems

The key benefits of using distributed systems are:

- **Reliability:** They remain available most of the time, irrespective of the failure of any particular system server. If one server fails, the service remains operational.
- **Scalability:** Using horizontal scalability, we can add a large number of servers independently.
- **Low latency:** We can use replication and place servers close to the user location to reduce the latency.
- **Cost-effective:** Compared to a single-machine system, the distributed system is made up of several machines together. Although such systems have high implementation costs, they are far more cost-effective when working on a large scale.
- **Efficiency:** Distributed systems are made efficient in every aspect since they possess multiple machines. Each of these computers could work independently to solve problems.

## Disadvantages of Distributed Systems

It is essential to know the various challenges that one may encounter while using any system. This will help in dealing with trade-offs. The shortcomings of distributed systems are:

- **Complexity:** Distributed systems are highly complex. Although using a large number of machines, the system can become scalable, but it increases the system’s complexity. There will be more messages,

network calls, devices, user requests, etc.

- **Network failure:** Distributed systems have heavily relied on network calls for communications and transferring information or data. In case of network failure, message mismatch or incorrect ordering of segments leads to communication failure and eventually deteriorates its application's overall performance.
- **Consistency:** Because of its highly complex nature, it becomes too challenging to synchronise the application states and manage the data integrity in the service.
- **Management:** Many functions, such as load balancing, monitoring, increased intelligence, logging, etc., need to be added to prevent the distributed systems' failures.

## Conclusion

Distributed systems are a necessity of the modern world as new machines need to be added, and applications need to scale to deal with technological advancements and better services. It enables modern systems to offer highly scalable, reliable, and fast services.

If you have any queries or feedback, please write us at contact@enjoyalgorithms.com. Enjoy learning, Enjoy system design!