Fundamentals of Distributed Systems

Understanding 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 that are 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 implement master-slave replication, where read and write requests are processed on separate machines.

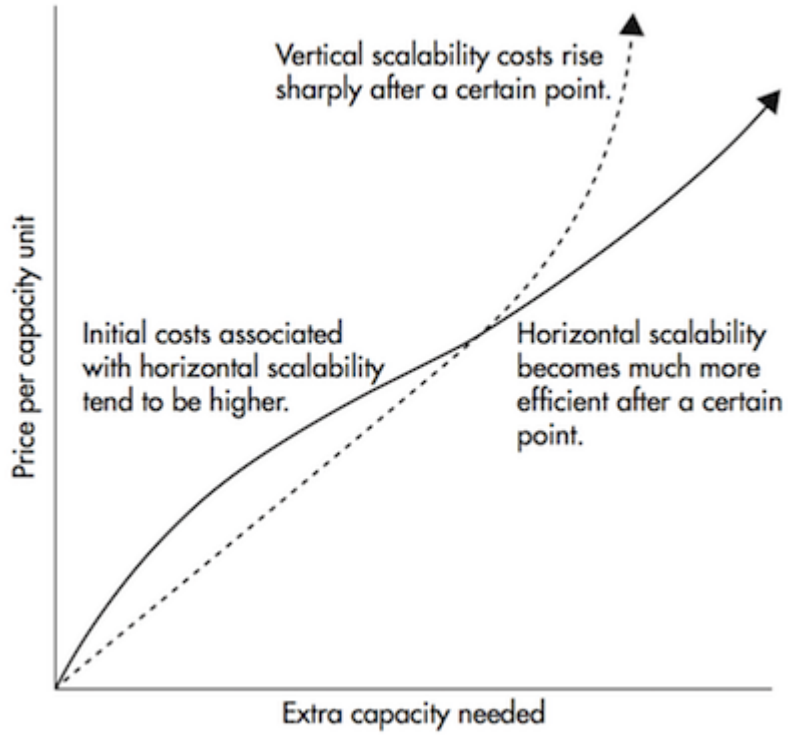
Scalability in a distributed system

One of the main benefits of using distributed systems is their ability to provide highly scalable services. These systems can continuously evolve to support growing workloads, such as handling a large number of user requests or a large number of database transactions. The goal of a scalable distributed system is to achieve high scalability without a decrease in performance.

Traditionally, systems scale using **vertical scaling**, which involves adding more power (such as CPU, RAM, and storage) to an existing server. However, this approach is not suitable for large-scale operations because it is expensive and prone to a single point of failure. It is also 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, we can simply add more machines, making the system extremely fast with minimal overhead cost compared to vertical scaling. 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 include Cassandra and MongoDB, as they both provide an easy way to scale horizontally by adding more machines to meet growing needs. MySQL is an example of a vertically scalable system, as it allows for easy scaling by switching to larger machines. However, 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: This refers to the delay in obtaining the response to the first request.
2. Throughput: This refers to the 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, which represent the volume of data exchanges. It's important to note that the performance of distributed systems also depends on other factors such as network load, the 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. However, there is a limit to this approach. To further increase performance, distributed systems can use sharding to scale the service by dividing the central database server into smaller servers called shards. This allows for a higher level of 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 100 concurrent users, each sending a request every 5 seconds (on average). In this situation, the system would need to handle a throughput of 20 requests per second. The performance of the system would determine how much time it takes to serve these 20 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 considered 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 information. Reliability can be achieved through redundancy, meaning that if the server hosting the user's shopping cart fails, another server with an exact replica of the shopping cart should take its place. However, 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 have significant costs in terms of lost revenue and damage to reputation. Even in noncritical applications, businesses have a responsibility to their users. Imagine if a customer stored all their important information in an application and that database was suddenly corrupted, with no mechanism to restore it from a backup. This would likely result in loss of trust from the customer. 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. For example, an aircraft that can fly for many hours per month without experiencing significant downtime can be considered to have high availability.

In the case of 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 therefore have a lot of data on

their accounts. 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. The 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 features of a distributed system

In designing a distributed system, **manageability** is an important factor to consider. This refers to how easy it is to operate and maintain the system. If the time it takes to fix system failures is long, availability will decrease. In other words, early detection of faults can reduce or prevent system downtime.

Concurrency is another important aspect of distributed systems. This refers to the ability of multiple components to access and update shared resources concurrently without interference. Concurrency helps to reduce latency and increase the throughput of the distributed system.

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 refers to the 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 is crucial in a distributed system, as users 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 is another characteristic of distributed systems, where 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. The 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: Distributed systems remain available most of the time, irrespective of the failure of any particular system server. If one server fails, the service remains operational.

Scalability: Distributed systems offer extensive scalability. Adding a large number of servers allows such systems to achieve horizontal scalability.

Low latency: Distributed systems offer high-speed service because of the replication of servers and servers' location close to users, reducing the query time.

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. This is not only considered efficient, but it significantly saves the user time.

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 the 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/doubts/feedback, please write us at contact@enjoyalgorithms.com. Enjoy learning, Enjoy system design, Enjoy algorithms!

Author: Shubham Gautam

Reviewer: Keshav Nandan

Share:

Related Tags: system-design-concepts system-design-interview

Share Feedback

Name

Email

Message

Submit Your Response

Coding Interview

DSA Course

DSA Blogs

Machine Learning

ML Course

ML Blogs

System Design

SD Course

SD Blogs

OOP Concepts

OOP Course

OOP Blogs

EnjoyAlgorithms Newsletter

Subscribe to get well designed content on data structure and algorithms, machine learning, system design, object orientd programming and math.

Email address

Subscribe

Explore More Content

Notification Service System Design

Notification services are widely used in almost every product. They are helpful if we want to be alerted of a price change or availability of a new product feature or if we want to be updated if a new job specification for a job search becomes available. This blog will focus on system design of notification service and discussion around various components.

Read More

Publisher Subscriber (Pub-Sub) Design Pattern

The publish subscribe pattern, sometimes known as pub sub pattern, is an architectural design pattern that enables publishers and subscribers to communicate with one another. This pattern rely on a message broker to send messages from publisher to subscribers. Messages are sent out by the publisher to a channel that subscribers can join.

Read More

CAP Theorem in System Design

CAP Theorem is an essential concept in system design for designing networked shared data systems. It states that a distributed database system can only provide two of these three properties : consistency, availability, and partition tolerance. We can make trade-offs between three available properties based on use cases for our dbms system.

Read More

Data Partitioning (Sharding) in System Design

Data partitioning or sharding is a technique of dividing data into independent components. It is a way of splitting data into smaller pieces so that data can be efficiently accessed and managed. Database partitioning is the backbone of modern system design, which helps to improve scalability, manageability, and availability.

Read More

WebSockets in System Design

Web Socket is a communications protocol which provides full-duplex communication channels over a single TCP connection. This is also a vital component behind multiplayer games or applications that rely on real-time data transfer. In this blog, we have explained: 1) What is WebSocket in system design? 2) How does it work?

Read More

Consistent Hashing in System Design

Consistent Hashing is the most widely used concept in system design, as it offers considerable flexibility in scaling the application. This blog discusses the key concepts and approaches which come in handy while scaling out the distributed system. Consistent Hashing is frequently applied to solving various system-related challenges.

Read More

Design Twitter: System Design Interview Question

Twitter is a social media platform where users can post and interact with tweets. Users can also subscribe to feeds of other users and receive tweet notifications from those they follow. Tweets are almost 140–280 characters communication. This blog will focus on system design of Twitter and discussion around various components.

Read More

API Rate Limiter System Design

A rate limiter restricts the number of events that a certain user or device can perform within a given time range. It helps us to limit the number of requests a sender can send in a particular period of time. Once the upper limit is reached, the rate limiter blocks further requests. In this blog, we will discuss components and algorithms related to design rate limiter.

Read More

Throughput in System Design

The throughput of a system is defined as the total units of information a system can process in a given amount of time. It is generally represented as the number of bits transmitted per second or HTTP operations per day. This blog will discuss the important factors that affect throughput in designing distributed systems.

Read More

Typeahead (Autocomplete) System Design

As users type their search query, the Typeahead search feature guesses the rest of a word and offers the top suggestions that begin with whatever they have written. Here frequency and recentness of a query are used to sort suggestions. Search autocomplete system is used by many search platforms like Facebook, Google, Instagram, etc.

Read More

Network Protocols Concept

In computer networking, a network is a group of devices connected in some way to exchange data. Similarly, network protocols define a common format and set of rules for exchanging messages between devices. Some common networking protocols are Hypertext Transfer Protocol (HTTP), Transmission Control Protocol (TCP), and Internet Protocol (IP).

Read More

Design Dropbox

Dropbox is a cloud storage service that allows users to store their data on remote servers. The remote servers store files durably and securely, which can be accessed from anywhere using the internet. These servers are maintained by cloud storage providers. In this blog, we will focus on system design of dropbox and discussion around various components.

Read More

Follow us on