Distributed Systems

3rd Semester Software Technology Engineering

Eduard Fekete

November 10, 2025

Contents

1 Distributed Systems

A distributed system is a collection of independent computers that appear to the users of the system as a single computer. The computers in a distributed system communicate and coordinate their actions by passing messages to one another over a network.

1.1 Architectural Qualities

1.1.1 Heterogeneity

Heterogeneity refers to the ability of a distributed system to operate across different types of hardware, operating systems, programming languages and network protocols. This allows for greater flexibility and adaptability in the system's design and implementation.

Heterogeneity should be avoided when possible, as it can introduce unnecessary complexity.

1.1.2 Security

Security in distributed systems involves protecting data and resources from unauthorized access, ensuring data integrity, and maintaining confidentiality. This is often achieved through encryption, authentication mechanisms, and secure communication protocols.

1.1.3 Scalability

Scalability refers to the ability of a distributed system to handle increasing amounts of work or to be easily expanded to accommodate growth.

Horizontal Scaling: Adding more machines to the system to distribute the load. **Vertical Scaling**: Adding more resources (CPU, RAM) to existing machines to improve performance.

1.1.4 Fault Tolerance

Fault tolerance is the ability of a distributed system to continue functioning correctly even in the presence of faults or failures, such as hardware malfunctions or network issues.

Usually achieved through redundancy, replication, and failover mechanisms.)

1.1.5 Concurrency

Concurrency is the ability of a distributed system to handle multiple operations or transactions simultaneously, improving efficiency and performance. One of the main problems is handling access to shared resources without conflicts.

1.1.6 Transparency

Transparency in distributed systems means hiding the complexity of the distributed nature from the users, making the system appear as a single, unified entity.

1.1.7 Quality of Service

Quality of Service (QoS) involves ensuring that the distributed system meets certain performance standards.

1.1.7.1 Reliability Reliability refers to the ability of a distributed system to consistently perform its intended functions without failure over a specified period of time. It is tightly related to fault tolerance and availability.

1.1.7.2 Performance Performance in distributed systems is often measured in terms of response time, throughput, and resource utilization. Optimizing performance involves minimizing latency, maximizing bandwidth, and efficiently managing resources across the distributed components.

Optimizing involves most importantly empirical testing and monitoring.

KIWI means "Kill It With Iron" and consists of scaling vertically to improve performance.

1.1.7.3 Maintainability Maintainability is the ease with which a distributed system can be modified, updated, or repaired.

Important concepts include high cohesion and low coupling.

1.1.7.4 Consistency Consistency ensures that all nodes in a distributed system have the same data at any given time, preventing discrepancies and ensuring data integrity.

1.1.7.5 Availability Availability refers to the ability of a distributed system to remain operational and accessible to users, even in the face of failures or high demand.

Often achieved through redundancy, load balancing, and failover mechanisms. Proximity to users also plays a role.

High availability is incompatible with strong consistency.

1.2 System Models

1.2.1 Physical Model

The physical model of a distributed system describes the actual hardware and network infrastructure that supports the system. This includes the physical locations of servers, data centers, and network connections.

1.2.2 Fundamental Model

The fundamental model of a distributed system describes the basic principles and concepts that underlie the system's design and operation. This includes concepts such as processes, communication, synchronization, and fault tolerance.

1.2.3 Architectural Model

Describes the high-level structure and organization of the distributed system, including the components, their interactions, and the communication protocols used.

Consists of:

- entities
- communication paradigms
- roles and responsibilities
- placement (physical mapping)

1.3 Entities

Entities are processes, nodes, or threads that perform specific functions within the distributed system. They can be objects, components, or services that interact with each other to achieve the system's goals.

1.4 Communication Paradigms

Communication paradigms define how entities in a distributed system communicate and interact with each other. Common communication paradigms include:

- Inter-process Communication
- Remote Invocation
- Indirect Communication (Group Communication, Message Queues)

1.4.1 Synchronous Communication

In synchronous communication, the sender waits for a response from the receiver before proceeding. This approach is often used in scenarios where immediate feedback is required, such as in remote procedure calls (RPCs) or client-server interactions.

1.4.2 Asynchronous Communication

In asynchronous communication, the sender does not wait for a response from the receiver and can continue processing other tasks. This approach is often used in scenarios where high throughput and responsiveness are required, such as in message queues or event-driven architectures.

1.5 Architectural Styles

1.5.1 Client-Server

Client-server architecture is a distributed system model where clients request services and resources from centralized servers. The server processes these requests and returns the appropriate responses to the clients.

1.5.2 Peer-to-Peer (P2P)

Peer-to-peer architecture is a distributed system model where each node (peer) in the network can act as both a client and a server. Peers communicate directly with each other to share resources and services without relying on a centralized server.

1.5.3 N-Tier

N-tier architecture is a distributed system model that separates the system into multiple layers or tiers, each responsible for specific functions. Common tiers include presentation, application, and data storage.

1.5.4 Microservices

Microservices architecture is a distributed system model where the application is composed of small, independent services that communicate with each other through well-defined APIs. Each microservice is responsible for a specific business function and can be developed, deployed, and scaled independently.

1.5.5 Service-Oriented Architecture (SOA)

Service-oriented architecture (SOA) is a distributed system model that organizes the system into a collection of services that communicate with each other over a network. Each service is a self-contained unit that performs a specific function and can be reused across different applications.

1.5.6 MVC

Model-View-Controller (MVC) is a software architectural pattern that separates an application into three main components: the Model, which represents the data and business logic; the View, which handles the presentation and user interface; and the Controller, which manages user input and interactions between the Model and View.

1.6 Architectural Patterns

1.6.1 Layering

Layering is an architectural pattern that organizes a distributed system into distinct layers, each responsible for specific functions and services. Each layer communicates with the layers directly above and below it, promoting modularity and separation of concerns.

2 Threats and Threat Modeling

$Risk = Frequency \times Impact$

Risk is a measure of the potential harm that a threat can cause to a distributed system. Based on risk, appropriate countermeasures should be implemented to mitigate the identified threats.

Based on the formula above, risk can be reduced by either lowering the frequency of a threat occurring or by reducing the impact of the threat if it does occur.

Most important ways of dealing with threats:

- **Prevention**: Implementing measures to reduce the likelihood of a threat occurring.
- **Correction**: Implementing measures to reduce the impact of a threat if it does occur.
- **Detection**: Implementing measures to identify when a threat has occurred.*

2.1 EINOO

2.2 STRIDE

3 CIA Triad

CIA Triad is a widely used model for understanding and implementing security in distributed systems. It consists of three core principles: Confidentiality, Integrity, and Availability.

3.1 Confidentiality

Confidentiality refers to the protection of sensitive information from unauthorized access or disclosure. This is often achieved through encryption, access controls, and secure communication protocols.

3.2 Integrity

Integrity ensures that data remains accurate and unaltered during transmission or storage. This is often achieved through hashing, checksums, and digital signatures.

3.3 Availability

Availability refers to the ability of a distributed system to remain operational and accessible to users, even in the face of failures or high demand. This is often achieved through redundancy, load balancing, and failover mechanisms.

4 Remote Procedure Call (RPC)

4.1 gRPC

5 Dependency Injection

6 Cryptography

6.1 Symmetric Encryption

6.2 Asymmetric Encryption

7 Integrity, Authentication, Digital Signatures

7.1 Roles

8 Web Services

8.1 REST

8.2 SOAP

9 Indirect Communication

Indirect communication is a communication paradigm in distributed systems where messages are sent through an intermediary rather than directly between the sender and receiver. This approach can help decouple components, improve scalability, and enhance fault tolerance.

Can be categorized into:

- State based
  - Tuple Spaces
  - Distributed Shared Memory
- Message based
  - Group Communication
  - Publish-Subscribe
  - Message Queues

9.1 Group Communication (Message based)

Group communication is a communication paradigm in distributed systems where messages are sent to a group of recipients rather than to a single recipient. This approach is often used in scenarios where multiple entities need to receive the same information, such as in multicast or broadcast communication.

9.1.1 Open Group Communication

Open Group Communication is a middleware that provides group communication services for distributed systems. It enables entities to send and receive messages to and from groups of recipients, facilitating efficient and reliable communication in distributed environments.

9.1.2 Closed Group Communication

Closed Group Communication is a communication paradigm in distributed systems where messages are sent to a predefined group of recipients. Only members of the group can send and receive messages, providing a controlled and secure communication environment.

9.2 Publish-Subscribe (Message based)

Publish-subscribe (pub-sub) is a messaging pattern in distributed systems where senders (publishers) send messages to a topic without knowledge of the subscribers, and receivers (subscribers) receive messages from topics they are interested in without knowledge of the publishers. This decouples the sender and receiver, allowing for greater scalability and flexibility.

9.3 Message Queues (Message based)

Message queues are a communication paradigm in distributed systems where messages are stored in a queue and processed asynchronously by the receiving entities. This approach helps decouple the sender and receiver, allowing for greater scalability and fault tolerance.

It is a type of asynchronous and indirect communication.

9.3.1 RabbitMQ

RabbitMQ is an open-source message broker that implements the Advanced Message Queuing Protocol (AMQP). It enables distributed systems to communicate asynchronously by sending and receiving messages through queues.

9.3.2 Apache Kafka*

Apache Kafka is a distributed event streaming platform that allows for the real-time processing and analysis of large volumes of data. It is designed for high throughput, fault tolerance, and scalability, making it suitable for building distributed systems that require reliable and efficient data communication.