Distributed Systems

Eduard Fekete

Character Count: 25813
Word Count: 3634

January 9, 2026

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.

1.2.2.1 Interaction Model Deals with the timing of events and the speed of communication. - **Synchronous**: Processes execute in lock-step; there are upper bounds on message delays and execution speeds. - **Asynchronous**: No bounds on message delays or execution speeds. Most real-world distributed systems (like the Internet) are asynchronous.

1.2.2.2 Failure Model Defines the ways in which components may fail. - **Omission Failures**: A process or communication channel fails to perform actions it is supposed to (e.g., dropped packets). - **Arbitrary (Byzantine) Failures**: Components fail in arbitrary ways, including sending conflicting or malicious data. - **Fail-Stop**: A process stops and other processes can detect that it has stopped.

1.2.2.3 Security Model Identifies the threats to the system and the assumptions made about the environment (already covered in Threats and CIA Triad).

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

- **E - Eavesdropping**: Unauthorized monitoring or interception of communication.
- **I - Interruption**: Making a resource or service unavailable or unusable (DoS).
- **N - Non-repudiation**: The ability to ensure that a party to a contract or a communication cannot deny the authenticity of their signature on a document or a message that they originated.
- **O - Observation**: Monitoring traffic patterns or other metadata without necessarily reading the content.
- **O - Originality (Replay)**: Ensuring that a message is fresh and not a replay of an old one.

2.2 STRIDE

STRIDE is a threat modeling framework developed by Microsoft for identifying and categorizing potential security threats.

- **S - Spoofing**: Pretending to be someone or something you are not (e.g., IP spoofing, masquerading).
- **T - Tampering**: Unauthorized modification of data (e.g., changing values in a message or file).
- **R - Repudiation**: Denying that an action was performed (e.g., "I didn't send that money").
- **I - Information Disclosure**: Unauthorized access to sensitive information (e.g., data breach).
- **D - Denial of Service**: Making a system or service unavailable to its intended users.
- **E - Elevation of Privilege**: Gaining more access or permissions than authorized (e.g., a user becoming an administrator).

3 Distributed Coordination

3.1 Time and Clocks

In distributed systems, there is no global clock. Each node has its own local clock, which can drift over time.

- **Clock Drift**: The difference in frequency between two clocks.
- **Clock Skew**: The difference in the time values shown by two clocks.

3.1.1 Christian's Algorithm

Used for synchronizing a client's clock with a time server. The client requests the time from the server and adjusts its clock by adding half of the round-trip time (RTT).

3.1.2 Berkeley Algorithm

A master clock polls all other clocks, calculates an average time, and then tells each clock how to adjust its time to match the average. This approach ignores outliers.

3.2 Logical Time

When physical time synchronization is not possible or sufficient, logical clocks are used to establish a causal ordering of events.

3.2.1 Lamport Clocks

Introduced the "happens-before" ($\rightarrow$) relationship. Each process maintains a counter that is incremented before each event. When a message is sent, the sender includes its counter. The receiver updates its counter to $\max(\text{local counter}, \text{received counter}) + 1$.

3.2.2 Vector Clocks

An extension of Lamport clocks that allows for detecting concurrent events. Each process maintains a vector of counters (one for each process in the system).

3.3 Consensus

Consensus is the process of getting multiple nodes in a distributed system to agree on a single value or state.

3.3.1 Raft

A consensus algorithm designed to be easy to understand. It decomposes the problem into three sub-problems: leader election, log replication, and safety.

3.4 Distributed Mutual Exclusion

Ensures that only one process can access a shared resource at a time.

- **Centralized**: A central coordinator manages access.
- **Ring-based**: A token is passed around a ring of processes.
- **Ricart-Agrawala**: A permission-based algorithm using logic clocks.

3.5 Election Algorithms

Used to elect a leader among a group of processes.

- **Bully Algorithm**: The process with the highest ID becomes the leader.
- **Ring-based Election**: A token-passing approach to elect a leader.

4 Replication and Consistency

Replication involves storing data on multiple nodes to improve availability and performance.

4.1 CAP Theorem

The CAP theorem states that a distributed system can only provide two out of the following three guarantees:

- **Consistency (C)**: Every read receives the most recent write or an error.
- **Availability (A)**: Every request receives a (non-error) response, without the guarantee that it contains the most recent write.
- **Partition Tolerance (P)**: The system continues to operate despite an arbitrary number of messages being dropped (or delayed) by the network between nodes.

In a partitioned network, a system must choose between Consistency and Availability.

4.2 PACELC Theorem

An extension of the CAP theorem that addresses the trade-off between latency and consistency even when there are no partitions.

- **P**: In case of a **partition (P)**, one must choose between **availability (A)** and **consistency (C)**.
- **E**: **Else (E)** (no partition), one must choose between **latency (L)** and **consistency (C)**.

5 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.

5.1 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.

5.2 Integrity

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

5.3 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.

6 Remote Procedure Call (RPC)

Remote Procedure Call (RPC) is a communication paradigm in distributed systems that allows a program to invoke a procedure or function on a remote server as if it were a local call. RPC abstracts the underlying network communication, making it easier for developers to build distributed applications.

6.1 How RPC Works

RPC uses a client-server model. When a client calls a remote procedure:

1. **Client Stub**: The client calls a local procedure called a "stub". The stub packs the parameters into a message (Marshaling).
2. **Communication Platform**: The message is sent across the network to the server.
3. **Server Skeleton**: The server-side equivalent of the stub, the skeleton, unpacks the parameters (Unmarshaling).
4. **Service Implementation**: The server executes the actual procedure and sends the result back through the same path.

- **Marshaling**: The process of encoding data into a format suitable for transmission.
- **Unmarshaling**: The process of decoding data back into its original form.

6.2 Invocation Semantics (Delivery Guarantees)

When a client calls a remote procedure, different guarantees can be provided regarding whether the call was executed:

- **Maybe**: No delivery guarantee. The message might be lost, or the response might be lost.
- **At-least-once**: The client retries until it receives a response. The procedure might be executed multiple times. Requires **idempotent** operations (operations that can be executed multiple times without changing the result beyond the first execution).
- **At-most-once**: The server detects duplicate requests (e.g., using sequence numbers) and ensures the procedure is executed at most once.

6.3 gRPC

gRPC is an open-source remote procedure call (RPC) framework developed by Google. It uses HTTP/2 for transport, Protocol Buffers (Protobuf) as the interface description language, and provides features such as authentication, load balancing, and bidirectional streaming.

- **Interface Definition Language (IDL)**: Protocol Buffers are used to define the service interface and the structure of the payload messages.

7 Dependency Injection

Dependency Injection (DI) is a design pattern used in software development to achieve Inversion of Control (IoC) between classes and their dependencies. Instead of a class creating its own dependencies, they are provided to the class from an external source.

- **Inversion of Control (IoC)**: A principle where the control of object creation and lifecycle is shifted from the application code to a framework or container.
- **Tight Coupling**: When a class is dependent on a specific implementation of another class.
- **Loose Coupling**: When a class depends on an interface rather than a concrete implementation, making it easier to swap dependencies.

7.1 Types of Injection

1. **Constructor Injection**: Dependencies are provided through the class constructor. This is the most common and recommended approach.
2. **Setter Injection**: Dependencies are provided through public setter methods.
3. **Interface Injection**: The dependency provides an injector method that will inject the dependency into any client that passes it.

8 Cryptography

Cryptography is the practice of securing communication and data through the use of mathematical algorithms and protocols. It is used to protect sensitive information from unauthorized access, ensure data integrity, and authenticate users.

8.1 Symmetric Encryption

Symmetric encryption is a type of encryption where the same key is used for both encryption and decryption of data. This approach is efficient and fast, making it suitable for encrypting large amounts of data.

- **Key Exchange Problem**: The main challenge is how to securely share the secret key between parties.
- **Examples**: AES (Advanced Encryption Standard), DES.

8.2 Asymmetric Encryption

Asymmetric encryption, also known as public-key cryptography, is a type of encryption that uses a pair of keys: a public key for encryption and a private key for decryption.

- **Confidentiality**: Encrypt with the recipient's public key; only the recipient's private key can decrypt it.
- **Authentication**: Encrypt with the sender's private key; anyone with the public key can verify the sender's identity (Digital Signatures).
- **Examples**: RSA, Elliptic Curve Cryptography (ECC).

8.3 Hashing

Hashing is a process of converting data of any size into a fixed-size string of characters. It is a one-way function, meaning it cannot be reversed.

- **Integrity**: Used to verify that data has not been tampered with.
- **Salting**: Adding random data (a salt) to a password before hashing to prevent dictionary attacks or rainbow table attacks.

8.4 Public Key Infrastructure (PKI)

A framework that manages digital certificates and public-key encryption. It involves:

- **Certificate Authority (CA)**: A trusted entity that issues digital certificates.
- **Digital Certificates**: Electronic documents used to prove the ownership of a public key.

9 Integrity, Authentication, Digital Signatures

9.1 Integrity

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

9.2 Authentication

Authentication is the process of verifying the identity of a user or entity in a distributed system. This is often achieved through the use of passwords, biometrics, or digital certificates.

9.3 Digital Signatures

Digital signatures are cryptographic mechanisms used to verify the authenticity and integrity of digital messages or documents. They provide a way to ensure that the sender of a message is who they claim to be and that the message has not been altered during transmission.

9.4 Roles

Roles help define the responsibilities and permissions of different entities within a distributed system.

10 Web Services

Web services are software systems designed to support interoperable machine-to-machine communication over a network. They enable different applications to communicate and exchange data regardless of their underlying platforms or technologies.

10.1 REST

REST (Representational State Transfer) is an architectural style for designing networked applications. It relies on a stateless, client-server communication model and uses standard HTTP methods (GET, POST, PUT, DELETE) for interacting with resources.

10.1.1 REST Constraints

- **Stateless**: Each request from a client must contain all the information necessary to understand and complete the request. The server does not store client context.
- **Cacheable**: Responses must define themselves as cacheable or not to prevent clients from reusing stale data.
- **Uniform Interface**: Simplifies and decouples the architecture. Includes resource identification (URIs), resource manipulation through representations (JSON/XML), and self-descriptive messages.
- **Layered System**: A client cannot ordinarily tell whether it is connected directly to the end server or to an intermediary (proxy/load balancer).
- **HATEOAS** (Hypermedia as the Engine of Application State): Clients interact with the network application entirely through hypermedia provided dynamically by the application servers.

10.2 SOAP

SOAP (Simple Object Access Protocol) is a protocol for exchanging structured information in the implementation of web services. It uses XML as the message format and typically relies on HTTP or SMTP for message transmission.

- **WSDL (Web Services Description Language)**: An XML-based language used for describing the functionality offered by a web service.

11 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.

- **Space Decoupling**: Senders do not need to know who they are sending to.
- **Time Decoupling**: Senders and receivers do not need to be active at the same time.

Can be categorized into:

- State based
  - **Tuple Spaces**: A shared memory pool where processes can read, write, and take "tuples".
  - **Distributed Shared Memory**: Simplifies programming by providing a shared address space across nodes.
- Message based
  - Group Communication
  - Publish-Subscribe
  - Message Queues

11.1 Group Communication (Message based)

Group communication is a communication paradigm in distributed systems where messages are sent to a group of recipients.

- **Multicast**: Sending a message to a specific group of recipients.
- **Atomic Multicast**: Guarantees that either all correct processes receive the message or none of them do, and all receive them in the same order.

11.1.1 Open and Closed Groups

- **Open Group Communication**: Any process can send to the group; useful for service discovery.
- **Closed Group Communication**: Only members can send to the group; useful for internal coordination.

11.2 Publish-Subscribe (Message based)

Publish-subscribe (pub-sub) is a messaging pattern where publishers send messages to a topic, and subscribers receive messages from topics they are interested in.

- **Topic-based**: Subscribers join a "channel" or "topic".
- **Content-based**: Subscribers define filters based on the content of the messages (e.g., "all messages where temperature > 30").

11.3 Message Queues (Message based)

Message queues provide a point-to-point asynchronous communication channel. Messages are stored in a queue until processed.

- **Persistence**: Messages can be stored on disk to survive server restarts.
- **Acknowledgment**: Receivers must acknowledge successful processing before a message is removed from the queue.

11.3.1 RabbitMQ

RabbitMQ is an open-source message broker that implements the Advanced Message Queuing Protocol (AMQP). It uses "Exchanges" to route messages to various queues based on rules.

11.3.2 Apache Kafka*

Apache Kafka is a distributed event streaming platform. Unlike traditional message queues, Kafka is a **log-based** system.

- **Pull-based**: Consumers pull data from Kafka at their own pace.
- **Retention**: Data can be retained for a long period, allowing for event replaying.

11.4 AMQP

AMQP (Advanced Message Queuing Protocol) is an open standard protocol for message-oriented middleware. It provides features such as message routing, delivery guarantees, and security.

12 Sources

Distributed Systems. (2025). Google Books. https://books.google.dk/books?hl=en&lr=&id=d63sQPvBezgC&oi=fnd&pg=PR1

Peiris, P. (2017, March 29). Kafka in Indirect Communication Paradigm. Dzone.com; DZone. https://dzone.com/articles/message-brokers-in-indirect-communication-paradigm