

Introduction to Distributed Systems

Introduction to Distributed Systems

Key concepts, characteristics, and foundational ideas

1. What Is a Distributed System?

Aspect	Description
Definition	A collection of independent computers that appear to the user as a single coherent system.
Goal	Provide shared resources, reliability, scalability, and performance across multiple nodes.
Typical Examples	Cloud platforms, micro-service architectures, peer-to-peer networks, distributed databases, sensor networks.

2. Core Characteristics

Characteristic	Meaning	Why It Matters
Transparency	Hides the complexity of distribution (e.g., location, replication, concurrency).	Simplifies programming and user interaction.
Scalability	Ability to grow in size (more nodes) or capacity (more data) with graceful performance degradation.	Supports increasing workloads and user bases.
Reliability & Fault Tolerance	System continues to operate despite node or communication failures.	Guarantees availability and data integrity.
Concurrency	Multiple processes execute simultaneously, potentially interacting.	Enables high throughput and resource utilization.
Heterogeneity	Nodes may differ in hardware, OS, or network technology.	Allows integration of diverse resources.
Openness	Use of standard interfaces and protocols, enabling extensibility.	Facilitates interoperability and evolution.

3. Architectural Models

Introduction to Distributed Systems

Model	Typical Use Case	Key Features
Client-Server	Web services, file sharing	Centralized servers, multiple clients.
Peer-to-Peer (P2P)	File distribution (BitTorrent), blockchain	Symmetric nodes, no dedicated server.
Three-Tier / Multi-Tier	Enterprise applications	Presentation, logic, and data layers on separate machines.
Service-Oriented Architecture (SOA) / Microservices	Cloud-native apps	Small, independent services communicating via APIs.
Distributed Shared Memory (DSM)	Parallel computing	Abstracts memory as a single address space across nodes.
Hybrid	Large-scale data centers	Combines multiple models for flexibility.

4. Communication Mechanisms

Mechanism	Synchronous vs. Asynchronous	Typical Protocols
Remote Procedure Call (RPC)	Synchronous (blocking)	gRPC, Thrift, JSON-RPC
Message Passing	Asynchronous (non-blocking)	AMQP, MQTT, Kafka, ZeroMQ
Publish/Subscribe	Asynchronous, decoupled	MQTT, NATS, Google Pub/Sub
Shared Memory (DSM)	Synchronous/Asynchronous depending on implementation	RMI, CORBA (legacy)
REST/HTTP	Typically request/response (synchronous)	HTTP/HTTPS, OpenAPI

5. Consistency Models

Model	Guarantees	Typical Scenarios
Strong Consistency	All reads see the latest write.	Financial transactions, critical data.

Introduction to Distributed Systems

Model	Guarantees	Typical Scenarios
Linearizability	Real-time ordering of operations.	Distributed locks, leader election.
Sequential Consistency	Operations appear in a total order, but not necessarily real-time.	Multi-core memory systems.
Eventual Consistency	Updates propagate eventually; reads may be stale temporarily.	Social feeds, large-scale caches.
**Causal Cons		