
Programación Distribuida

¿Qué es Programación Distribuida?

- Paradigma de programación que busca desarrollar sistemas donde tareas se ejecutan en diferentes nodos.
- Los nodos pueden ser servidores, PCs, dispositivos móviles, etc.
- Permite dividir tareas y coordinar su resolución simultánea.
- Enfocada en la concurrencia, comunicación y coordinación.
- Ejemplo: procesamiento de datos en paralelo utilizando múltiples máquinas.

¿Qué es un Sistema Distribuido?

- Colección de computadores independientes conectados que aparentan ser un solo sistema coherente al usuario.
- Colaboran coordinando acciones y compartiendo recursos.
- Interactúan principalmente mediante el paso de mensajes.
- Objetivo: compartir recursos, alcanzar fiabilidad, disponibilidad y escalabilidad.
- Ejemplo: Google Search, Internet, base de datos distribuida.

Diferencias y Similitudes Clave

- Programación distribuida: enfoque en el desarrollo y técnicas para aplicaciones que funcionarán en sistemas distribuidos.
- Sistemas distribuidos: enfoque en la arquitectura y la coordinación del sistema como un todo.
- Similitud: ambos buscan distribución de tareas, cooperación y comunicación entre nodos.
- Diferencia: uno es el paradigma/herramienta (programación), el otro el entorno/arquitectura (sistema).

Paradigmas de Programación Distribuida

- Programación concurrente: ejecución simultánea de procesos/hilos.
- Programación orientada a eventos: reacción según estímulos externos.
- Programación orientada a mensajes: comunicación por envío/recepción de mensajes.
- Programación orientada a objetos distribuidos: objetos que interactúan a través de la red.
- Programación funcional distribuida: ejemplo, Erlang, Elixir.

Arquitecturas de Sistemas Distribuidos

- Cliente-servidor: nodos cliente acceden a servicios en nodos servidores.
- Peer-to-peer (P2P): todos los nodos tienen capacidades similares.
- Arquitecturas multinivel (n-capas): separación de presentación, lógica y datos.
- Arquitectura de microservicios: servicios independientes que colaboran.
- Orientada a servicios (SOA) y basada en eventos (publish/subscribe).

Comunicación y Sincronización

- Comunicación por paso de mensajes: envío/recepción de información entre procesos o nodos.
- Sincronización: coordinar el orden y momento de las operaciones.
- Protocolos de comunicación: TCP/IP, HTTP, RPC, gRPC, RMI.
- Sincronización mediante semáforos, relojes, algoritmos de consenso (Paxos, Raft).
- Modelos síncronos y asíncronos.

Tolerancia a Fallos y Redundancia

- La tolerancia a fallos permite mantener la operación aun si fallan algunos componentes.
- Redundancia: duplicar componentes, datos o procesos.
- Técnicas: replicación activa/pasiva, almacenamiento estable, punto de control (checkpointing), consenso distribuido.
- Conmutación por error automática y degradación elegante.
- Ejemplos en bancos, comercio electrónico, computación en la nube.

Ventajas de la Programación y Sistemas Distribuidos

- Escalabilidad: fácil crecimiento horizontal añadiendo nodos.
- Redundancia y alta disponibilidad.
- Compartición de recursos y procesamiento paralelo.
- Flexibilidad geográfica y colaboración remota.
- Mejor adaptación a grandes cargas, resiliencia ante fallos.

Desafíos Técnicos de los Sistemas Distribuidos

- Complejidad en el desarrollo y depuración.
- Comunicación lenta o inconsistente por la red.
- Consistencia de datos: mantener información uniforme en todos los nodos.
- Tolerancia y detección de fallos parciales.
- Seguridad y protección frente a ataques distribuidos.

Herramientas y Frameworks de Programación Distribuida

- Message Passing Interface (MPI): estándar en HPC y clusters científicos.
- Apache Hadoop/Spark: procesamiento de datos y big data.
- Erlang/Elixir: concurrencia, tolerancia a fallos y mensajería eficiente.
- gRPC, SOAP, REST: protocolos para servicios distribuidos.
- Kubernetes y Docker: orquestación y despliegue de contenedores.
- Middlewares: RabbitMQ, Kafka, ZeroMQ, CORBA.

Ejemplos

- Comunicación básica entre procesos con envío de mensajes.
- Sincronización distribuida usando semáforos.
- Implementación de tolerancia a fallos por simulación de fallos y recuperación.
- Demostraciones de escalabilidad agregando nodos.
- Procesamiento paralelo de tareas y recolección de resultados.

Casos de Uso

- Computación científica y simulaciones en clústeres HPC (ejemplo: predicción meteorológica).
- Grandes servicios web (ejemplo: Google, Netflix, Facebook).
- Bases de datos distribuidas usadas en banca, telecomunicaciones y salud.
- Blockchain y criptomonedas (redes p2p de validación y registro).
- Big Data y análisis masivo (Amazon, airlines, NBA, sector salud).
- Internet de las Cosas (IoT): dispositivos conectados gestionados y coordinados en la nube.

Buenas Prácticas

- Patrones de diseño: balanceo de carga, caché distribuida, colas de mensajes.
- Observabilidad y monitoreo proactivo (Prometheus, Grafana).
- Modularidad y desacoplamiento de componentes.
- Considerar el teorema CAP: consistencia, disponibilidad, tolerancia a particiones.
- Gestión de la latencia y optimizing del throughput.
- Seguridad, autenticación y control de acceso centralizados.
- Documentación clara, pruebas de resiliencia y automatización de despliegues.

Concepto	Qué estudia	Ejemplo de foco
Sistemas Distribuidos	La arquitectura, comunicación, sincronización y tolerancia a fallos de múltiples computadoras que cooperan como un solo sistema.	Diseño de clusters, coordinación, replicación, consenso, escalabilidad.
Programación Distribuida	Las técnicas y paradigmas de software para escribir programas que se ejecutan en sistemas distribuidos.	Modelos de comunicación (mensajes, RPC, objetos remotos), frameworks distribuidos, paralelismo.