Sistemas Distribuidos: Conceptos Clave y Modelos

Dr. Cosijopii García

March 6, 2025

1.1.1 Definición de Sistema Distribuido

Definición Formal (Tanenbaum & Van Steen)

Colección de computadoras autónomas que:

- Se presentan como un sistema único y coherente
- Coordinan acciones a través de paso de mensajes
- Comparten recursos mediante redes de comunicación

Ejemplos:

- Cluster de servidores web (Google, AWS)
- Sistemas blockchain (Bitcoin, Ethereum)
- Plataformas IoT conectadas (Smart Cities)

1.1.2 Objetivos Clave

- **Recursos compartidos**: Acceso unificado a:
  - Almacenamiento distribuido (Ej: HDFS en Hadoop)
  - Potencia computacional (Ej: SETI@home)
- **Transparencia**:
  - Acceso: Mismo interfaz para recursos locales/remotos
  - Ubicación: Usuario no conoce posición física (Ej: CDNs)
- **Escalabilidad**:
  - Horizontal: Añadir nodos (Ej: Kubernetes)
  - Vertical: Mejorar nodos existentes
- **Tolerancia a fallos**:
  - Replicación de datos (Ej: Apache Kafka)
  - Consenso distribuido (Ej: Raft algorithm)

1.2 Comparativa de Sistemas

Ventajas

- Escalabilidad horizontal: AWS añade 1000+ nodos/día
- Tolerancia a fallos: Google File System (3 réplicas)
- Menor latencia: CDNs (Akamai acerca contenido)
- Eficiencia costo: Raspberry Pi clusters

Desventajas

- Complejidad de diseño: CAP Theorem
- Seguridad distribuida: Surface attack mayor
- Consistencia eventual: DynamoDB (AWS)
- Costo inicial: Infraestructura de red

Caso de estudio: Twitter migró de monolithic a sistema distribuido (2010-2012) para manejar 500M+ tweets/día

Teorema CAP (Brewer)

- **C**onsistencia: Todos nodos ven mismos datos simultáneamente
- **A**vailability: Todo request recibe respuesta (no errores/timeouts)
- **P**artition Tolerance: Sistema funciona con particiones de red

Imposible tener las 3 simultáneamente

- **CP**: Sistemas financieros (Ej: HBase)
- **AP**: Redes sociales (Ej: Cassandra)
- **CA**: Solo en sistemas no distribuidos

1.3 Modelos Arquitectónicos

- **Cliente-Servidor**:
  - Web tradicional (Apache/Nginx + browsers)
  - Base de datos centralizada (Ej: PostgreSQL)
- **Peer-to-Peer (P2P)**:
  - BitTorrent (swarm de nodos)
  - Blockchain (sin autoridad central)
- **Three-Tier Architecture**:
  - Presentación (Frontend)
  - Lógica (API REST)
  - Datos (DB cluster)
- **Microservicios**:
  - Docker + Kubernetes (Ej: Netflix: 700+ microservicios)
- **Event-Driven**:
  - Apache Kafka (Ej: Uber: 1Tb+/día de eventos)

1.3.1 Modelo Cliente-Servidor

- **Ejemplos clave**:
  - Web tradicional
    - Servidores: Apache/Nginx
    - Clientes: Navegadores web
  - Bases de datos centralizadas
    - PostgreSQL, MySQL
    - Acceso controlado por servidor único
- **Características principales**:
  - Comunicación asimétrica
  - Servidor central como punto único de gestión
  - Clientes no comparten recursos

1.3.2 Modelo Peer-to-Peer (P2P)

- **Implementaciones notables**:
  - BitTorrent
    - Swarm de nodos colaborativos
    - Distribución descentralizada de archivos
  - Blockchain
    - Redes sin autoridad central
    - Consenso distribuido (Ej: Bitcoin)
- **Ventajas clave**:
  - Escalabilidad horizontal
  - Resistencia a fallos
  - Distribución de carga

1.3.3 Arquitectura Three-Tier

- **Capas fundamentales**:
  - Presentación (Frontend)
    - HTML/CSS/JavaScript
    - Frameworks: React, Angular
  - Lógica de negocio (API REST)
    - Spring Boot, Node.js
    - Comunicación vía HTTP
  - Datos (DB cluster)
    - MySQL Cluster, MongoDB Atlas
    - Replicación y sharding

1.3.4 Arquitectura de Microservicios

- **Implementación moderna**:
  - Docker + Kubernetes
    - Contenedorización de servicios
    - Orquestación automática
  - Caso real: Netflix
    - 700+ microservicios
    - Escalado independiente
- **Beneficios**:
  - Despliegues independientes
  - Tecnologías heterogéneas
  - Aislamiento de fallos

1.3.5 Arquitectura Event-Driven

- **Tecnologías principales**:
  - Apache Kafka
    - Stream processing en tiempo real
    - Mensajería pub/sub
  - Caso de éxito: Uber
    - 1Tb+ de eventos/día
    - Tracking en tiempo real
- **Ventajas**:
  - Escalabilidad masiva
  - Bajo acoplamiento
  - Tolerancia a latencia

Conclusiones

- Sistemas distribuidos resuelven escalabilidad global
- Compromisos fundamentales (CAP theorem)
- Nuevos paradigmas: Service Mesh, Serverless
- Retos actuales: Edge computing, Quantum distribution

Referencias

- Tanenbaum, A.S., Van Steen, M. (2017) Distributed Systems
- Coulouris, G. (2011) Distributed Systems: Concepts and Design
- Documentación AWS Architecture Center

¡Gracias!

¿Preguntas?