🧭

Introducción a los Sistemas Distribuidos

Definiciones y conceptos

Existen dos grandes tipos:

💡 **Sistema centralizado**: ordenador central y red de terminales sin capacidad de procesamiento (ej: el mainframe IBM System Z)

💡 **Sistema distribuido**: conjunto de nodos autónomos que cooperan entre sí para resolver un problema (generalmente proveer un servicio). Debemos añadir que tratan de aparentar ser un sistema único y coherente a sus usuarios.

▼ *¿Cuando debemos escoger uno frente al otro?*

Los sistemas distribuidos presentan una gran ventaja en:

- Economía: mejor relación rendimiento/coste.
- Procesamiento paralelo
- Escalabilidad
- Flexibilidad y modularidad

Y además permiten **acceso a recursos remotos** de forma que también **se pueden compartir** con otros sistemas (*en cierto modo podemos ver la WWW como un mega sistema distribuido compuesto de otros más pequeños*).

Sin embargo, debemos tener en cuenta que sufren en:

- Confidencialidad

    - ¿Cómo garantizamos que la información de sesión se mantiene entre el usuario final y el sistema?
- Seguridad
    - Al aumentar los nodos estamos aumentando la superficie de ataque.
- Comunicaciones
    - Son muy vulnerables a saturación, latencia y pérdida de paquetes.
- Aumento de la complejidad arquitectónica

▼ ¿Cómo consiguen los SD aparentar uno solo y coherente?

Aquí hablamos de la habilidad de un SD de ser transparente a los usuarios finales.

> Entendemos transparencia como el hecho que el usuario final no tiene que saber absolutamente nada del sistema que hay debajo.
> En pocas palabras, que **sólo necesite utilizar la interfaz**.

Hablamos de distintos tipos de transparencia:

- **Acceso**: el usuario accede a los recursos como si fueran locales.
    - Un ejemplo es un objeto creado por un middleware que aparenta ser local pero realmente el recurso que representa (por ejemplo un documento) se encuentra en un SD.
- **Localización**: no importa donde están los recursos
    - Relocalización: Movimiento de los recursos en caliente
    - Migración: Movimiento de los recursos en frío
- **Replicación**: Se crean copias de los recursos pero siguen aparentando uno único al usuario
    - Un buen ejemplo es Netflix porque van copiando series y peliculas de un servidor a otro para que la latencia y calidad sean aceptables en varios países.
- **Concurrencia**: Varios usuarios acceden simultáneamente sin entrar en conflicto

    - Un OLTP (Online Transaction Processing) es un buen ejemplo real de como conseguir esta transparencia. Los RPC o RMI no evitan los conflictos !
- **Fallos**: El sistema oculta y corrige los fallos
- **Escalado**: El sistema aumenta y disminuye su tamaño sin afectar a las aplicaciones.

La **transparencia completa** es **imposible** de alcanzar e incluso ciertas combinaciones de transparencias pueden afectar al rendimiento del SD de forma considerable.

▼ ¿Cómo coordinamos a los nodos de un SD?

Evidentemente si vamos a tener una colección de nodos que tratarán de solventar un problema juntos, tendrán que comunicarse. Esto presenta un reto porque la red (a no ser que se trate de una intranet o LAN) no es siempre fiable.

- Coordinación directa: acoplamiento temporal y referencial.
- Coordinación indirecta: (nos da más flexibilidad)
    - **Desacoplamiento referencial**: los nodos no se conocen explícitamente.
    - **Desacoplamiento temporal**: los nodos no necesitan estar activos simultáneamente para coordinarse.

▼ Tipos de SD

Acoplamiento

- Fuertemente acoplados: Procesadores que comparten memoria o buses de E/S.
- Débilmente acoplados: Procesadores autónomos interconectados por sistemas de comunicaciones.

Arquitectura SW

- Igual a igual (p2p)
    - Es un **sistema simétrico** en el sentido que todos los nodos hacen tareas semejantes que en su conjunto resuelven la actividad distribuida. La interacción es N - N.

- Cliente servidor
    - Es un **sistema asimétrico** puesto que los clientes solicitan servicios que son realizados en el servidor (que posteriormente envía los resultados). La interacción es N - 1.

Arquitecturas de los sistemas distribuidos (SW)

Las arquitecturas que vamos a ver son estrictamente de carácter lógico y organizativo, lo que nos indica que el *Software* va a ser más decisivo que el *Hardware*.

Las arquitecturas más básicas son:

- Igual a igual (p2p)
- Cliente-Servidor
    - Es actualmente la más utilizada (incluyendo a sus variantes).
    - Los componentes software que identifica son:
        - **Cliente**: Cualquier proceso que reclama/consume servicios.
        - **Middleware**: Interfaz Cliente/Servidor o Servidor/Servidor. Se puede ver como el SO de los SSDD.
        - **Servidor**: Cualquier proceso que proporciona un servicio (servidor de disco, impresión, comunicaciones, presentación, procesos, BBDD, seguridad, …)

▼ Arquitectura en capas

En este tipo de arquitectura agrupamos las tareas y responsabilidades en capas. Como regla general se puede ver que al incrementar el número de capas se distribuye más el sistema.

Arquitectura cliente-servidor de 1 capa

- El acceso a datos, lógica y presentación se ejecutan en el cliente → Modelo de cliente pesado.

Arquitectura cliente-servidor de 2 capas

- La presentación y la lógica la ejecuta el cliente → Modelo de cliente pesado.
- El servidor se encarga del acceso a los datos.
- Mientras que es sencillo de implementar para aplicaciones pequeñas, resulta muy inflexible al tratar de ampliar la aplicación o migrarla a otros entornos (por ejemplo pasar de Windows a Linux).

Arquitectura cliente-servidor de 3 capas

- La presentación se ejecuta en el cliente → Modelo de cliente ligero.
- La lógica reside en servidores esparcidos en la red.
- Los datos se encuentran desligados de la lógica (en servidores de BBDD).

Arquitectura cliente-servidor de N capas

- La idea fundamental es que ahora hay múltiples niveles de clientes y servidores en el sentido que un servidor puede ser cliente de otro (y así de forma sucesiva).

Arquitectura orientada a objetos y servicios

La idea fundamental es que el sistema se compone de objetos (componentes) que se intercontectan con llamadas a procedimientos (RMI o RPC). Dichos objetos se pueden encontrar en máquinas distintas y encapsulan datos por lo que hablamos de objetos con estado y objetos remotos.

Arquitectura orientada a recursos

Como base el sistema distribuido ahora se basa en componentes que gestionan individualmente parte de los recursos. La idea es que mediante peticiones las aplicaciones pueden añadir, eliminar, obtener y modificar los recursos.

▼ Un buen ejemplo son los **sistemas web** con las arquitecturas *RESTful* (Representational State Transfer):

- Los recursos se identifican con URI (Unique Resource Identifier).
- Los servicios ofrecen una interfaz común para añadir (POST), eliminar (DELETE), obtener(GET) y modificar (PUT) los recursos.

Arquitectura basada en eventos

Este modelo separa el procesamiento y la coordinación del sistema mediante un esquema de publicación y suscripción entre procesos autónomos. De esta forma consigue el desacoplamiento temporal (tras una publicación la respuesta puede tardar puesto que no se espera una respuesta de forma síncrona) y referencial (no se necesita conocer quien lanzó el evento).

▼ Modelos de SD

Servidores de BBDD

Los clientes pasan al servidor DBMS las consultas en SQL y este les responde a través de la red.

Proceso de transacciones

Los clientes ahora solicitan servicios mediante RPC al servidor. El procedimiento invocado agrupa las acciones necesarias (consultas SQL o de otro tipo) en transacciones, de forma que son tratadas como un grupo y si falla una fallan todas (**atomicidad**). Esto se hace mediante un gestor OLTP y permite en el desarrollo de la aplicación dejar la gestión de las transacciones en el servidor.

Servicios WEB

… pls fix this shit with tanenbaum

Características de los sistemas cliente - servidor

> En resumen, su arquitectura se caracteriza por la relación asimétrica (N-1) entre los clientes y el servidor. Considera como **unidad básica de diseño al servicio**. Los clientes son (generalmente) activos enviando y esperando la respuesta a las peticiones, mientras que los servidores son pasivos (generalmente) esperando a una petición.

Dicha arquitectura le otorga:

- Débil acoplamiento: interacción basada en mensajes

- Transparencia: el sistema aparece como única unidad de proceso
- Escalabilidad: horizontal (con balanceo de carga) y vertical.
    - El **escalamiento horizontal** consiste en añadir más nodos (servidores) al sistema.
    - El **escalamiento vertical** consiste en potenciar a los nodos dotándolos de más procesamiento y/o memoria.
- Integridad y facilidad de mantenimiento de los programas y datos si se centralizan en servidores.

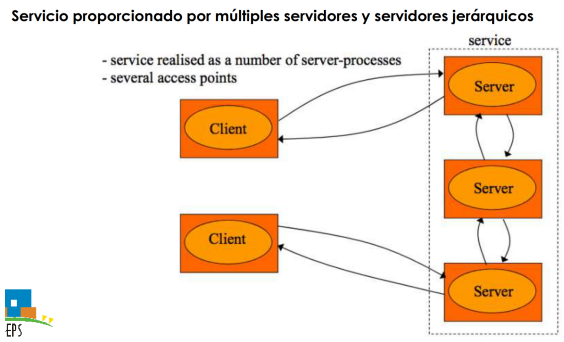
Es interesante recalcar que precisamente por la asimetría de las comunicaciones, el servidor se convierte en el cuello de botella. Para evitar problemas se juega con la actividad o pasividad y el escalado horizontal.

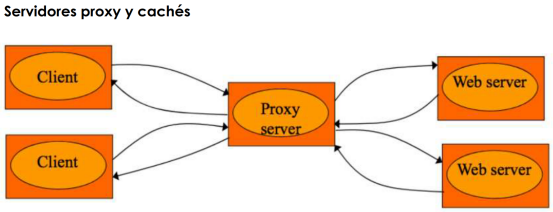
▼ **Detalles sobre la interacción entre un cliente y un servidor**

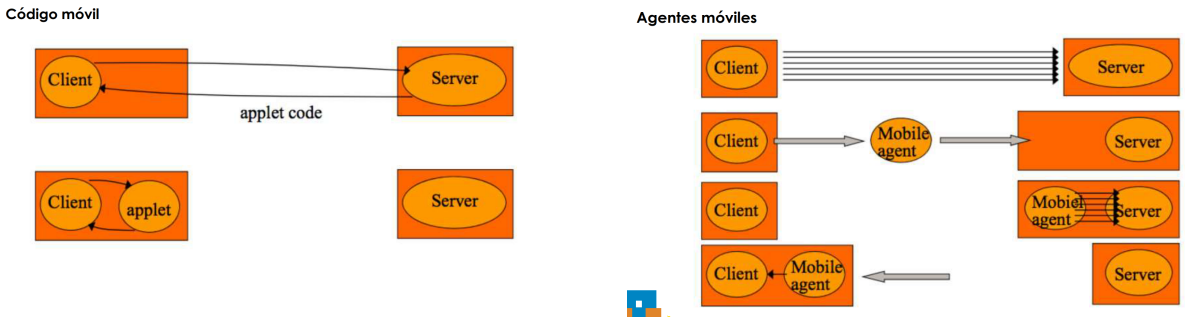
El modelo arquitectónico básico es el client **PULL** (cliente activo y servidor pasivo).
Existe otro modelo que es el server **PUSH** (cliente pasivo y servidor activo).

- Un buen ejemplo de modelo **PUSH** es precisamente los servidores de WhatsApp
    - Estos envían a los terminales los mensajes relevantes y descargan al cliente de tener que hacer checkeos periódicos de si han llegado mensajes. Además otra ventaja del modelo **PUSH** es que descarga al servidor de peticiones (e incluso si es puro, evita los ataques DDOS).

Variantes de la arquitectura cliente-servidor







Arquitectura de las aplicaciones WWW

Estos sistemas se basan (originalmente) en el modelo del **cliente ligero** con comunicación bajo HTTP.

En este tipo de sistemas solemos encontrar tres capas lógicas que forman la arquitectura:

- **Nivel del cliente**
  - Tradicionalmente el navegador (cliente universal) pero también puede ser un programa ejecutado del lado del cliente (cliente específico).
    - Un buen ejemplo de cliente específico es el programa de *Spotify*.
- **Nivel intermedio** (Nivel de lógica o *Business Tier*)
  - Se compone del Servidor Web (maneja las peticiones de los documentos HTML, CSS, JS) y del entorno de ejecución de aplicaciones (Servidor de Aplicaciones).
  - El servidor Web se encarga del proceso de peticiones y el envío de las respuestas
  - El servidor de aplicaciones se encarga de llevar a cabo la lógica de los servicios.
    - Un ejemplo clásico es el relleno de plantillas HTML mediante un interprete de PHP (método muy similar a Flask con Jinja2).
- **Nivel del backend**
  - Enlace con sistemas y programas antiguos (también denominados *legacy systems*).
  - El servidor web (o de aplicaciones) realiza consultas mediante un enlace via middleware o conectores.

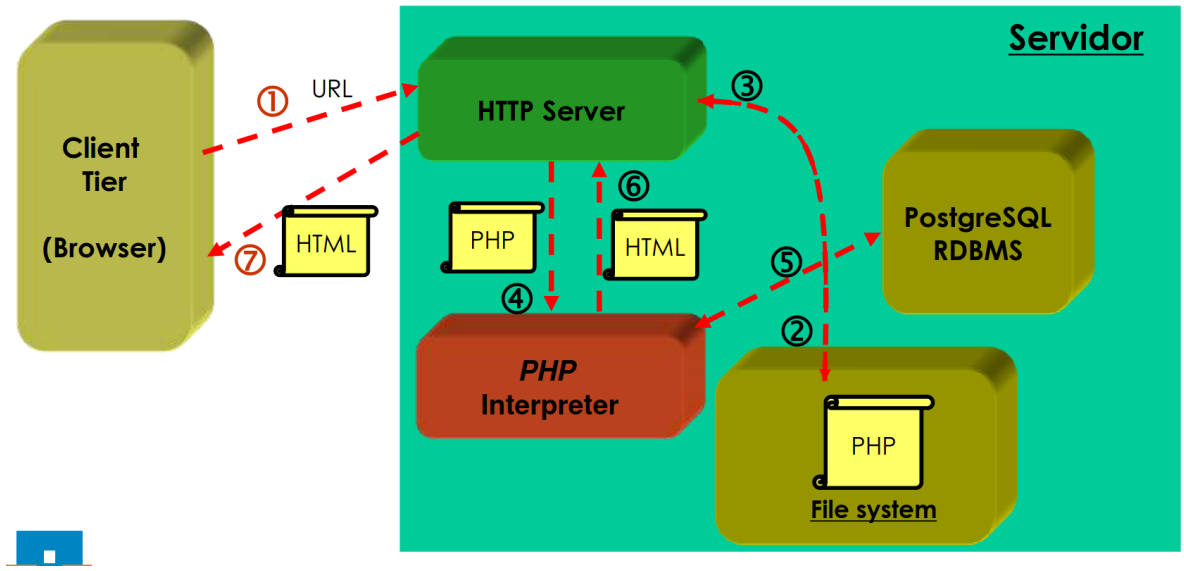
- Buenos ejemplos son DBS, OLTP e incluso mainframes.

▼ **¿Por qué necesitamos a los *legacy systems*?**

La filosofía de los SD es seguir evolucionando la forma en la que se estructuran sistemas informáticos. Sin embargo eso no significa que sustituyan a los esquemas y modelos antiguos.

> Es decir, los nuevos modelos, arquitecturas o paradigmas no sustituyen a los antiguos, sino que los extienden y complementan.

Ejemplo de los niveles mencionados anteriormente algo más específico:



Ejercicio: ¿Sabrías identificar las capas y los componentes de la figura?

Computación en la nube

Actualmente existen gigantes de la WWW (como Google, Amazon, Microsoft, …) que tuvieron que solventar una gran cantidad de problemas en su escalado. Amazon fue el pionero en ofrecer al mundo su sistema (AWS, *Amazon Web Services*) de modo que ofrece a los clientes la solución hecha a medida para sus necesidades. Esto fue el inicio del **cloud computing**.

Ventajas y desventajas de la computación en la nube

Entre las ventajas nos encontramos:

- Disminución de costes
  - Con **pay-as-you-go** (paga por lo que consumes) o **pay-as-per-use** (paga por lo que vas a usar) y el hecho que no necesitamos infraestructura HW/SW.
- Simplificación de desarrollos complejos
  - El uso de servicios generalmente requiere solo de configuración
  - Acceso a nuevas tecnologías junto a rapidez de desarrollo con menos riesgo.
- Administración
  - Los sistemas son altamente escalables y disponibles.
  - El proveedor se encarga de mantener los sistemas actualizados.
  - Permite la gestión con monitorización de tiempo y recursos.

Pero existen unas desventajas serias para considerar

- Privacidad: ¿A quien le pertenecen los datos una vez entran en el sistema?
- Falta de control y dependecia del proveedor (cada uno suele forzarte a su ecosistema)

Modelos de despliegue de SD en la nube

1. Nube privada: Sólo una organización tiene acceso al sistema
2. Nube pública: Cualquiera tiene acceso a la infraestructura.
3. Nube comunitaria: Acceso compartido entre varias empresas.
4. Nube híbrida: Composición de modelos de nubes.
   - En cierto modo es similar a la composición de la web de AS (*Autonomous Systems*).

Modelos de servicio de la nube

- **IaaS** : Se ofrecen servicios de virtualización, servidores, SO, … de tal forma que se puede desplegar cualquier tipo de sistema distribuido. El proveedor se encarga del mantenimiento, dejándole al cliente ciertas libertades en la configuración de los recursos que contrata.

- **PaaS** : El servicio se orienta a los desarrolladores proporcionando un entorno preconfigurado de desarrollo y ejecución. Estos clientes no gestionan ni controlan la infraestructura .
- **SaaS** : La capacidad de acceso se limita al uso de las aplicaciones que se ejecutan en la infraestructura (como el Word 365).