Tema 4.
Sistemas distribuidos.

Introducción. Modelos HW y SW.

Marisol García Valls

Departamento de Ingeniería Telemática
Universidad Carlos III de Madrid
mvalls@it.uc3m.es

Arquitectura de sistemas II

Grado en Ingeniería Telemática
Curso: 3º, cuatrimestre: 2º

Universidad Carlos III de Madrid

Índice

Sistema Distribuido

- Un **sistema distribuido** es aquél en el que sus **componentes** hardware o software ubicados en computadores **en red** **se comunican y coordinan** sus acciones **sólo pasándose mensajes**.

- Características:
  - **Concurrencia** de sus componentes.
    - Acceden a recursos compartidos (páginas web, ficheros, etc.)
  - **No existe un reloj global**.
    - La única comunicación es por paso de mensajes.
  - **Fallos independientes**.
    - Pueden causar aislamiento
    - Difícil detectar si es un fallo o es que la red se ha vuelto lenta.

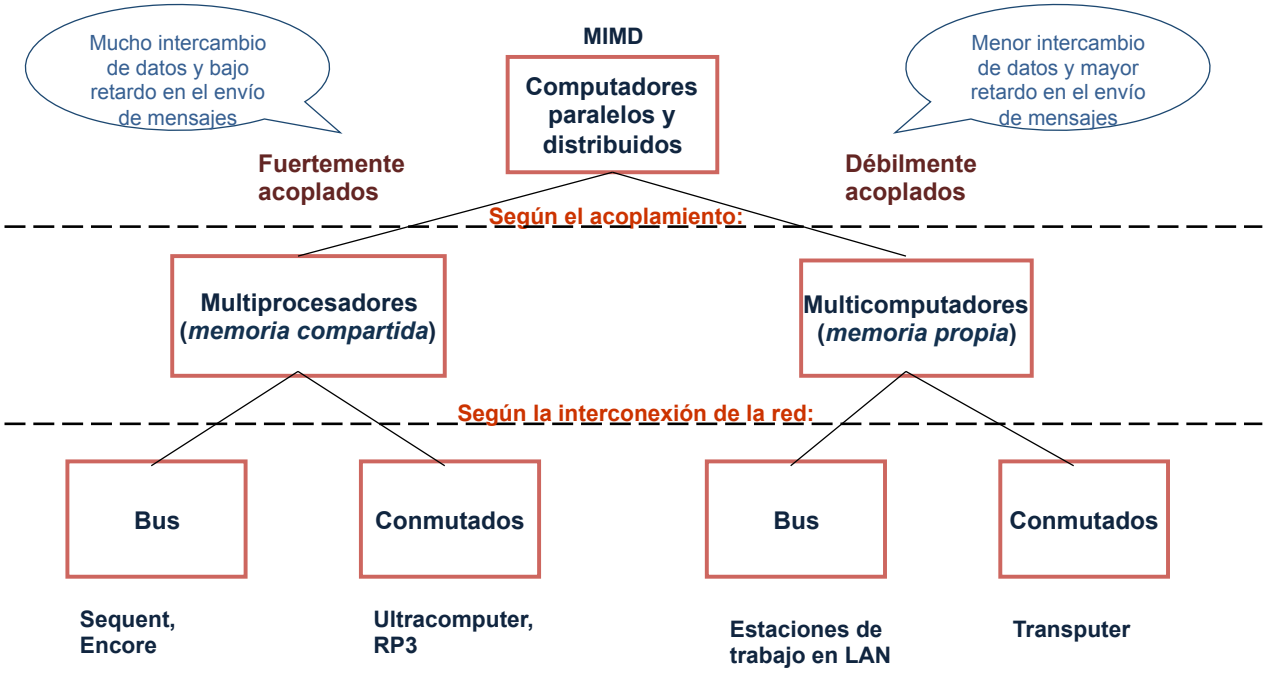
El Hardware de los sistemas distribuidos

- La *configuración del hardware* depende de **cómo están interconectados** los computadores y **cómo comunican**.
- Clasificación de Flynn (1992) según el número de:
  - Rutas de instrucciones.
  - Rutas de datos.

SISD	Una única ruta de instrucciones y de datos.
SIMD	Una única ruta de instrucciones y múltiples rutas de datos.
MISD	Múltiples rutas de instrucciones y una única ruta de datos.
MIMD	Grupo de computadores independientes, cada uno con su propio contador de programa, programa y datos.

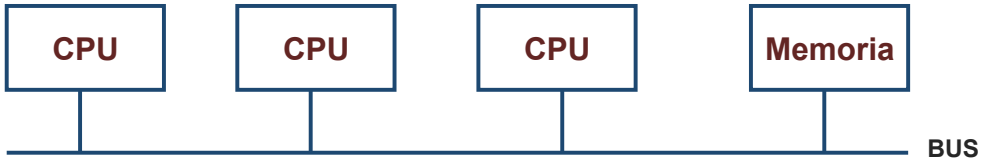
- Los sistemas distribuidos son MIMD

Tipos de sistemas MIMD



Multiprocesadores de bus

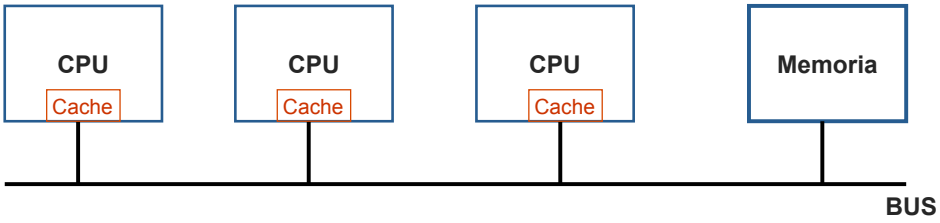
- Un multiprocesador basado en bus consta de **varias CPUs** y un módulo de **memoria** que se conectan a un **bus común.**



- Un bus tiene líneas de direcciones, líneas de datos y líneas de control.
- Para leer una palabra de memoria, una CPU coloca la dirección de la palabra en el bus de direcciones y la señal de lectura en las líneas de control.
- La memoria responde enviando el valor de la palabra por el bus de datos.
- Memoria **coherente**: si la CPU A escribe una palabra en memoria y B lee esa palabra después, B deberá obtener el valor que escribió A.
- Si existen muchas CPUs, este esquema produce **sobrecarga** en el bus.

Multiprocesadores de bus II

- La introducción de **memorias cache** de alta velocidad soluciona esto.
- Mantienen las palabras de memoria consultadas más recientemente.
- Todas las peticiones de memoria pasan por la cache.
- Se producirá un **acierto en cache** (*cache hit*) si la palabra solicitada está en la cache. En caso contrario, se produce un **fallo en cache** (*cache miss*).



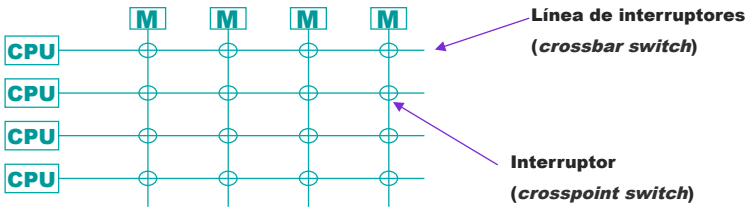
- A mayor tamaño de cache, menor sobrecarga del bus por tráfico y mayor podrá ser el número de CPUs en el sistema.

Multiprocesadores de bus III

- Las **caches** introducen un problema importante de **coherencia**:
  - Supongamos que dos CPU, *A y B, leen la misma palabra* de memoria en sus caches respectivas.
  - *A rescribe* la palabra (lo hace en su cache).
  - *B lee* el valor de esa palabra (valor antiguo que coge de su cache, y no el valor que A acaba de escribir).
  - La memoria ahora es ***incoherente***.
- Se han desarrollado algunas soluciones:
  - **Cache de acceso a memoria para escritura** (*write-through cache*). Se accede a memoria cuando se producen:
    - Fallos de cache para lectura.
    - Todas las escrituras (sean fallos o aciertos).
  - **Cache supervisora** (*snoopy cache o snooping cache*): monitoriza el bus constantemente para *detectar escrituras*.
- Un diseño con caches que combinen *write-through* y *snopping* es coherente y transparente al programador.
- Con este método es posible tener un número considerable de CPUs en un único bus (32 a 64).

Multiprocesadores conmutados

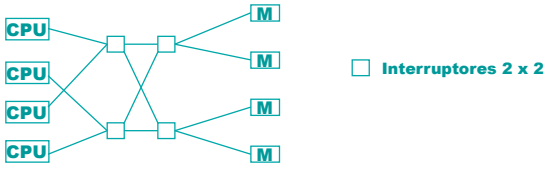
- Establecen una forma especial de interconectar CPUs y memoria.
- Permiten tener más de 64 procesadores.
- Se divide la memoria en módulos.
- Los módulos de memoria se conectan con las CPUs con una **línea de interruptores**.



- Para que una CPU tenga acceso a un módulo de memoria particular, el interruptor que las une se baja.
- Varias CPUs pueden acceder a memoria a la vez.
- El problema radica en que el **número de interruptores** necesarios puede ser **prohibitivo**.

Multiprocesadores conmutados II

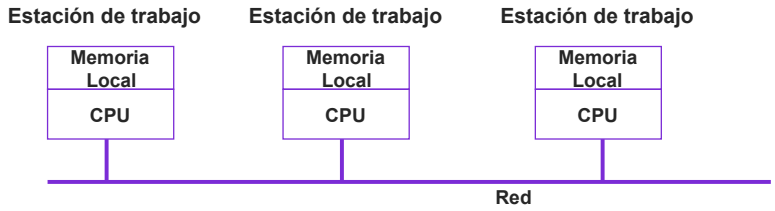
- Existen otras arquitecturas de conmutación en multiprocesadores que intentan **reducir el número de interruptores**.
- Máquinas basadas en la **red omega**:
  - Un interruptor une a más de una CPU con una o más memorias.
  - Presentan mayor retardo.



- Sistemas jerárquicos y **NUMA** (Non Uniform Memory Access):
  - Cada CPU tiene acceso a su módulo de memoria local (acceso rápido) y el módulo de memoria de otros (acceso más lento).
  - Tienen menor tiempo medio de acceso que omega.
  - Tienen algoritmos complicados para la distribución del código de programas de forma óptima.
- Es *complicado y caro* desarrollar grandes multiprocesadores de memoria compartida y fuertemente acoplados.

Multicomputadores de bus

- Cada CPU tiene una conexión directa a su memoria local.
- El tráfico generado por la comunicación de CPU a CPU será mucho menor que para CPU a memoria.
- Debido al menor tráfico no es necesaria la existencia de un bus de alta velocidad; puede ser una LAN.



- La red puede ser una LAN o un bus de alta velocidad (*high speed bus backplane*).

Multicomputadores conmutados

- En ellos cada CPU tiene acceso directo y exclusivo a su propia memoria, que es privada.
- Existen diferentes topologías de interconexión.
- La topología en **cuadrícula** (o *grid*) es la más sencilla y suele utilizarse en las tarjetas y placas.
- La topología en **hipercubo** (*hypercube*) es n-dimensional. Cada CPU tiene tantas conexiones como el número de dimensiones de la topología.

El software de los sistemas distribuidos

Índice

- Capas software de un sistema distribuido.
  - Sistema operativo
  - Middleware
- Arquitecturas básicas de sistemas distribuidos
  - Cliente-servidor, Múltiples servidores, Servidores proxy y caché, Procesos de igual a igual (*peer-to-peer*)
- Variantes del modelo cliente-servidor
  - Código móvil, Agentes móviles, Computadores de red, Dispositivos móviles y enlace espontáneo a la red.
- El sistema operativo de los sistemas distribuidos
- Conceptos de diseño de sistemas operativos

El sistema operativo en sistemas distribuidos

- El software del **sistema operativo** es la parte más importante y que **presenta la imagen del sistema** a los usuarios.

- Básicamente son:
  - **Fuertemente acoplados** o
  - **Débilmente acoplados**.

- El acoplamiento débil permite que las **máquinas y usuarios** de un sistema distribuido sean **independientes** pero **puedan interactuar**.

(a) Sistemas operativos de red

- Los **sistemas operativos de red** imponen pocos requisitos de sistema.
- Pueden existir distintas configuraciones: clientes, servidores de ficheros, etc.
- Clientes y servidores pueden ejecutarse en distintos sistemas operativos
  - deben acordar el formato y significado de los mensajes.

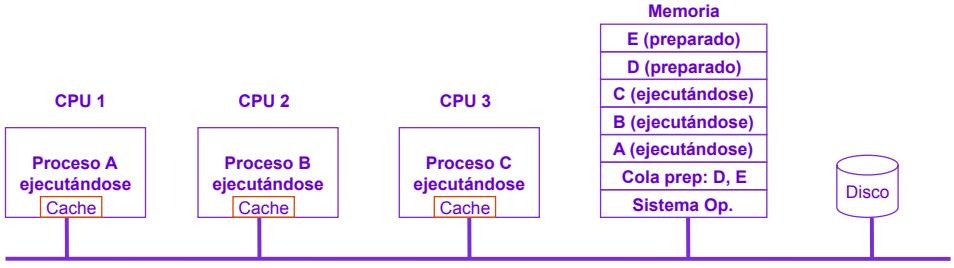
- **Es software débilmente acoplado sobre hardware débilmente acoplado**.

(b) Sistemas operativos distribuidos

- Un **sistema (operativo) distribuido** es **software es fuertemente acoplado sobre hardware débilmente acoplado**
- El software da una **imagen única del sistema** y crea una imagen de **monoprocesador virtual**
- Mecanismo global de comunicación entre procesos.
- Mecanismo global de protección.
- La gestión de procesos debe ser la misma en todos los subsistemas.

(c) Sistemas multiprocesador de tiempo compartido

- Son un **software fuertemente acoplado** sobre **hardware fuertemente acoplado**.
- Existen varias máquinas de propósito general en esta categoría
- Por ejemplo, máquinas de *bases de datos dedicadas* y *multiprocesadores de tiempo compartido con UNIX*.
- Existe una única cola de procesos preparados que se mantiene en la memoria compartida.
- Implicaciones:
  - El planificador deberá ejecutarse como una sección crítica para evitar que dos CPUs escojan para ejecutar el mismo proceso.
  - Deberán emplearse mecanismos de exclusión mutua para este fin.



Comparación entre sistemas

	S.O. de red	S.O. distribuido	S.O. multiprocesador
¿Parece un monoprocesador virtual?	No	Si	Si
¿Todos se ejecutan con el mismo S.O.?	No	Si	Si
¿Cuántas copias del S.O. hay?	n	n	1
¿Cómo se realiza la comunicación?	Ficheros compartidos	Mensajes	Memoria Compartida
¿Se deben acordar los protocolos de red a utilizar?	Si	Si	No
¿Existe una única cola de ejecución?	No	No	Si
¿Está bien definida la semántica de compartición de ficheros?	Generalmente no	Si	Si

Conceptos de diseño de sistemas operativos

- Transparencia
- Flexibilidad
- Fiabilidad
- Desempeño
- Escalabilidad

Transparencia (*transparency*)

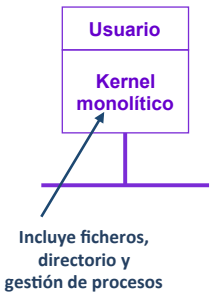
- Apariencia de ser un sistema de tiempo compartido y monoprocesador.

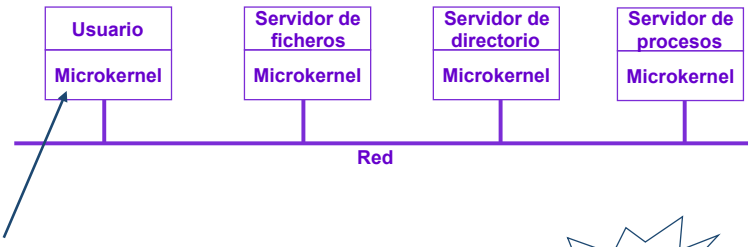
- Hay dos niveles de transparencia:
  - **Alto nivel**: esconder los detalles de distribución a los usuarios.
  - **Bajo nivel**: transparencia a los programas (mediante una interfaz de llamadas al sistema).

- La transparencia puede ser de:
  - **Ubicación**: no se puede saber dónde están los recursos.
  - **Migración**: los recursos pueden moverse sin cambios en su nombre.
  - **Réplica**: los usuarios no pueden saber cuántas copias existen.
  - **Concurrencia**: varios usuarios pueden compartir recursos automáticamente.
  - **Paralelismo**: las actividades pueden suceder en paralelo sin que los usuarios lo perciban.

Flexibilidad (*flexibility*)

- **Kernel monolítico**. En cada máquina debería ejecutarse un kernel tradicional que incluya la mayoría de servicios.
  - Es el SO centralizado con funcionalidad de red y de integración de servicios remotos.
- **Microkernel**. El kernel ofrecerá la funcionalidad mínima indispensable; la mayoría de los servicios del SO estarán disponibles en servidores de usuario.





Un microkernel es muy flexible ya que es mínimo.
Únicamente incluye:
- Planificación y gestión de procesos básica y de bajo nivel.
- Algo de gestión de memoria.
- Un mecanismo de comunicación entre procesos.
- Entrada / salida de bajo nivel.

Fiabilidad (*reliability*)

- Ejemplo: si una máquina cae, otra deberá continuar el trabajo que ésta ya no puede realizar.
- La ***fiabilidad*** de un sistema está relacionada con aspectos de: disponibilidad, redundancia, seguridad y tolerancia a fallos.
- La **disponibilidad** de un sistema se refiere al porcentaje de tiempo que dicho sistema es utilizable.
- La **redundancia** implica replicar piezas clave de hardware y software.
- La **seguridad** de un sistema se refiere a proteger ficheros y otros recursos de la utilización no autorizada.
- La **tolerancia a fallos** implica diseñar el sistema para enmascarar los fallos que se produzcan; ocultarlos al usuario.

Desempeño (*performance*)

- La velocidad de respuesta debería ser comparable a las ejecuciones sobre un sistema centralizado.
- Las comunicaciones son más lentas en sistemas distribuidos.
- **Métricas**: rendimiento (*throughput*, número de trabajos por *udt*), utilización del sistema y capacidad de red consumida.
- Hay que prestar atención a la granularidad de las operaciones:
  - Paralelismo fino. Pequeños trabajos, comunicantes.
  - Paralelismo grueso. Trabajos de gran duración, poco comunicantes.

Escalabilidad (*scalability*)

- Capacidad de crecimiento de un sistema.
- Cada vez es posible construir sistemas distribuidos más grandes (*large scale systems*).

- Aspectos a evitar: por ejemplo, la centralización.
  - centralizar componentes (único servidor), información (única base de datos) y algoritmos (que posean información completa).

- Aspectos a fomentar: utilizar algoritmos descentralizados:
  - Ninguna máquina posee la información completa del estado de un sistema.
  - Las máquinas toman decisiones basándose únicamente en información local.
  - El fallo de una máquina no arruina el algoritmo.
  - No se realizan suposiciones implícitas sobre la existencia de un reloj global.

Arquitecturas de software de sistemas distribuidos

Arquitecturas básicas:

- Las arquitecturas se clasifican según la ubicación de los procesos en los computadores.

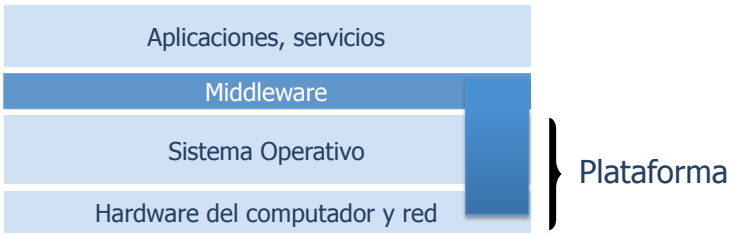
- Modelo cliente-servidor.
- Servicios proporcionados por múltiples servidores.
- Servidores proxy y cachés.
- Procesos de igual a igual (*peer to peer*).

Variaciones del modelo cliente-servidor:

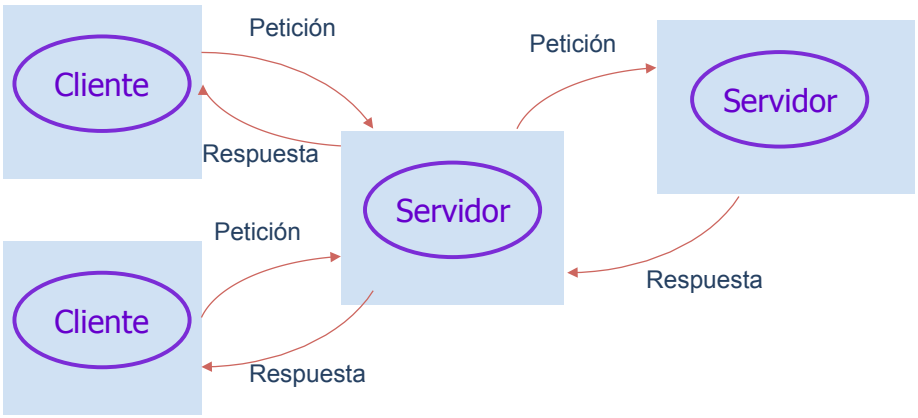
- Código móvil.
- Agentes móviles.
- Computadores de red.
- Clientes ligeros.
- Dispositivos móviles y enlace espontáneo.

Capas de software. El middleware

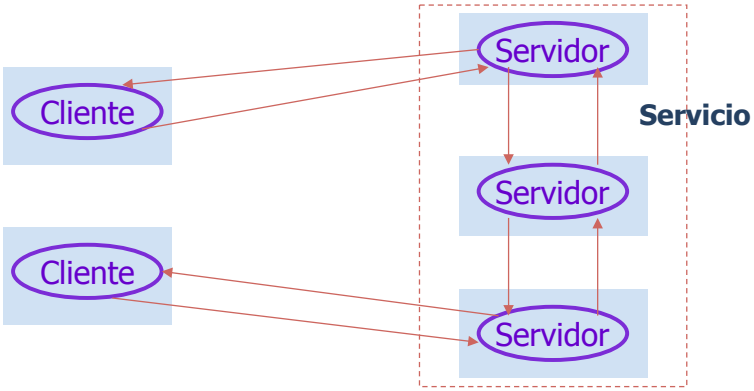
- El **middleware enmascara la heterogeneidad** de las plataformas.
- Consiste en un **conjunto de procesos y objetos** en un conjunto de computadores que interactúan entre sí para implementar **mecanismos de:**
  - **comunicación** y
  - de **gestión de recursos compartidos** para aplicaciones distribuidas.
- Proporciona bloques útiles para la construcción de componentes software de un sistema distribuido.
- Soporta **abstracciones** como: **invocación remota**, **comunicación entre grupos** de procesos, **notificación de eventos**, **replicación de datos compartidos** y transmisión de datos de tiempo real.



Cliente - Servidor

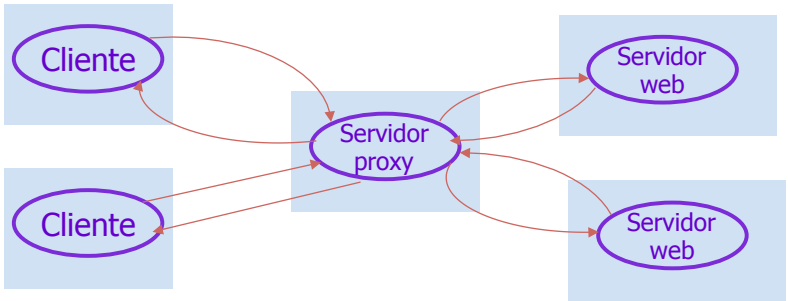


Servicio proporcionado por múltiples servidores



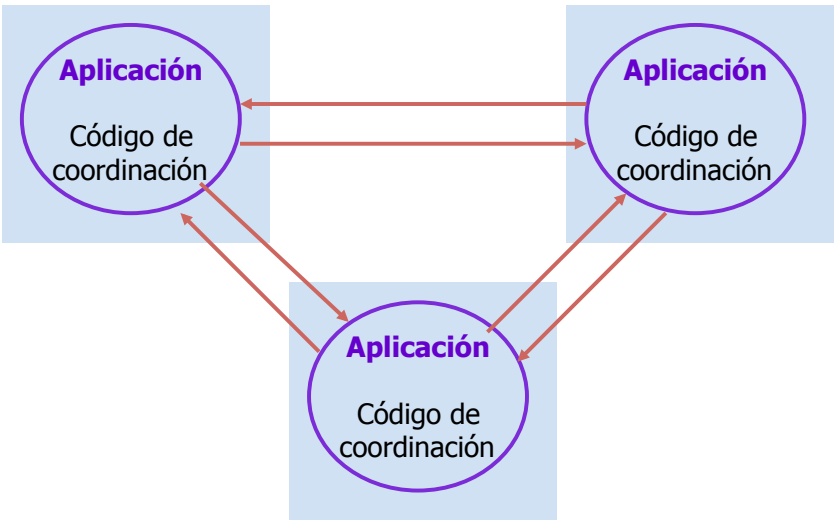
- **Un servicio** implementado como **varios procesos** de servidor **en computadores separados**.
- **Interaccionan cuando es necesario** para proporcionar un servicio a un cliente.
- Los objetos de un servicio pueden estar distribuidos entre los distintos servidores.
- Varios/todos **los objetos** de un servicio pueden estar **replicados** entre servidores. Esto mejora las prestaciones, disponibilidad y tolerancia a fallos.

Servidores proxy y caches



- Una **caché** es un **almacén de objetos** de datos **utilizados recientemente**.
- Se encuentra **más próxima** que los objetos en sí.
- Al recibir un objeto nuevo en un computador, se añade a la caché.
- Si un cliente necesita un objeto, el servicio caché comprueba inicialmente la caché y le proporciona el objeto de una copia actualizada.
- Las **cachés** pueden estar ubicadas **en cada cliente o** en un **servidor proxy** compartido por varios clientes.

Procesos de igual a igual (*peer-to-peer*)



- Todos los procesos desempeñan tareas semejantes, interactuando cooperativamente.
- No hay distinción entre clientes y servidores.
- El código de los procesos iguales mantiene la consistencia de los recursos y sincroniza las acciones a nivel de aplicación cuando es necesario.

Variantes del modelo cliente-servidor

- Computadores de red.
- Código móvil.
- Agentes móviles.
- Dispositivos móviles y enlace espontáneo a la red.

Computadores de red

- Simplifican la gestión de los archivos de aplicación y mantenimiento de software de base local.
- Un computador de red **descarga su sistema operativo** y cualquier **aplicación software** desde un servidor de archivos remoto.
- Las aplicaciones se lanzan localmente.
- Los archivos se gestionan desde un **servidor de archivos remoto**.
- Las capacidades de procesador y memoria pueden reducirse.
- Si tienen disco, éste aloja un software mínimo.
- El resto de **disco** se usa como almacenamiento intermedio o **caché**.
- La caché no necesita mantenimiento manual alguno.

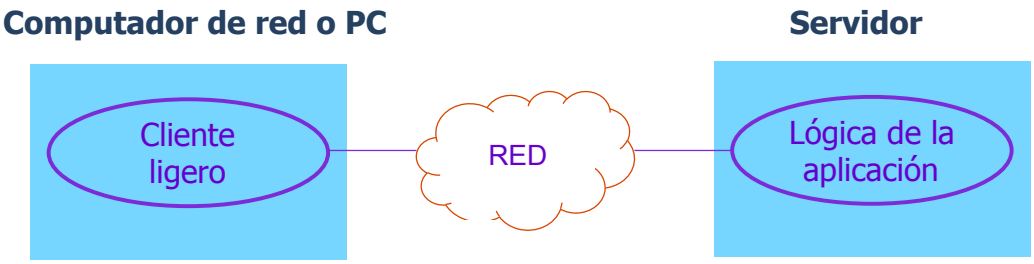
Código móvil

- El ejemplo más conocido: ***applets***.
- Al seleccionar un enlace a un applet, se descarga su código desde el servidor web donde esté y se ejecuta.
- Esta ejecución local da respuesta interactiva.
- Si el applet necesita código adicional no presente en el navegador cliente, puede descargarlo también.
- Este código puede ser para comunicarse desde el servidor al cliente. Ejemplo donde un broker informa a sus usuarios de los cambios en el valor sus acciones.
- Este es el modelo push, donde, en lugar del cliente, el servidor es el que inicia las interacciones.

Agentes móviles

- Un agente es un programa en ejecución (lo que incluye tanto código como datos).
- Un agente móvil es un **programa en ejecución que se traslada** de un computador a otro realizando alguna función para alguien.
- Ejemplo 1: recolecta información y regresa con los datos.
- **Reducen** el coste de la **comunicación** y el **tiempo** (sustituye solicitudes remotas por locales).
- Ejemplo 2: instalar y mantener software en los computadores, comparar precios de productos de vendedores visitando sus sitios web, el gusano (*worm*) de Xerox PARC.
- Son una **amenaza de seguridad** para los recursos locales.
- El **entorno debe decidir** a **qué recursos locales deja acceder** a un agente.
- Hay que **autenticar** al usuario (la identidad del propietario debe ir de forma segura en su código y datos).

Clientes ligeros



- Un cliente ligero tiene una **capa de software** que soporta básicamente la presentación o interfaz de la aplicación a la que accede.
- Los programas de aplicación a los que accede se ejecutan en un servidor remoto.
- No se descarga el código (lógica) de la aplicación en el computador local del cliente.
- Éste sólo ejecuta la interfaz de acceso al código que reside en el servidor remoto.
- Desventajas en, p.ej., actividades gráficas fuertemente interactivas por los retrasos.