Sistemas Distribuidos

Martín Silva

24 de noviembre de 2004

Capítulo 1

Introducción a los Sistemas Distribuidos

1.1. Introducción

Existen redes de computadores en cualquier parte. Una de ellas es Internet, como lo son las muchas redes de las que se compone. Las redes de teléfonos móviles, las redes corporativas, las de las empresas, los campus, las casas, redes dentro del coche, todas, tanto separadas como combinadas, comparten las características esenciales que las hacen elementos importantes para su estudio bajo el título de *sistemas distribuidos*.

> "Un sistema distribuido consiste en una colección de computadoras autónomas enlazadas por una red y equipadas con un sistema de software distribuido."[Tanenbaum 1995]

> "Definimos un sistema distribuido como aquel en el que los componentes hardware o software, localizados en computadores unidos mediante red, comunican y coordinan sus acciones sólo mediante paso de mensajes."[Coulouris 2001]

Las computadoras que están conectadas mediante una red pueden estar separadas espacialmente por cualquier distancia. Esta definición de sistemas distribuidos tiene las siguientes consecuencias significativas:

- *Concurrencia*: en una red de computadoras, la ejecución de programas concurrentes es la norma. Yo puedo realizar mi trabajo en mi computador, mientras tú realizas tu trabajo en la tuya, compartiendo recursos como páginas web o archivos, cuando es necesario.
- *Inexistencia de reloj global*: cuando los programas necesitan cooperar coordinan sus acciones mediante el intercambio de mensajes. La coordinación estrecha depende a menudo de una idea compartida del instante en el que ocurren las acciones de los programas. Pero resulta que hay límites a la precisión con lo que los computadores en una red pueden sincronizar sus relojes, no hay una única noción global del tiempo correcto. Esto es

una consecuencia directa del hecho que la única comunicación se realiza enviando mensajes a través de la red.

- *Fallos independientes*: Cada componente del sistema puede fallar independientemente, permitiendo que los demás continúen su ejecución.

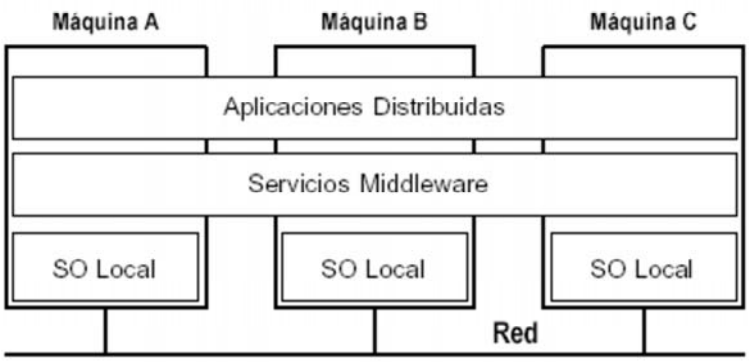


Figura 1.1: Un sistema distribuido organizado como *middleware*

1.2. Ejemplos

1.2.1. Internet

Internet es una vasta colección de redes de computadoras de diferentes tipos interconectados. La figura 1.2 muestra una porción típica de Internet. Programas ejecutándose en los computadores conectados a ella interactúan mediante paso de mensajes, empleando un medio común de comunicación. El diseño y la construcción de los mecanismos de comunicación Internet es una realización técnica fundamental, que permite que un programa que se está ejecutando en cualquier parte dirija mensajes a programas en cualquier otra parte.

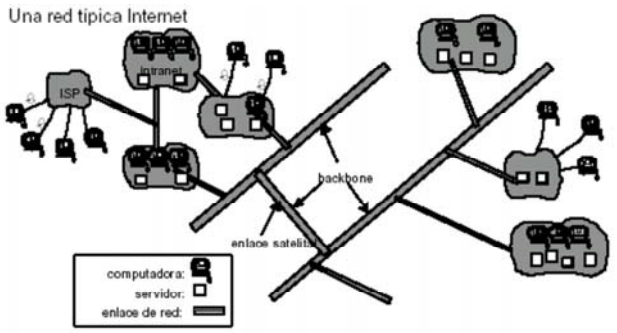


Figura 1.2: Una red típica Internet

Internet es también un sistema distribuido muy grande. En Internet hay disponibles servicios multimedia, que permiten a los usuarios el acceso a datos

de audio y video, incluyendo música, radio y canales de televisión y mantener videoconferencias. La capacidad de Internet para mantener los requisitos especiales de comunicación de los datos multimedia es actualmente bastante limitada porque no proporciona la infraestructura necesaria para reservar capacidad de la red para flujos individuales de datos.

1.2.2. Intranets

Una intranet es una porción de Internet que es, administrada separadamente y que tiene un límite que puede ser configurado para hacer cumplir políticas de seguridad local.

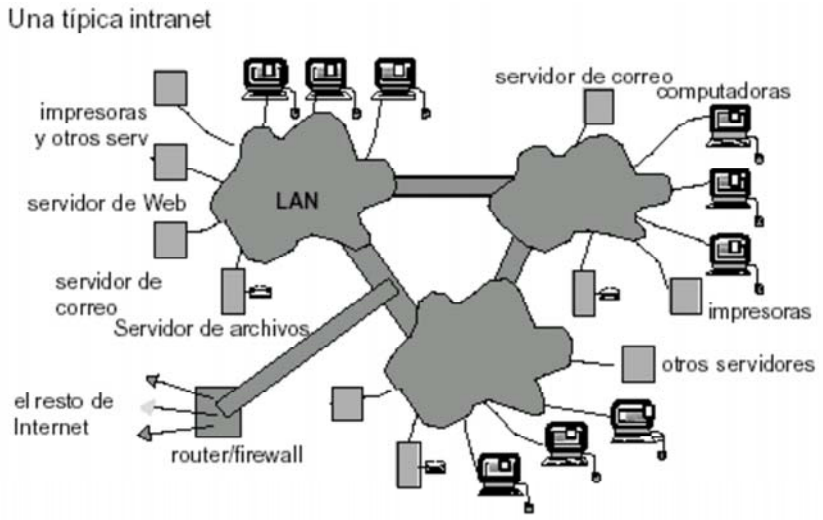


Figura 1.3: Una intranet típica

La figura 1.3 muestra una intranet típica. Está compuesta de varias redes de área local (LANs) enlazadas por conexiones backbone. La configuración de red de una intranet particular e responsabilidad de la organización que la administra y puede variar ampliamente, desde una LAN en un único sitio a un conjunto de LANs conectadas perteneciendo a ramas de la empresa u otra organización en diferentes países.

1.2.3. Computación móvil y ubicua

Los avances tecnológicos en la miniaturización de dispositivos y en redes inalámbricas han llevado cada vez más a la integración de dispositivos de computación pequeños y portátiles en sistemas distribuidos. Estos dispositivos incluyen:

- Computadores portátiles.
- Dispositivos de mano (*handheld*), entre los que se incluyen asistentes digitales personales (PDA), teléfonos móviles, buscapersonas y videocámaras o cámaras digitales.

- Dispositivos que se pueden llevar puestos, como relojes inteligentes con funcionalidad semejante a la de los PDAs.
- Dispositivos insertados en aparatos, como lavadoras, sistemas de alta fidelidad, coches y frigoríficos.

La facilidad de transporte de muchos de estos dispositivos, junto con su capacidad para conectarse adecuadamente a redes en diferentes lugares, hace posible la *computación móvil*. Se llama computación móvil (también llamada *computación nómada* ([Kleinrock 1997], www.cooltown.hp.com) a la realización de tareas de cómputo mientras el usuario está en movimiento o visitando otros lugares distintos de su entorno habitual.

Computación ubicua [Weiser 1993] es la utilización concertada de muchos dispositivos de computación pequeños y baratos que están presentes en los entornos físicos de los usuarios, incluyendo la casa, la oficina y otros. El término *ubicuo* está pensado para sugerir que los pequeños dispositivos llegarán a estar tan extendidos en los objetos de cada día que apenas nos daremos cuenta de ellos.

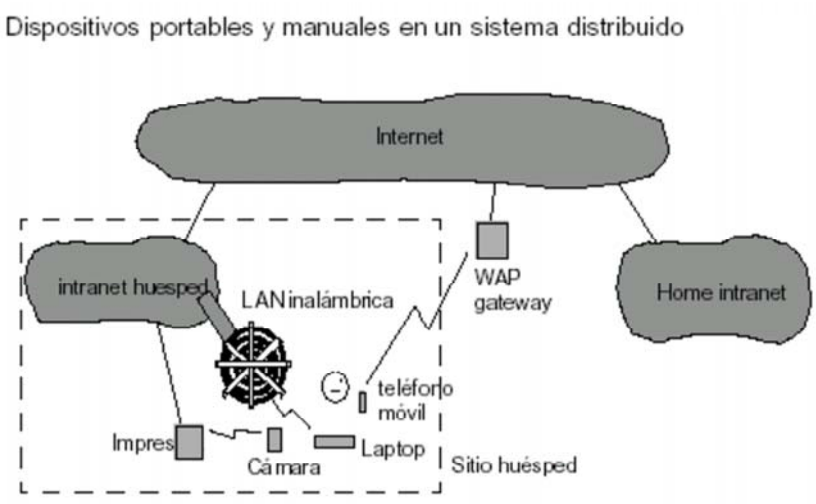


Figura 1.4: Dispositivos portátiles y de mano en un sistema distribuido

La figura 1.4 muestra a un usuario que está visitando una organización. En la figura se aprecia la intranet de la casa del usuario y la de la organización anfitriona en el lugar que está visitando dicho usuario. Ambas intranets están conectadas al resto de Internet.

1.2.4. Recursos compartidos y Web

Los usuarios están tan acostumbrados a los beneficios de compartir recursos que pueden pasar por alto su significado. Normalmente compartimos recursos hardware como impresoras, recursos de datos como archivos, y recursos con una funcionalidad más específica como máquinas de búsqueda.

Considerado desde el punto de vista de la provisión de hardware, se comparten equipos como impresoras y discos para reducir costes. Pero es mucho

más significativo para los usuarios compartir recursos de alto nivel que forman parte de sus aplicaciones y su trabajo habitual y sus actividades sociales. Por ejemplo, los usuarios están preocupados con compartir datos en forma de una base de datos compartida o un conjunto de páginas web, no los discos o los procesadores sobre los que están implementados.

El término *servidor* se refiere a un programa en ejecución (un *proceso*) en un computador en red que acepta peticiones de programas que se están ejecutando en otras computadoras para realizar un *servicio* y responder adecuadamente. Los procesos solicitantes son llamados *clientes*. Las peticiones se envían a través de mensajes desde los clientes al servidor y las contestaciones se envían mediante mensajes desde el servidor a los clientes. Cuando un cliente envía una petición para que se realice una operación, decimos que el cliente *invoca una operación* del servidor. Se llama *invocación remota* a una interacción completa entre un cliente y un servidor, desde el instante en el que el cliente envía su petición hasta que recibe la respuesta del servidor.

El mismo proceso puede ser tanto un cliente como un servidor, puesto que los servidores a veces invocan operaciones en otros servidores. Los términos *cliente* y *servidor* se aplican a los roles desempeñados en una única solicitud. Ambos son distintos, en muchos aspectos, los clientes son activos y los servidores pasivos, los servidores se están ejecutando continuamente, mientras que los clientes sólo lo hacen el tiempo que duran las aplicaciones de las que forman parte.

Hay que señalar que por defecto los términos *cliente* y *servidor* se refieren a *procesos* no a las computadoras en las que se ejecutan, aunque en lenguaje coloquial dichos términos se refieren también a las propias computadoras. En un sistema distribuido escrito en un lenguaje orientado a objetos, los recursos pueden ser encapsulados como obetos y accedidos por objeto clientes, en cuyo caso hablaremos de un *objeto cliente* que invoca un método en un *objeto servidor*.

Muchos sistemas distribuidos, aunque no todos, pueden ser construidos completamente en forma de clientes y servidores que interaccionan. El World Wide Web, el correo electrónico y las impresoras en red concuerdan con este modelo.

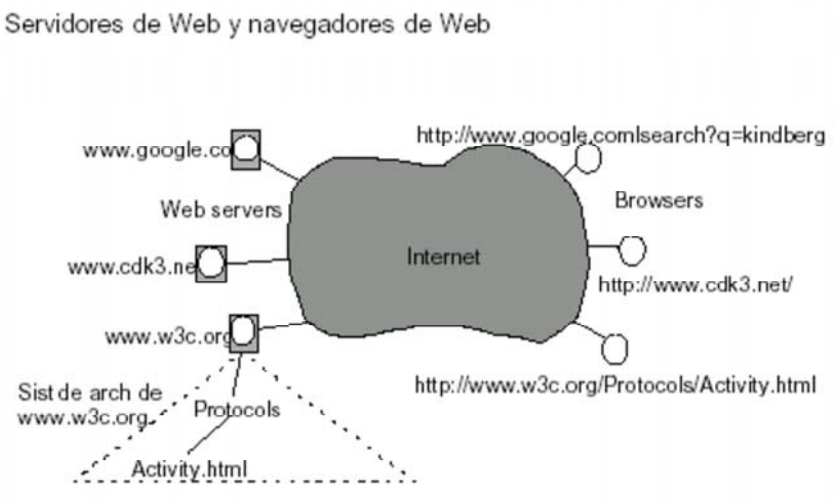


Figura 1.5: Servidores web y visualizadores web

Un navegador (*browser*) es un ejemplo de cliente. El navegador se comunica con el servidor web para solicitarle páginas. Ver figura 1.5.

1.3. Objetivos

1.3.1. Ventajas de los sistemas distribuidos frente a los centralizados

- *Economía*: Los microprocesadores ofrecen mejor proporción entre el precio y su rendimiento que los mainframes. Hace un cuarto de siglo, una persona experta en computadoras, Herb Grosch, enunció lo que se conocería después como la ley de Grosch: el poder de cómputo de CPU es proporcional al cuadrado de su precio. Si se paga el doble, se obtiene cuatro veces el desempeño. Con la tecnología del microprocesador, la ley de Grosch ya no es válida.
- *Velocidad*: Un sistema distribuido puede tener mayor poder de cómputo que un mainframe.
- *Distribución inherente*: Algunas aplicaciones utilizan máquinas que están separadas a cierta distancia.
- *Confiabilidad*: Si una máquina se descompone, el sistema puede sobrevivir como un todo.
- *Crecimiento por incrementos*: Se puede añadir poder de cómputo en pequeños incrementos.

1.3.2. Ventajas de los sistemas distribuidos sobre las computadoras aisladas

- *Datos compartidos*: Permiten que varios usuarios tengan acceso a una base de datos común.
- *Dispositivos compartidos*: Permiten que varios usuarios compartan periféricos caros, como las impresoras a color.
- *Comunicación*: Facilita la comunicación de persona a persona; por ejemplo, mediante correo electrónico.
- *Flexibilidad*: Difunde la carga de trabajo entre las máquinas disponibles en la forma más eficaz en cuanto a los costos.

1.3.3. Desventajas de los sistemas distribuidos

- *Software*: Existe poco software para los sistemas distribuidos en la actualidad.
- *Redes*: La red se puede saturar o causar otros problemas.
- *Seguridad*: Un acceso sencillo también se aplica a datos secretos.

1.4. Limitaciones que crean problemas tecnológicos en los SD

- No existe una memoria global, cada nodo tiene su memoria local.
- Establecer un estado global es complejo.
- No se puede asegurar un tiempo global.

1.5. Características y objetivos de diseño

Hay seis características claves responsables de la utilidad de los sistemas distribuidos: *compartir recursos, apertura, concurrencia, escalabilidad, tolerancia a las fallas y transparencia.* Debe notarse que no son consecuencias automáticas de la distribución; el software de sistema y de aplicación debe ser diseñado cuidadosamente con el objeto de asegurar que se alcancen.

1.5.1. Compartir recursos

El término *recurso* es un tanto abstracto, pero caracteriza al rango de cosas que pueden ser compartidas de forma útil en un sistema distribuido. El rango va desde componentes hardware hasta entidades software tales como archivos y bases de datos. Los beneficios del acceso compartido a un sistema de almacenamiento que contiene bases de datos, programas, documentación y otro tipo de información fueron reconocidos por primera vez con la aparición de sistemas de tiempo compartido multiusuarios a comienzos de los años 60 y en los sistemas UNIX multiusuarios en la década de los 70.

Para compartir en forma efectiva cada recurso debe ser administrado por un progama que ofrezca una interfaz de comunicación que habilite el recurso a ser accedido, manipulado y actualizado en forma confiable y consistente. El término genérico *administrador de recursos* se usa a veces para designar un módulo software que administra a un conjunto de recursos de un tipo particular. Podemos desarrollar esta perspectiva para derivar en dos modelos muy interesantes de sistemas distribuidos:

- *El modelo cliente-servidor*: Que es actualmente el modelo más conocido y más adoptado para sistemas distribuidos, en el que hay un conjunto de procesos servidores, cada uno actuando como administradores de recursos para una colección de recursos de un tipo determinado y una colección de procesos cliente cada uno realizando una tarea que requiere acceso a algún tipo de recurso compartido.
- *El modelo basado en objetos*: Que no es muy distinto al modelo de programación orientada a objetos en la cual cada entidad en un programa que está ejecutando se ve como un objeto con una interfaz de manejo de mensajes que provee acceso a sus operaciones [Wegner 1984, Myer 1988]. Usaremos el término *administrador de objetos* para referirnos a la colección de procedimientos y valores de datos que en conjunto caracterizan a una clase de objetos. Hay una correspondencia entre estos y el administrador de recursos del modelo cliente-servidor.

La implementación del modelo basado en objetos trae aparejado la aparición de algunos problemas. Requiere que sea ubicado un administrador de objetos del tipo relevante cada vez que se ubica un objeto, porque los objetos contienen una representación de su estado, y se debe encontrar un administrador cada vez que se ubica un objeto para acceder a su estado. Esto no presenta problemas cuando los objetos no pueden moverse y se han construido varios sistemas distribuidos diseñados para soportar el direccionamiento uniforme de objetos. Por ejemplo Argus [Liskov 1988], Amoeba [Tanenbaum 1990] y Mach. Hasta la fecha el modelo de objetos distribuidos más general en el cual los objetos pueden migrar libremente ha sido implementado sólo en sistemas experimentales tales como Arjuna, Clouds y Emerald.

1.5.2. Apertura o Extensibilidad

La apertura de un sistema de computadoras es la característica que determina que el sistema sea extendido en diferentes maneras. Un sistema puede ser abierto o cerrado con respecto a extensiones de hardware - por ejemplo, el agregado de periféricos, memoria o interfaces de comunicación - o con respecto a la extensión de software - el agregado de facilidades del sistema operativo, protocolos de comunicación y servicios para compartir recursos. La apertura de los sistemas distribuidos está determinada primariamente por el grado de que nuevos servicios de compartición de recursos puedan ser agregados sin la interrupción o duplicación de servicios existentes.

La apertura se logra especificando y documentando las interfaces claves de software de un sistema y dejándolas disponibles a los desarrolladores de software, es decir, *publicándolas.*

1.5.3. Concurrencia

Cuando existen varios procesos en una computadora decimos que se están ejecutando *concurrentemente*, ya que cada proceso sólo existe mientras que dure la ejecución de un programa, la coexistencia implica concurrencia de ejecución. Si la computadora está equipada con un sólo procesador central esto se logra intercalando la ejecución de porciones de cada proceso. Si la computadora tiene N procesadores, entonces hasta N procesos pueden ejecutarse simultáneamente, es decir, en paralelo, alcanzando una mejora de N-veces el rendimiento computacional.

En los sistemas distribuidos hay varias computadoras, cada una con uno o más procesadores. Si hay M computadoras en un sistema distribuido con un procesador central en cada una, entonces hasta M procesos pueden correr en paralelo, suponiendo que los procesos estén ubicados en computadoras diferentes.

1.5.4. Escalabilidad

Los sistemas distribuidos operan efectiva y eficientemente en muchas escalas diferentes. Se dice que un sistema es *escalable* si conserva su efectividad cuando ocurre un incremento significativo en el número de recursos y el número de usuarios.

El sistema distribuido más chico probablemente consiste de dos estaciones de trabajo y un servidor de archivos, mientras que un sistema distribuido construido sobre una única red de área local puede contener varios cientos de estaciones de trabajo y varios sistemas de archivos, servidores de impresión y otros servidores de propósito especial. Varias redes de área local a menudo están conectadas formando una interred (**internetwork**), y estas pueden contener varios miles de computadoras que forman un único sistema distribuido, permitiendo que sean compartidos los recursos entre todas ellas.

El software de sistema y de aplicación no debería necesitar cambiar cuando la escala del sistema incremente. Esta característica se alcanza hasta un nivel significativo en muchos sistemas distribuidos actuales, pero es un área en la cual se necesita más investigación.

Fecha	Computadoras	Servidores de Web
1979, Dic	188	0
1989, Jul	130.000	0
1999, Jul	56.218.000	5.560.866

Cuadro 1.1: Computadoras en Internet

Fechas	Computadoras	Servidores Web	Porcentaje
1993, Jul	1.776.000	130	0,008
1995, Jul	6.642.000	23.500	0,4
1997, Jul	19.540.000	1.203.096	6
1999, Jul	56.218.000	6.598.697	12

Cuadro 1.2: Computadoras vs servidores de Web en Internet

Internet proporciona un ejemplo de un sistema distribuido en el que el número de computadoras y servicios experimenta un dramático incremento. El cuadro 1.1 muestra el número de computadoras y servicios en Internet durante 20 años, desde 1979 hasta 1999, y el cuadro 1.2 muestra el creciente número de computadoras y servidores *web* durante los 16 años de historia del Web hasta 1999.

El diseño de los sistemas distribuidos escalables presenta los siguientes retos:

- *Control del coste de los recursos físicos*: según crece la demanda de un recurso, debiera ser posible extender el sistema, a un coste razonable, para satisfacerla.
- *Control de las pérdidas de prestaciones*: los algoritmos que emplean estructuras jerárquicas se comportan mejor frente al crecimiento de la escala que los algoritmos que emplean estructuras lineales. Pero incluso con estructuras jerárquicas un incremento en tamaño traerá consigo pérdidas en prestaciones: el tiempo que lleva acceder a datos estructurados jerárquicamente es $O(log\ n)$, donde n es el tamaño del conjunto de datos. Para que un sistma sea escalable, la máxima pérdida de prestaciones no debiera ser peor que esta medida.
- *Prevención de desbordamiento de recursos software*: por ejemplo el problema del agotamiento de direcciones IPv4. Es difícil predecir la demanda

que tendrá que soportar un sistema con años de anticipación. Además, sobredimensionar para prever el crecimiento futuro pudiera ser peor que la adaptación a un cambio cuando se hace necesario.

- *Evitación de cuellos de botella de prestaciones*: en general, para evitar cuellos de botella de prestaciones, los algoritmos deberían ser descentralizados. Por ejemplo, antes de la aparición del DNS, la tabla de nombres se alojaba en un solo archivo maestro que podía descargarse a cualquier computadora que la necesitara. Esto funcionaba bien cuando sólo había unos cientos de computadoras en Internet, pero pronto se convirtió en un serio cuello de botella de prestaciones y de administración.

1.5.5. Tolerancia a las fallas

Los sistemas de computación a veces fallan. Cuando ocurre una falla en el hardware o en el software, los programas pueden producir resultados incorrectos o pueden detenerse antes de que hayan completado.

El diseño de los sistemas de computación tolerantes a fallas están basados en dos enfoques:

- *redundancia de hardware*: puede lograrse que los servicios toleren fallos mediante el empleo redundante de componentes;
- *recuperación de software*: el diseño de programas que se recuperan ante los fallos.

1.5.6. Transparencia

La transparencia se define como la ocultación al usuario y al programador de aplicaciones de la separación de los componentes en un sistema distribuido, de manera tal que el sistema es percibido como un todo en vez de una colección de componentes independientes. Las implicaciones de la transparencia son una influencia importante en el diseño del sistema de software.

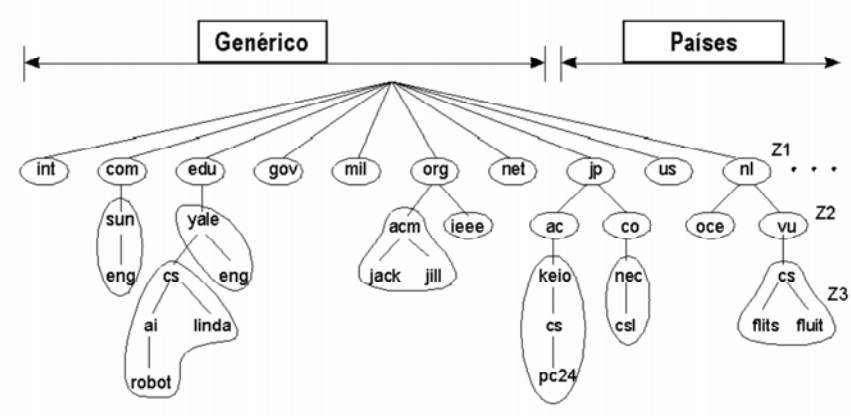
El Manual de Referencia ANSA (ANSA Reference Manual) y el Modelo de Referencia para el Procesamiento Distribuido Abierto (RM-ODP: Reference Model for Open Distributed Processing) de la Organización Internacional de Estándares identifican ocho formas de transparencia:

1. *Transparencia de acceso*: que permite acceder a los recursos locales y remotos empleando operaciones idénticas. Esconde diferencias en la representación de datos y cómo un recurso es accedido.
2. *Transparencia de ubicación*: que permite acceder a los recursos sin conocer su localización. Esconde la ubicación del recurso.
3. *Transparencia de concurrencia*: que permite que varios procesos operen concurrentemente sobre recursos compartidos sin interferencia mutua. Esconde que un recurso pueda ser compartido por varios usuarios competidores.
4. *Transparencia de replicación*: que permite utilizar múltiples ejemplares de cada recurso para aumentar la fiabilidad y las prestaciones sin que los

usuarios y los programadores de aplicaciones necesiten su conocimiento. Esconde desde donde es utilizado un recurso compartido por varios usuarios competidores.

5. *Transparencia frente a fallos*: que permite ocultar los fallos, dejando que los usuarios y programas de aplicación completen sus tareas a pesar de fallos del hardware o de los componentes software. Esconde la falla y recuperación de un recurso.
6. *Transparencia de movilidad*: que permite la reubicación de recursos y clientes en un sistema sin afectar la operación de los usuarios y los programas.
7. *Transparencia de prestaciones*: que permite reconfigurar el sistema para mejorar las prestaciones según varía su carga.
8. *Transparencia al escalado*: que permite al sistema y a las aplicaciones expandirse en tamaño sin cambiar la estructura del sistema o los algoritmos de aplicación. Algunos ejemplos de las limitaciones de la escalabilidad:
   - Servicios centralizados: Un único servidor para todos los usuarios.
   - Datos centralizados: Una sola guía telefónica en línea.
   - Algoritmos centralizados: Ruteo basado en información completa.

Figura 1.6: Ejemplo de técnica de escalamiento: dividir el espacio de nombres DNS en zonas:



Las dos más importantes son la transparencia de acceso y la transparencia de ubicación; su presencia o ausencia afecta principalmente a la utilización de recursos distribuidos. A veces se les da el nombre conjunto de *transparencia de red.*

1.6. Conceptos de hardware

Aunque todos los sistemas distribuidos constan de varios CPU, existen diversas formas de organizar el hardware; en particular, en la forma de interconectarlos y comunicarse entre sí.

Con el paso de los años, se han propuesto diversos esquemas de clasificación para los sistemas de cómputo con varios CPU, pero ninguno de ellos ha tenido un éxito completo ni se ha adoptado de manera amplia. Es probable que la taxonomía más citada se la de Flynn (1972), aunque es algo rudimentaria. Flynn eligió dos características consideradas por él como esenciales: el número de flujos de instrucciones y el número de flujos de datos. Una computadora con un flujo de instrucciones y uno de datos se llama SISD (Single Instruction, Single Data). Todas las computadoras tradicionales de un procesador (es decir, aquellas que tienen un CPU) caen dentro de esta categoría, desde las computadoras personales hasta los grandes mainframes.

La siguiente categoría es SIMD (Single Instruction, Multiple Data), con un flujo de instrucciones y varios flujos de datos. Este tipo se refiere a ordenar procesadores con unidad de instrucción que busca una instrucción y después instruye a varias unidades de datos para que la lleven a cabo en paralelo, cada una con sus propios datos. Estas máquinas son útiles para los cómputos que repiten los mismos cálculos en varios conjuntos de datos, por ejemplo, sumando todos los elelementos de 64 vectores independientes. Ciertas computadoras son SIMD.

La siguiente categoría es MISD (Multiple Instruction, Single Data), con un flujo de varias instrucciones y un flujo de datos. Ninguna de las computadoras conocidas se ajusta a este modelo. Por último, está MIMD (Multiple Instruction, Multiple Data), que significa un grupo de computadoras independientes, cada una con su propio contador de programa y datos. Todos los sistemas distribuidos son MIMD, por lo que este sistema de clasificación no es muy útil para nuestros fines.

Aunque Flynn se detuvo en este punto, nosotros ([Tanenbaum 1995]) avanzaremos un poco más. Dividimos todas las computadoras MIMD en dos grupos: aquellas que tienen memoria compartida, que por lo general se llaman **multiprocesadores** y aquellas que no, que a veces reciben el nombre de **multicomputadoras**. La diferencia esencial es ésta: en un multiprocesador, existe un espacio de direcciones virtuales, compartido por todos los CPU. Por ejemplo, si algún CPU escribe el valor 44 en la dirección 1000, cualquier otro CPU que haga una lectura posterior de *su* dirección 1000 obtendrá el valor 44. Todas las máquinas comparten la misma memoria.

En contraste, en una multicomputadora, cada máquina tiene su propia memoria. Si un CPU escribe el valor 44 en la dirección 1000 y otro CPU lee su dirección 1000, obtendrá el valor que se encontraba ahí antes. La escritura de 44 no afecta *su* memoria de manera alguna. Un ejemplo común de multicomputadora es una colección de computadoras personales conectadas mediante una red.

Cada una de estas categorías se puede subdividir, con base en la arquitectura de la red de interconexión. Describimos esas dos categorías como **bus o canal**, y con **conmutador** (ver figura 1.7). En la primera queremos indicar que existe una red, plano de base, bus, cable u otro medio que conecta todas las máquinas.

Los sistemas **con conmutador** no tienen sólo una columna vertebral como en la televisión por cable, sino que tienen cables individuales de una máquina a otra y utilizan varios patrones diferentes de cableado. Los mensajes se mueven a través de los cables y se toma una decisión explícita de conmutación en cada etapa, para dirigir el mensaje a lo largo de uno de los cables de salida. El sistema mundial de teléfonos públicos está organizado de esta manera.

Otra división de nuestra taxonomía es que, en ciertos sistemas, las máquinas

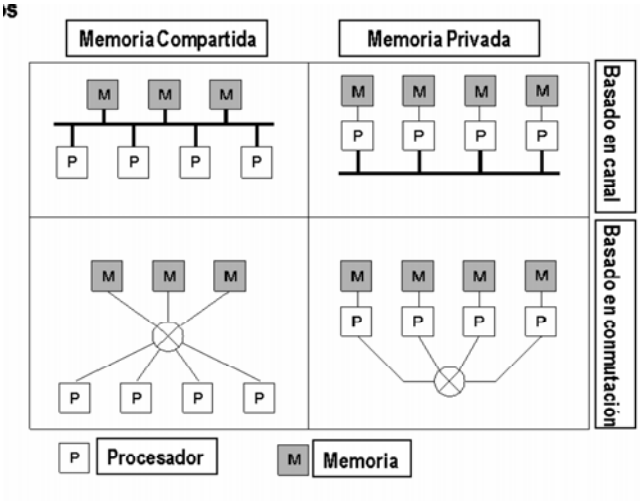


Figura 1.7: Diferentes organizaciones básicas y memoria en sistemas de cómputo distribuidos

están **fuertemente acopladas** y en otras están **débilmente acopladas**. En un sistema fuertemente acoplado, el retraso que se experimenta al enviar un mensaje de una computadora a otra es corto y la tasa de transmisión de los datos, es decir, el número de bits por segundo que se pueden transferir, es alta. En un sistema débilmente acoplado ocurre lo contrario: el retraso de los mensajes entre las máquinas es grande y la tasa de transmisión de los datos es baja. Por ejemplo, es probable que dos circuitos de CPU de la misma tarjeta de circuito impreso, conectados mediante cables insertados en la tarjeta, estén fuertemente acoplados, mientras que dos computadoras conectadas mediante un módem de 2400 bits/seg a través del sistema telefónico están débilmente acopladas.

Los sistemas fuertemente acoplados tienden a utilizarse más como sistemas paralelos (para trabajar con un problema) y los débilmente acoplados tienden a utilizarse como sistemas distribuidos (para trabajar con varios problemas no relacionados entre sí), aunque esto no siempre es cierto. Un contraejemplo famoso es un proyecto en el que cientos de computadoras en todo el mundo trabajaron en forma conjunta para factorizar un enorme número de cerca de 100 dígitos. A cada computadora se le asignó un límite distinto de divisores para su análisis y todas ellas trabajaron en el problema durante su tiempo correspondiente y reportaban sus resultados mediante el correo electrónico.

En general, los multiprocesadores tienden a estar más fuertemente acoplados que las multicomputadoras, puesto que pueden intercambiar datos a la velocidad de sus memorias, pero algunas multicomputadoras basadas en fibras ópticas pueden funcionar también con velocidad de memoria.

1.6.1. Multiprocesadores con base en canal

Los multiprocesadores con base en canales (buses) constan de cierta cantidad de CPU, conectados en un canal común, junto con un módulo de memoria. Una configuración sencilla consta de un plano de base (*backplane*) de alta velocidad o tarjeta madre, en el cual se pueden insertar las tarjetas de memoria y el CPU.

Un canal típico tiene 32 o 64 líneas de direcciones, 32 o 64 líneas de datos y 32 o más líneas de control, todo lo cual opera en paralelo. Para leer una palabra de memoria, un CPU coloca la dirección de la palabra deseada en las líneas de direcciones del canal y coloca una señal en las líneas de control adecuadas para indicar que desea leer. La memoria responde y coloca el valor de la palabra en las líneas de datos para permitir la lectura de ésta por parte del CPU solicitante. La escritura funciona de manera similar.

Puesto que sólo existe una memoria, si el CPU A escribe una palabra en la memoria y después el CPU B lee esa palabra un microsegundo después, B obtendrá el valor recién escrito. Una memoria con esta propiedad es **coherente**. La coherencia juega un papel importante en los sistemas operativos distribuidos en una variedad de formas que veremos más adelante.

El problema con este esquema es que si sólo se dispone de 4 o 5 CPUs, el canal estará por lo general sobrecargado y el rendimiento disminuirá en forma drástica. La solución es añadir una **memoria caché** de alta velocidad entre el CPU y el canal, como se muestra en la figura 1.8. El caché guarda las palabras de acceso reciente. Todas las solicitudes de la memoria pasan a través del caché. Si la palabra solicitada se encuentra en el caché, éste responde al CPU y no se hace solicitud alguna al canal. Si el caché es lo bastante grande, la probabilidad de éxito (la **tasa de encuentros**) será alta y la cantidad de tráfico en el canal por cada CPU disminuirá en forma drástica, lo que permite un número mayor de CPUs en el sistema. Los tamaños comunes del caché van desde los 64K hasta 1M, lo que da como resultado una tasa de encuentros del 90 % o más.

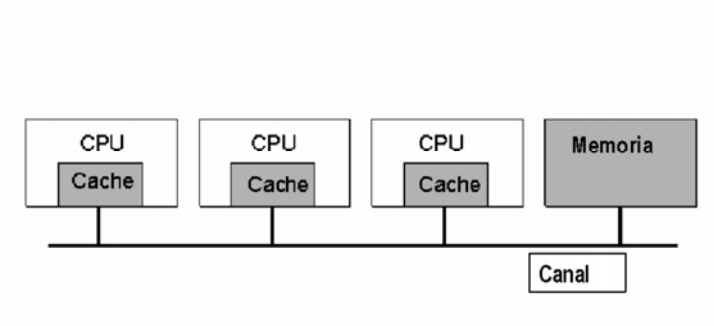


Figura 1.8: Multiprocesador basado en canal

1.6.2. Multiprocesadores con conmutador

Para construir un multiprocesador con más de 64 procesadores, es necesario un método distinto para conectar cada CPU con la memoria. Una posibilidad es dividir la memoria en módulos y conectarlos a las CPU con un **conmutador de cruceta**, como se muestra en la figura 1.9 (a). Cada CPU y cada memoria tiene una conexión que sale de él, como se muestra en la figura. En cada intersección está un delgado **conmutador de punto de cruce** electrónico que el hardware puede abrir y cerrar. Cuando un CPU desea tener acceso a una memoria particular, el conmutador del punto de cruce que los conecta se cierra de manera momentánea, para permitir dicho acceso. La virtud del conmutador de cruceta es que muchos CPU pueden tener acceso a la memoria al mismo tiempo, aunque

si dos CPU intentan tener acceso a la misma memoria en forma simultánea, uno de ellos deberá esperar.

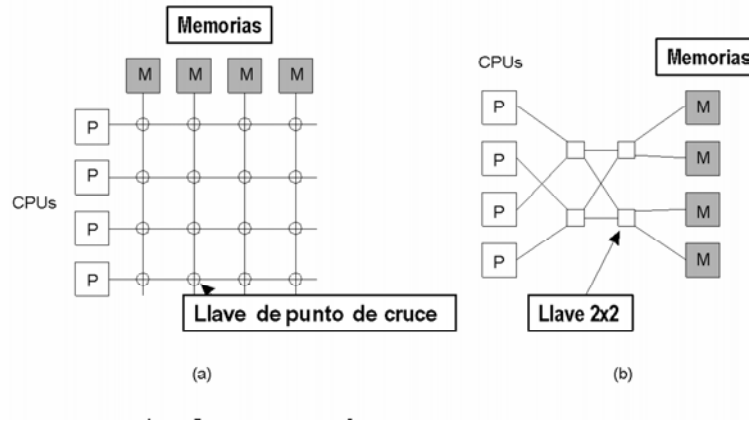


Figura 1.9: (a) Conmutador de cruceta (b) Red omega

La desventaja del conmutador de cruceta es que con n CPU y n memorias, se necesitan n^2 conmutadores en los puntos de cruce. Si n es grande, este número puede ser prohibitivo. Como resultado, las personas han buscado y encontrado otras redes de conmutación que necesiten menos conmutadores. La **red omega** de la figura 1.9 (b) es un ejemplo. Esta red contiene conmutadores 2x2, cada uno de los cuales tiene dos entradas y dos salidas. Cada conmutador puede dirigir cualquiera de las entradas en cualquiera de las salidas. Un análisis cuidadoso de la figura mostrará que si se eligen los estados adecuados de los conmutadores, cada CPU podrá tener acceso a cada memoria. Estos conmutadores se pueden activar en cuestión de nanosegundos.

En el caso general, con n CPU y n memorias, la red omega necesita $\log_2 n$ etapas de conmutación, cada una de las cuales tiene $\frac{n}{2}$ conmutadores, para un total de $\frac{n \log_2 n}{2}$ conmutadores. Aunque este número es mejor que n^2 , sigue siendo considerable.

En resumen, los multiprocesadores basados en canales, incluso con cachés monitores, quedan limitados a lo más a 64 CPUs por la capacidad del canal. Para rebasar estos límites, es necesaria una red con conmutador, como uno de cruceta, una red omega o algo similar. Los grandes conmutadores de cruceta y las grandes redes omega son muy caros y lentos. Las máquinas NUMA necesitan complejos algoritmos para la buena colocación del software. La conclusión es clara: la construcción de un multiprocesador grande, fuertemente acoplado y con memoria compartida es difícil y cara.

1.6.3. Multicomputadoras con base en canales

Por otro lado, la construcción de una multicomputadora (es decir, sin memoria compartida) es fácil. Cada CPU tiene conexión directa con su propia memoria local. El único problema restante es la forma en que las CPUs se comunicarán entre sí. Es claro que aquí también se necesita cierto esquema de interconexión, pero como sólo es para la comunicación entre un CPU y otro, el volumen

del tráfico será de varios órdenes menor en relación con el uso de una red de interconexión para el tráfico CPU-memoria.

En la figura XX vemos una multicomputadora con base en un canal. Es similar, desde el punto de vista topológico, al multiprocesador basado en canal, pero como tendrá menor tráfico, no necesita ser un canal con un plano de base de alta velocidad. Puede ser una LAN de menor velocidad (por lo general de 10-100 Mb/seg, en comparación con 300 Mb/seg o más para un bus con un plano de base).

1.6.4. Multicomputadoras con conmutador

Nuestra última categoría es la de las multicomputadoras con conmutador. Se han propuesto y construido varias redes de interconexión, pero todas tienen la propiedad de que cada CPU tiene acceso directo y exclusivo a su propia memoria particular. La figura 1.10 muestra dos topologías populares, una retícula y un hipercubo. Las retículas son fáciles de comprender y se basan en las tarjetas de circuitos impresos. Se adecuan mejor a problemas con naturaleza bidimensional inherente, como la teoría de gráficas o la visión (por ejemplo, los ojos de un robot o el análisis de fotografías).

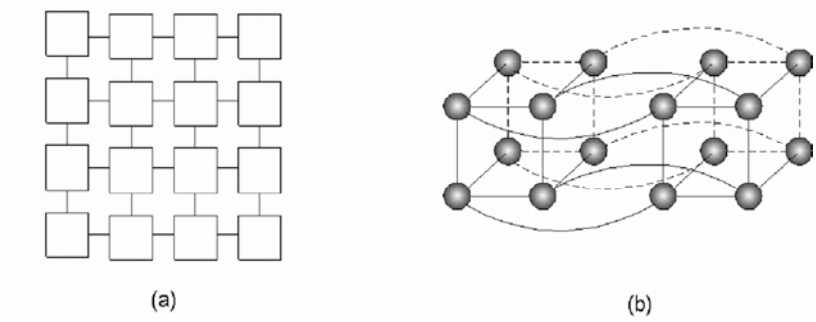


Figura 1.10: (a) Retícula (b) Hipercubo

Un **hipercubo** es un cubo n-dimensional. El hipercubo de la figura 1.10 es de dimensión 4. Se puede pensar como dos cubos ordinarios, cada uno de los cuales cuenta con 8 vértices y 12 aristas. Cada vértice es un CPU. Cada arista es una conexión entre dos CPU. Se conectan los vértices correspondientes de cada uno de los cubos.

1.7. Conceptos de software

Aunque el hardware es importante, el software lo es más. La imagen que presenta y la forma de pensar de los usuarios de un sistema, queda determinada en gran medida por el software del sistema operativo, no por el hardware.

Es posible distinguir dos tipos de sistemas operativos para los de varios CPU: los **débilmente acoplados** y los **fuertemente acoplados**.

El software débilmente acoplado permite que las máquinas y los usuarios de un sistema distribuido sean independientes entre sí en lo fundamental, pero que interactúen en cierto grado cuando sea necesario. Consideremos un grupo

de computadoras personales, cada una de las cuales tiene su propio CPU, su propia memoria, su propio disco duro y su propio sistema operativo, pero que comparten ciertos recursos, como las impresoras láser y las bases de datos en una LAN. Este sistema está débilmente acoplado, puesto que las máquinas individuales se distinguen con claridad, cada una de las cuales tiene su propio trabajo por realizar. Si la red falla por alguna razón, las máquinas individuales continúan su ejecución en cierto grado considerable, aunque se puede perder cierta funcionalidad (por ejemplo, la capacidad de imprimir archivos).

Esta distinción entre sistemas débilmente acoplados y fuertemente acoplados está relacionado con la clasificación de hardware dada en la sección previa. Un sistema operativo fuertemente acoplado se lo refiere generalmente como **sistema operativo distribuido** (SOD), y se usa para administrar multprocesadores y multicomputadoras homogéneas. Como los sistemas operativos de uniprocesadores tradicionales, el objetivo principal de un sistema operativo distribuido es esconder las particularidades de la administración del hardware subyacente de manera tal que pueda ser compartido por proceso múltiples.

El **sistema operativo de red** (SOR) débilmente acoplado se usa para sistemas de multicomputadoras heterogéneas. A pesar de que la administración del hardware subyacente es un aspecto importante de un SOR, la distinción de los sistemas operativos tradicionales viene del hecho que los servicios locales se hacen disponibles a clientes remotos.

Para llegar a un sistema distribuido, son necesarias mejoras a los servicios de los sistemas operativos de red de manera tal que se provea un mejor soporte para la transparencia de distribución. Estas mejoras llevan a lo que se conoce como **middleware**, y yacen en el corazón de los sistemas distribuidos modernos.

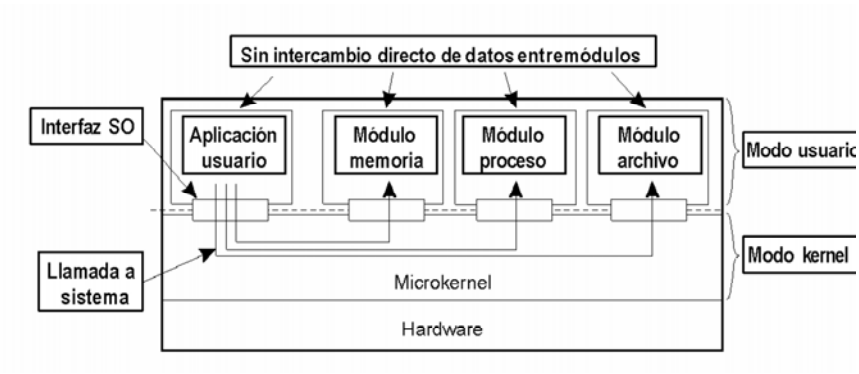


Figura 1.11: Sistema operativo uniprocesador

1.7.1. Sistemas operativos de redes

Comenzaremos con el software débilmente acoplado en hardware débilmente acoplado, puesto que tal vez ésta sea la combinación más común en muchas organizaciones. Un ejemplo típico es una red de estaciones de trabajo de ingeniería conectadas mediante una LAN. En este modelo, cada usuario tiene una estación de trabajo para su uso exclusivo. Puede o no tener un disco duro. En definitiva, tiene su propio sistema operativo. Lo normal es que todos los comandos se

ejecuten en forma local, justo en la estación de trabajo.

Sin embargo, a veces es posible que un usuario se conecte de manera remota con otra estación de trabajo mediante un comando como

rlogin machine

El efecto de este comando es convertir la propia estación de trabajo del usuario en una terminal enlazada con la máquina remota. Los comandos escritos en el teclado se envían a la máquina remota y la salida de la máquina remota se exhibe en la pantalla.

Las redes de estaciones de trabajo también tienen un comando de copiado remoto para copiar archivos de una máquina a otra. Por ejemplo, un comando como

rcp machine1:file1 machine2:file2

copiaría el archivo *file1* de *machine1* a *machine2*, con el nombre de *file2*. De nuevo, el movimiento de los archivos es explícito y se requiere que el usuario esté por completo consciente de la posición de todos los archivos y el sitio donde se ejecutan todos los comandos.

El sistema operativo a utilizar en este tipo de ambiente debe controlar las estaciones de trabajo en lo individual, a los servidores de archivo y también debe encargarse de la comunicación entre ellos. Es posible que todas las máquinas ejecuten el mismo sistema operativo, pero esto no es necesario. Si los clientes y los servidores ejecutan diversos sistemas, entonces, como mínimo, deben coincidir en el formato y significado de todos los mensajes que podrían intercambiar. En una situación como ésta, en la que cada máquina tiene un alto grado de autonomía y existen pocos requisitos a lo largo de todo el sistema, las personas se refieren a ella como un **sistema operativo de red**.

1.7.2. Sistemas realmente distribuidos

El siguiente paso en la evolución es el del software fuertemente acoplado en hardware débilmente acoplado (es decir, en multicomputadoras). El objetivo de un sistema de este tipo es crear la ilusión en las mentes de los usuarios que toda la red de computadoras es un sistema de tiempo compartido, en vez de una colección de máquinas diversas. Algunos autores se refieren a esta propiedad como la **imagen de único sistema**. Otros tienen un punto de vista diferente y dicen que un sistema distribuido es aquel que se ejecuta en una colección de máquinas enlazadas mediante una red pero que actúan como un **uniprocesador virtual**. No importa la forma en que se exprese, la idea esencial es que los usuarios no deben ser conscientes de la existencia de varios CPUs en el sistema.

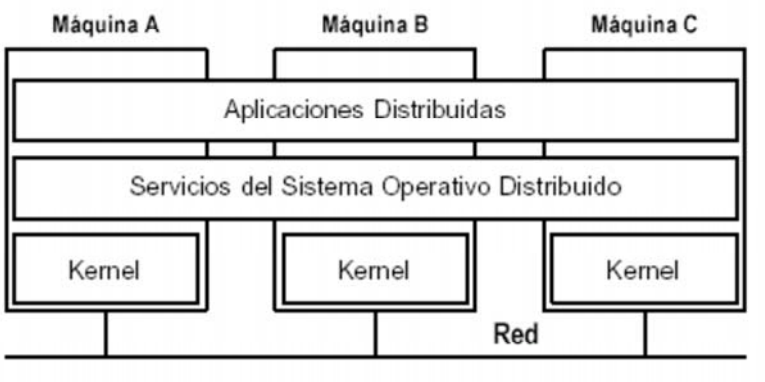


Figura 1.12: Sistemas Operativos Multicomputadora

1.7.2.1. Sistemas operativos uniprocesadores

1.7.2.2. Sistemas operativos multiprocesadores

1.7.2.3. Sistemas operativos multicomputadoras

1.7.2.4. Sistemas de memoria compartida distribuida

1.7.3. Middleware

1.7.3.1. Posición del middleware

1.7.3.2. Middleware y apertura

1.7.3.3. Comparación entre sistemas

1.8. Modelos de sistemas

1.8.1. Capas de software

La figura 1.13 muestra términos importantes como *plataforma* y *middleware*, que definimos a continuación:

> **Plataforma**: el nivel de hardware y las capas más bajas de software se denominan, a menudo, plataforma para sistemas distribuidos y aplicaciones. Estas capas más bajas proporcionan servicios a las que están por encima de ellas, y que son implementadas independientemente en cada computador, proporcionando una interfaz de programación del sistema a un nivel que facilita la comunicación y coordinación entre procesos. Windows para Intel x86, Sun OS para Sun SPARC, Solaris para Intel x86, Mac OS para Power PC son los principales ejemplos.[Coulouris 2001]
>
> **Middleware**: Es una capa de software cuyo propósito es enmascarar la heterogeneidad y proporcionar un modelo de programación conveniente para los programadores de aplicaciones. Se representa mediante procesos u objetos en un conjunto de computadoras que interactúan entre sí para implementar mecanismos de comunicación y de recursos compartidos para aplicaciones distribuidas.

> El middleware se ocupa de proporcionar bloques útiles para la construcción de componentes software que puedan trabajar con otros en un sistema distribuido. En particular, mejora el nivel de las actividades de comunicación de los programas de aplicación soportando abstracciones como: procedimiento de invocación remota, comunicación entre un grupo de procesos, notificación de eventos, replicación de datos compartidos y transmisión de datos multimedia en tiempo-real.[Coulouris 2001]

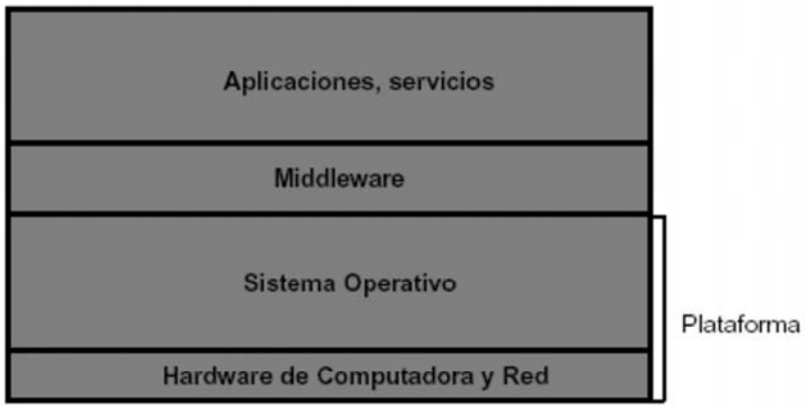


Figura 1.13: Capas de Software y Hardware

1.8.2. Modelo cliente-servidor

Es la arquitectura que se cita más a menudo cuando se discuten sistemas distribuidos. La figura 1.14 muestra la sencilla estructura sobre la que interaccionan los procesos clientes con los procesos servidores individuales, en computadores separados, con el fin de acceder a los recursos compartidos que ellos gestionan. La figura muestra la disposición de los procesos (elipses) en las computadoras (cajas grises). Utilizamos los términos *invocación* y *resultados* para etiquetar los mensajes, aunque se podrían haber etiquetado igualmente como *solicitud* y *respuesta*.

Los servidores pueden, a su vez, ser clientes de otros servidores, como se indica en la figura.

1.8.3. Servicios proporcionados por múltiples servidores

Los servicios pueden implementarse como distintos procesos de servidor en computadores separados interaccionando, cuando es necesario, para proporcionar un servicio a los procesos clientes (véase la figura 1.15). Los servidores pueden dividir el conjunto de objetos en los que está basado el servicio y distribuírselos entre ellos mismos, o pueden mantener copias replicadas de ellos en varias máquinas.

La replicación se utiliza para aumentar las prestaciones y disponibilidad y para mejorar la tolerancia a fallos. Proporciona múltiples copias consistentes de datos en procesos que se ejecutan en diferentes computadoras.

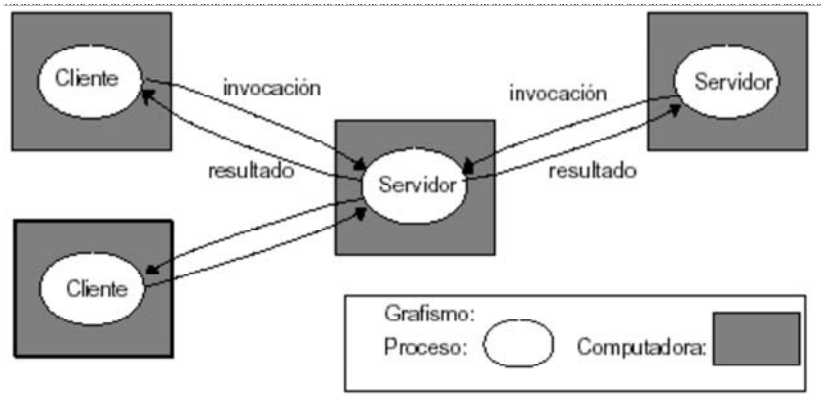


Figura 1.14: Modelo Cliente-Servidor

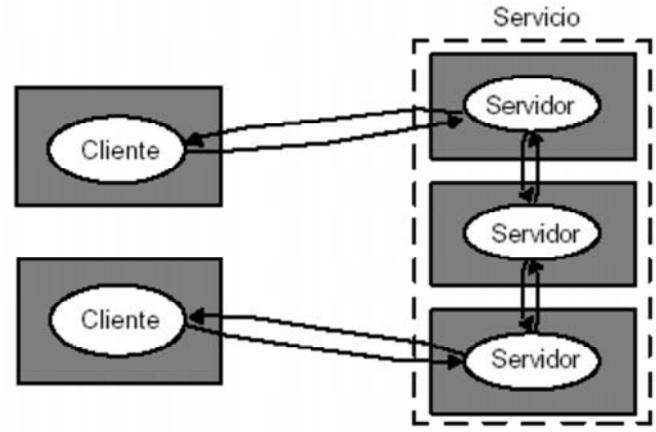


Figura 1.15: Servicio provisto por múltiples servidores

1.8.4. Servidores proxy y cachés

Una caché es un almacén de objetos de datos utilizados recientemente, y que se encuentra más próximo que los objetos en sí. Al recibir un objeto nuevo en una computadora se añade al almacén de la caché, reemplazando, si fuera necesario, algunos objetos existentes. Cuando se necesita un objeto en un proceso cliente, el servicio caché comprueba inicialmente la caché y le proporciona el objeto de una copia actualizada. Si no, se buscará una copia actualizada. Las cachés pueden estar ubicadas en cada cliente o en un servidor proxy que puede compartirse desde varios clientes.

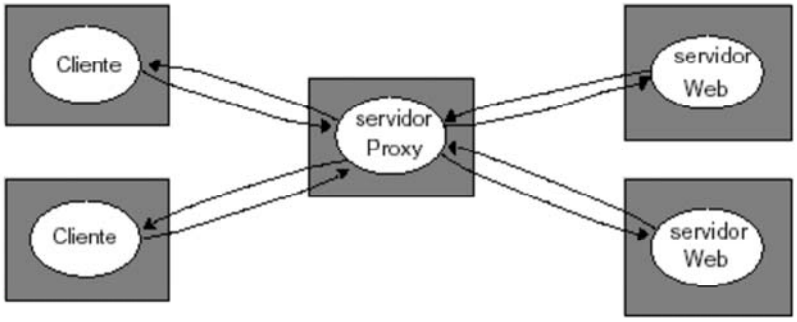


Figura 1.16: Servidor proxy

Los servidores proxy para el web (véase figura 1.16) proporcionan una caché compartida de recursos web a las máquinas cliente de uno o más sitios. El propósito de los servidores proxy es incrementar la disponibilidad y prestaciones del servicio, reduciendo la carga en redes de área amplia y en servidores web.

1.8.5. Procesos "de igual a igual" (peer-to-peer)

En esta arquitectura todos los procesos desempeñan tareas semejantes, interactuando cooperativamente como iguales para realizar una actividad distribuida o cómputo sin distinción entre clientes y servidores. La figura 1.17 muestra un ejemplo con tres instancias de esta situación; en general, n procesos parejos podrán interactuar entre ellos, dependiendo el patrón de comunicación de los requisitos de la aplicación.

La eliminación del proceso servidor reduce los retardos de comunicación entre procesos al acceder a objetos locales. Considérese una aplicación de *pizarra* distribuida que permite que usuarios en varias computadoras vean y modifiquen interactivamente un dibujo que se comparte entre ellos.

1.8.6. Código móvil

Los *applets* son el ejemplo más conocido y más ampliamente extendido de código móvil; un usuario que ejecute un navegador y que seleccione un enlace con un *applet*, cuyo código esté almacenado en un servidor web, descargará el código en el navegador y se ejecutará allí, tal y como se ve en la figura 1.18.

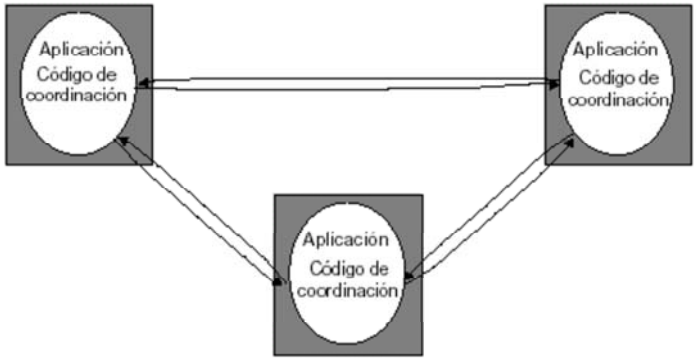


Figura 1.17: Aplicación basada en procesos *peer*

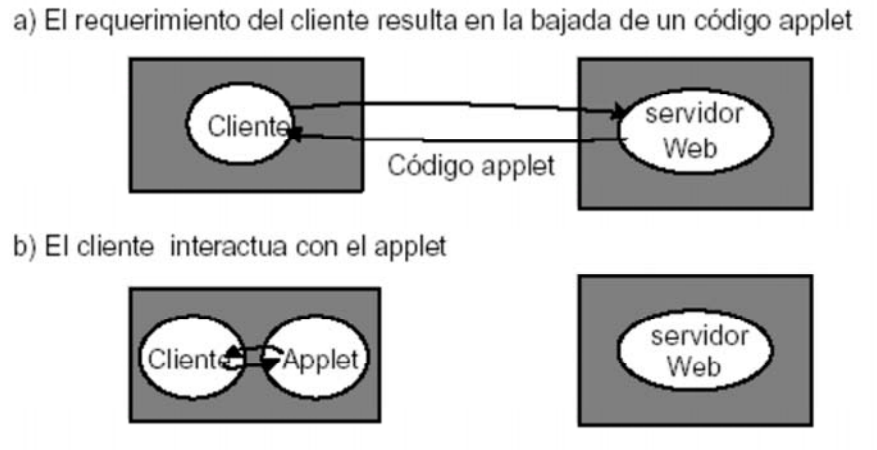


Figura 1.18: Código applet

1.8.7. Agentes móviles

Un agente móvil es un programa en ejecución (lo que incluye tanto código como datos) que se traslada de un computador a otro en la red realizando una tarea para alguien; por ejemplo, recolectando información, y retornando eventualmente con los resultados.

Los agentes móviles (como el código móvil) son una amenaza potencial de seguridad para los recursos de los computadores que visitan. El entorno que recibe el agente móvil debe decidir a cuál de los recursos locales le estará permitido tener acceso, en base a la identidad del usuario en cuyo nombre está actuando el agente; la identidad de éste debe incluirse de una forma segura en el código y los datos del agente móvil.

1.8.8. Computadores de red

Cuando las aplicaciones se ejecutan en un computador de oficina local del usuario, el sistema operativo y el software de aplicación para computadores de

oficina necesitan normalmente que gran parte del código y datos activos estén ubicados en un disco local. Pero la gestión de los archivos de aplicación y el mantenimiento del software de base local precisa un esfuerzo técnico considerable y de una naturaleza que la mayoría de los usuarios no están calificados para proporcionarlo.

El computador de red es una respuesta a este problema. Descarga su sistema operativo y cualquier aplicación software que necesite el usuario desde un servidor de archivos remoto. Las aplicaciones se lanzan localmente pero los archivos se gestionan desde un servidor de archivos remoto.

1.8.9. Clientes ligeros

El término *cliente ligero* se refiere a una capa de aplicación que soporta una interfaz de usuario basada en ventanas sobre un computador local del usuario mientras se ejecutan programas de aplicación en un computador remoto (ver figura 1.19). Esta arquitectura tiene los mismos costes bajos de gestión y hardware que en el esquema de computador de red, pero en lugar de descargar el código de las aplicaciones en el computador del usuario, se ejecutan en un *servidor de cómputo*, un potente computador que tiene capacidad para ejecutar gran número de aplicaciones simultáneamente. El servidor de cómputo será normalmente un multiprocesador o un sistema de computadoras fuertemente acopladas (*cluster*).

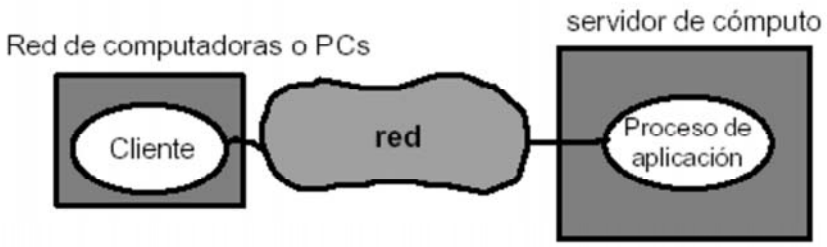


Figura 1.19: Clientes y servicio de cómputo