

Sistemas Distribuidos

1. Introducción
2. La Comunicación
3. Sistemas Operativos Distribuidos
4. Sincronización y Coordinación

En esta asignatura se pretende presentar algunas nociones básicas de los sistemas distribuidos distribuidas en estos cuatro capítulos.

En la Introducción presentaremos a los sistemas distribuidos mediante sus características y la comparación con los sistemas centralizados e independientes.

A continuación se tratarán los mecanismos de comunicación entre procesos que les permiten abstraerse del hecho de que quizás se encuentran en máquinas distintas. Entre estos mecanismos abordaremos las RPC y el estándar de la OMA: CORBA.

El capítulo de Sistemas Operativos se dedica casi íntegramente al reparto de carga, esto es, al aprovechamiento eficiente de las múltiples unidades de proceso que componen el sistema distribuido.

Por último se abordarán los mecanismos de Sincronización y Coordinación que deben utilizar los procesos para mantener la consistencia del sistema distribuido.

Sistemas Distribuidos

1. Introducción

1. ¿Qué es un Sistema Distribuido?
 2. Características de los S.D.
 3. Ventajas e Inconvenientes de los S.D.
 4. Aplicaciones de los S.D.
2. La Comunicación
 3. Sistemas Operativos Distribuidos
 4. Sincronización y Coordinación

En esta introducción vamos a tratar de hacer una presentación de los sistemas distribuidos, más que mediante definiciones, mediante sus características.

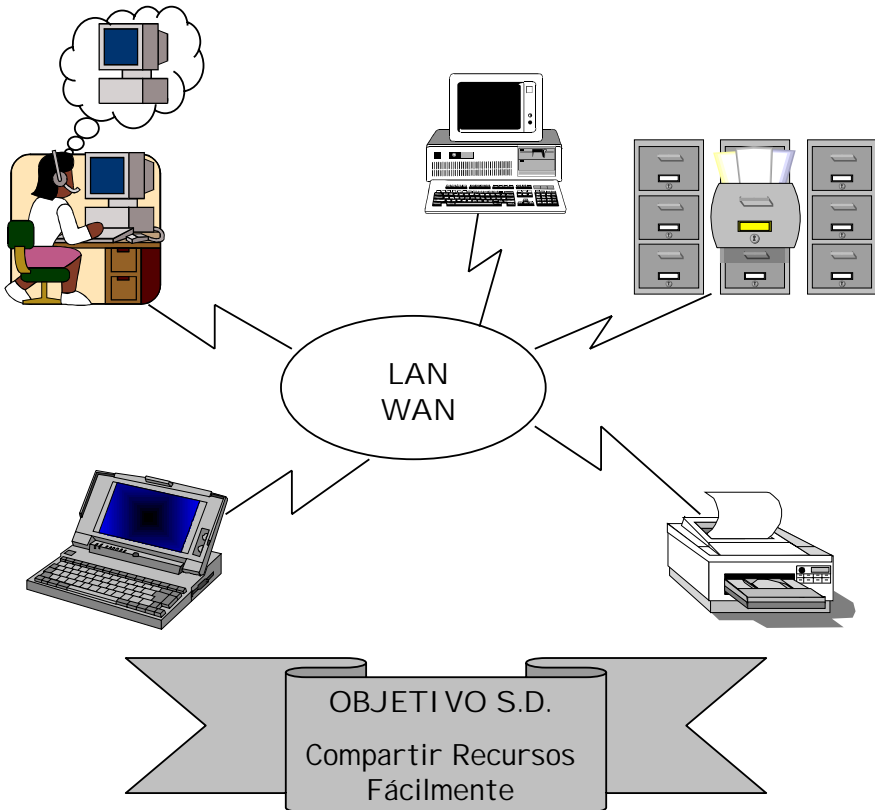
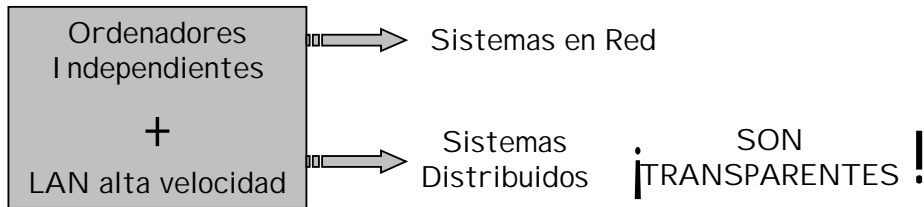
A continuación se verán algunas ventajas e inconvenientes de los sistemas distribuidos frente a los sistemas centralizados (sistemas multiusuario convencionales) y frente a los sistemas individuales e independientes, como son los PC's con sistema operativo monousuario.

Por último se comentarán algunas áreas de aplicación en las que los sistemas distribuidos se muestran especialmente apropiados.

¿Qué es un Sistema Distribuido?

Años 60-70: Sistemas Centralizados (Caros)

Años 80: Ordenadores Personales | Accesorios Caros y Poco Aprovechables



Cuando a finales de los años 50 los ordenadores empezaron a estar comercialmente disponibles, los pocos que había resultaban caros y no se aprovechaban bien. Los **grandes ordenadores centralizados** multiusuario (años 60 y 70) resolvieron el problema del aprovechamiento. Aún así, pocas empresas podían disponer de ellos. Pero a mediados de los años 80, surgió el desarrollo de potentes microprocesadores y la aparición del PC, un **ordenador personal**. Si bien éste era de precio asequible, el problema de los PC's era la carencia o dificultad para disponer de recursos como impresoras de alta calidad, *scanners*, procesadores especializados, grandes sistemas de almacenamiento, etc., pues éstos seguían siendo considerablemente caros y poco aprovechables.

La solución vino de la mano de las redes de alta velocidad, que permitieron la interconexión de todo tipo de ordenadores (grandes, medianos, pequeños, de uso general y especializados) mediante una red de comunicaciones, lo cual les permitió compartir y aprovechar recursos. Con la unión de los ordenadores personales y las LAN nacieron los **Sistemas en Red**. Los usuarios de estos sistemas pueden hacer uso de los recursos que les ofrece la red de ordenadores, son conscientes de la multiplicidad de máquinas en la red y necesitan conocerlas, pues para acceder a cada recurso deben saber en qué máquina está ubicado tal recurso, y deben conectarse explícitamente a la máquina apropiada para poder acceder al recurso deseado (o para transmitir datos de la máquina local a la remota).

El siguiente paso fue el de los **Sistemas Distribuidos**. Con estos sistemas, los usuarios pueden saber que hay multiplicidad de estaciones y equipos en la red, pero no necesitan conocerlos explícitamente (ni cuántos, ni cuáles), ellos solamente saben que hay recursos en la red y conocen su nombre o identificador, de tal forma que pueden acceder a ellos de igual manera que acceden a los recursos locales (por su nombre), sin tener que conectarse (explícitamente) en ningún caso a la máquina propietaria para utilizar el recurso. Es más, ni siquiera necesita conocer el nombre de la máquina que alberga el recurso.

Como podemos ver, la diferencia fundamental entre los sistemas en red y los sistemas distribuidos es la **transparencia**. En el sistema distribuido, la composición y estructura de la red le pasa desapercibida al usuario, es decir, ni la ve ni le importa. Solamente le importan los recursos disponibles o, a veces, simplemente el tipo de los recursos disponibles, sin tener en cuenta en qué máquina están realmente ubicados.

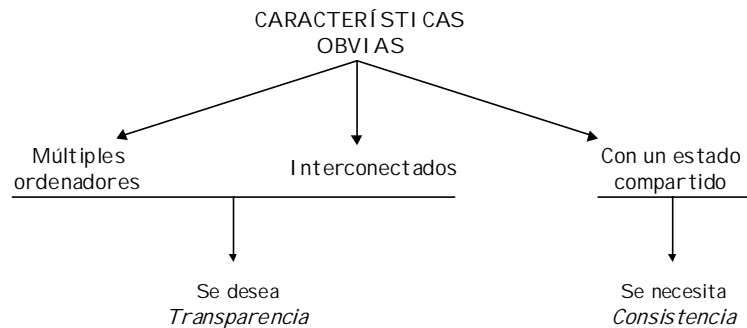
Aunque el concepto ya tiene bastantes años, todavía está en completo desarrollo, y hay distintas ideas y opiniones sobre lo que debe ser un sistema distribuido. Una de las definiciones más populares es la de Lamport: "*Un Sistema Distribuido es aquel que deja de funcionar cuando se estropea una máquina de la que ni siquiera habíamos oído hablar*". Esta definición es tan graciosa como significativa.

Más formal es la definición que cita Tanenbaum: "*Se trata de un conjunto de ordenadores independientes e intercomunicados por una red y provistos de software distribuido, tal que el usuario lo percibe como si se tratara de un único ordenador*". Es decir, que no se debe ver al sistema compuesto por elementos heterogéneos (aunque así sea), cuyo protocolo de acceso depende del sistema operativo de cada ordenador, de su estructura de ficheros o de los comandos que en cada ordenador se utilicen para acceder a sus componentes, sino como una serie de recursos a los que se accede como si fueran locales.

Lo que debe quedar claro es que el **objetivo primordial** de los sistemas distribuidos es el **compartimiento fácil y eficiente de los recursos** entre múltiples usuarios. Por esto, no se les debe confundir con los sistemas paralelos, cuyo propósito es acelerar la ejecución de un cierto programa en un único ordenador (normalmente aplicaciones científicas), y están compuestos por múltiples procesadores que suelen compartir memoria y reloj.

Características de los S.D.

Características de los Sistemas Distribuidos



Sistemas Distribuidos

Introducción - 4

Sistemas Distribuidos

Introducción - 4

Un sistema distribuido puede verse como un sistema formado por varios ordenadores haciendo algo conjuntamente, de lo que se desprenden tres características inmediatas:

- **Compuesto por múltiples ordenadores.** Un sistema distribuido está compuesto de más de un sistema independiente, cada uno con una o más CPU's, memoria local, memoria secundaria (discos) y, en general, conexiones con periféricos de acceso inmediato (*on line*).
- **Hay interconexión entre ellos.** Parece claro que si varios ordenadores distintos van a colaborar en la realización de tareas, deben comunicarse y sincronizarse entre ellos, por lo que debe haber alguna línea o red de interconexión.
- **Tienen un estado compartido.** Si los ordenadores realizan un trabajo conjuntamente, deben mantener un estado compartido, es decir, todos los ordenadores tienen la misma visión del estado del sistema distribuido (tablas, bases de datos del sistema, de servidores, etc).

Pero la construcción de un sistema distribuido que se comporte según esperan los usuarios, va a requerir considerar otro conjunto de características que se pueden resumir en estas dos: Consistencia y Transparencia.

La **consistencia** es una necesidad imperativa, pues sin ella, simplemente es que el sistema no funciona. La **transparencia**, como ya veremos en detalle, si bien es algo de que caracteriza especialmente a los sistemas distribuidos, su ausencia no impide que el sistema funcione; no obstante, es una característica deseable, de tal manera que cuanto más transparente sea un sistema distribuido, más lo podemos considerar como tal.

Veamos a continuación, en detalle, estas dos características, que aunque no son exclusivas de los sistemas distribuidos, pues también tienen mucho sentido en los centralizados, veremos que cobran una mayor importancia en los distribuidos.

En Sistemas
Centralizados

INCONSISTENCIAS
si acceso incontrolado a
datos compartidos

En Sistemas Distribuidos

Más Peligros

Peores Consecuencias

- ✓Consistencia de Actualización
- ✓Consistencia de Replicación
- ✓Consistencia de Caché
- ✓Consistencia de Reloj
- ✓Consistencia de Interfaz de Usuario

El problema de la consistencia (o mejor, de la falta de consistencia) no es exclusivo de los sistemas distribuidos. Ya sabemos que en los sistemas centralizados el acceso concurrente a los datos puede dejarlos inconsistentes si no se proveen mecanismos que proporcionen exclusión mutua en el acceso a los mismos.

En los sistemas distribuidos el problema de la inconsistencia cobra una mayor dimensión, tanto por la importancia como por la cantidad de situaciones en que pueden producirse problemas. Un sistema distribuido es un único sistema formado por múltiples máquinas independientes. Ya que es un único sistema, debe tener un único estado global compartido por todos los equipos que lo componen. El estado global se refiere a datos: como tablas de mantenimiento del sistema, la hora actual o datos que estén compartidos por procesos de distintas máquinas. Como veremos, hay situaciones en las que no resulta fácil mantener estos datos iguales en todas las máquinas.

Atendiendo a las situaciones, en un sistema distribuido se pueden producir inconsistencias de los siguientes tipos:

- Inconsistencia de Actualización
- Inconsistencia de Replicación
- Inconsistencia de Caché
- Inconsistencia de Reloj
- Inconsistencia de Interfaz de Usuario

En las siguientes transparencias iremos comentando los problemas de inconsistencia de cada uno de estos tipos.

CONSISTENCIA DE ACTUALIZACIÓN

Se pierde cuando la escritura concurrente en datos compartidos no se realiza como una única acción atómica en exclusión mutua.

Problema común en Bases de Datos

Más grave en S.D.

- Se presta a tener más usuarios
- Gestión del S.D. completo depende de B.D.

Solución: **TRANSACCIONES**

(ACID)

```
Begin_Transaction
End_Transaction
Read
Write
Abort_Transaction
```

Cambio de Cuenta

BEGIN_TRANSACTION;

Retiro (cantidad, cuenta_1);

Deposito (cantidad, cuenta_2);

END_TRANSACTION;

PREMIO: Viaje a Hawai

BEGIN_TRANSACTION;

Reserva (Madrid, N.Y.);

Reserva (N.Y., Los Angeles);

Reserva (Los Angeles, Hawai); **¡Lleno!**

END_TRANSACTION;

↓
ABORT_TRANSACTION

Consistencia de Actualización

Cuando varios procesos acceden concurrentemente a un dato para actualizarlo se puede producir una inconsistencia, porque la actualización de todo el dato en su conjunto no se realiza como una única operación atómica en exclusión mutua. Es el problema clásico del acceso a bases de datos, o simplemente a datos compartidos, pero en los sistemas distribuidos cobra una particular importancia debido a que estos se prestan a tener más usuarios. Además, la propia gestión del sistema distribuido completo también depende de los datos de control del sistema operativo de cada equipo de la red.

Este tipo de inconsistencia se evita utilizando **Transacciones**. Las transacciones son las primitivas equivalentes a las de entrada y salida de una región crítica que se utilizan para proteger la consistencia de un dato compartido. Además, aseguran que las operaciones incluidas en la transacción o se realizan todas, o no se ejecuta ninguna. Veámoslo con un ejemplo. Supongamos que un cliente de un banco se conecta por Internet al banco para sacar dinero de una cuenta y depositarlo en otra. Para asegurarse de que no se pierde el dinero si el sistema se cae durante las operaciones, se utiliza esta secuencia de acciones:

```
BEGIN_TRANSACTION;
    Retiro (cantidad, cuenta_1);
    Deposito (cantidad, cuenta_2);
END_TRANSACTION;
```


Si el sistema se cae después del Retiro, y antes del Deposito, al rearrancar, las cuentas 1 y 2, estarán igual que antes de comenzar la transacción. Esto es porque hasta que no se llega a END_TRANSACTION no se reflejan en la memoria persistente las acciones encerradas en la transacción.


Otra primitiva común en las transacciones es ABORT_TRANSACTION, que permite abortar una transacción, anulando cualquier operación que se hubiera realizado hasta el momento dentro de la transacción abortada.

Se utilizan las siglas **ACID** para referirse a las propiedades, en inglés, de las transacciones: *Atomicity, Consistency, Isolation, Durability*.

CONSISTENCIA DE REPLICACION

Cuando un conjunto de datos debe mantenerse replicado en varias estaciones.

Cuando hay modificación en uno de ellos  MULTICAST

Si no llega a alguno  INCONSISTENCIA

Ejemplo: Juego multiusuario en red.

Consistencia de Replica

Cuando un conjunto de datos debe mantenerse replicado en diversos ordenadores de la red, pudiendo ser modificado en cualquiera de ellos, claramente se pueden producir situaciones en las que tales datos no son iguales en todos los ordenadores al mismo tiempo. Como ejemplo podemos pensar en un juego multiusuario en la red. Cuando un usuario toma cualquier acción, ésta debe propagarse inmediatamente, mediante *multicast*, a los ordenadores del resto de los usuarios. Si esto no es así, cada uno está teniendo una visión distinta del mismo juego.

También pueden tenerse datos replicados en el sistema distribuido para mejorar la velocidad general del sistema o para tener tolerancia a fallos.

CONSISTENCIA DE CACHE

Para agilizar acceso
a datos compartidos

MEMORIA
CACHE

Cuando un cliente modifica su caché

Las copias de los otros
clientes quedan
anticuadas

¡ INCONSISTENCIA !

Consistencia de Caché

Cuando un cliente accede a un recurso (un fichero de datos), se pueden guardar copias de estos datos en una memoria local del cliente (memoria caché) para facilitar su acceso en posteriores referencias, evitando tener que transferir de nuevo los datos por la red. El problema de la consistencia surge cuando un cliente actualiza datos que también residen en las memorias caché de otros clientes. En ese momento se dice que las copias que están en otras cachés quedan anticuadas. Hay distintas técnicas para asegurar la consistencia de las cachés, y se suelen tratar en la gestión de memoria de los sistemas operativos distribuidos y en las arquitecturas de sistemas multiprocesadores.

Aunque tanto en la réplica de datos como en la memoria caché se tienen datos duplicados, obsérvese que el objetivo de la réplica de datos es la tolerancia a fallos y la rapidez global del sistema mediante el reparto de datos frente a múltiples consultas, mientras que el objetivo de la memoria caché es mejorar la rapidez de los accesos locales mediante el modelo de jerarquías de memoria.

CONSISTENCIA DE RELOJ

Hay algoritmos que dependen de la hora (*timestamps*)

- Make
- Sustitución de páginas

En S.D. hay que comparar *timestamps* generados en una estación remota con otros locales.

HAY QUE SINCRONIZAR
LOS RELOJES

Una Solución: Enviar la hora a todos los ordenadores

Retardo de μ n ms?

Consistencia de Reloj

Muchos de los algoritmos utilizados en aplicaciones y programación de sistemas dependen de unas marcas de tiempo o *timestamps* que indican el momento en el que ha sucedido un evento. Esto lo utiliza por ejemplo el comando *make*, o ciertos algoritmos de reemplazo de páginas en memoria virtual.

En sistemas distribuidos, una marca de tiempo generada en un ordenador se puede pasar a cualquier otro, donde se podrá comparar con otras marcas generadas localmente. El problema estriba en que **no resulta fácil mantener la misma hora física** en todos los ordenadores o componentes de la red simultáneamente. Una posibilidad consiste en enviar la hora por la red a todos los ordenadores, pues a pesar de que la transmisión en sí ya requiere un tiempo, esto se podría solucionar fácilmente si se pudiera añadir el tiempo de transmisión a la hora recibida en cada estación; pero no es así, pues el tiempo de transmisión en una red es algo bastante impredecible.

En el capítulo dedicado a la Sincronización y Coordinación se afronta este problema y se ofrecen diversas soluciones.

CONSISTENCIA DE INTERFAZ DE USUARIO

En una aplicación interactiva distribuida, a veces, se pulsa un botón del ratón

¡y no cambia nada en la pantalla!

INCONSISTENCIA DE INTERFAZ

El retardo no debe ser mayor de 0,1 s.

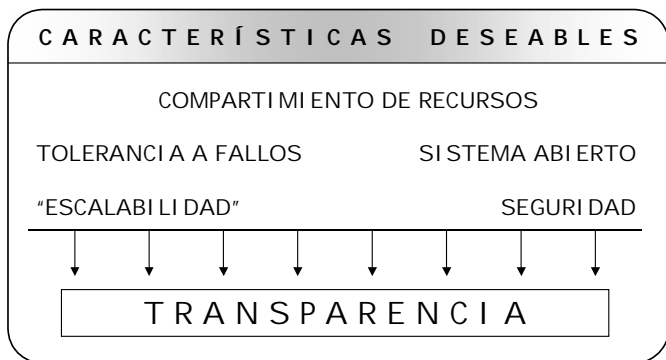
Para dar la impresión de disponer de una máquina dedicada

Consistencia de Interfaz de Usuario

Cuando un usuario está trabajando con una aplicación interactiva distribuida, hay veces que, por ejemplo, pulsa el botón del ratón para realizar cierta acción. Esto seguramente genera el envío de algún mensaje por la red hacia el servidor, que recibirá el mensaje, lo procesará y transmitirá, de vuelta a la estación cliente, la respuesta o las acciones a ejecutar. Pues bien, desde el momento en que se pulsó el botón del ratón, el usuario ha estado esperando a que la acción esperada se manifieste en su pantalla. Es decir, que desde que se pulsa una tecla o botón hasta que la acción correspondiente se refleja en la pantalla se produce una inconsistencia de interfaz, pues no se corresponde lo tecleado con lo reflejado en la pantalla.

Ciertos estudios de ergonomía indican que el retardo en mostrar los datos correspondientes en una pantalla no debe ser mayor de una décima de segundo si se quiere dar la impresión de estar trabajando con una máquina dedicada.

Como vemos, en un sistema distribuido, las aplicaciones interactivas pueden tener un comportamiento (en términos de acción-respuesta) similar a los sistemas centralizados, en los que las respuestas a las acciones del usuario también pueden ser lentas e irregulares (dependiendo de la carga del sistema centralizado o del tráfico de la red en el sistema distribuido).



La construcción de un sistema distribuido que se comporte según esperan los usuarios, requiere la consideración de otras características, que en principio diremos que son deseables:

Compartimiento de Recursos

Sistema Abierto

Escalabilidad

Tolerancia a Fallos

Seguridad

Un sistema que reúna todas estas propiedades conseguirá entonces la principal característica deseada por el usuario: la TRANSPARENCIA. Es decir, su sistema distribuido va a comportarse de manera ideal sin que el usuario se aperciba en ningún momento de los posibles y normales problemas que pueden producirse en el sistema (en las máquinas que lo componen). Dicho de otra manera, en el sistema se pueden producir fallos, pero el usuario no los va a notar; el sistema podrá crecer, pero para ello no habrá que parar el sistema; se podrán compartir recursos, pero no será necesario saber dónde están o a dónde se han movido; ...

Estas características deben entenderse simplemente como **deseables** en un sistema distribuido, no como requisitos imprescindibles para considerar a un sistema como tal. Podremos decir que un sistema es más o menos distribuido dependiendo de cuántas de estas características reúne.

En los siguientes apartados haremos una breve descripción de cada una de estas propiedades.

Posibilidad de Utilización de Recursos y Datos Públicos a los Distintos Usuarios Autorizados del Sistema.

BENEFICIOS DE LA COMPARTICIÓN

HARDWARE
Economía

SOFTWARE
Desarrollo en equipo
Acceso a Datos

¿CÓMO SE COMPARTEN RECURSOS?

EN SISTEMAS CENTRALIZADOS

Directamente

EN SISTEMAS DISTRIBUIDOS

No Directamente

Recursos Encapsulados en otra Máquina

Requiere Comunicación

Interfaz Homogéneo

Gestor de Recursos

Modelo CLIENTE-SERVIDOR

Aunque el término **recurso** pueda parecer un poco abstracto, es el que mejor caracteriza el conjunto de cosas que pueden compartirse en un sistema distribuido, puesto que éstas se extienden desde las puramente hardware, como discos, impresoras, procesadores, etc., hasta las entidades definidas dentro del software, como son los ficheros, bases de datos, etc.

Los beneficios de poder compartir recursos se aprecian enseguida.

Desde el punto de vista hardware, tenemos la economía. Está claro que una impresora conectada a una estación de trabajo, por ejemplo, que no se está utilizando el 100% del tiempo, se aprovecharía mejor si pudieran imprimirse ficheros provenientes de otras estaciones, sin necesidad de tener que comprar otra impresora y utilizarlas las dos el 10% del tiempo solamente.

Desde un punto de vista más lógico, lo que va a permitir la interconexión de equipos va a ser la posibilidad de un acceso a datos que permita un trabajo cooperativo en equipo que antes no era posible en sistemas independientes, y sí en sistemas centralizados, aunque con la problemática de que estos últimos no reúnen todas las características deseables enumeradas en la transparencia anterior.

En los sistemas centralizados los recursos ya se compartían, y además directamente, puesto que eran recursos locales. En los sistemas distribuidos el acceso no es tan automático o inmediato, puesto que puede ocurrir fácilmente que los recursos residan o pertenezcan a otra máquina distinta de la que quiere utilizarlos, es decir, que estén encapsulados en otra máquina. Por esto, cada estación que tiene un recurso concreto que compartir, lo rodea de un **gestor de recurso** o servidor. Este gestor se va a encargar de atender y gestionar las peticiones de utilización de tal recurso. Para poder atender peticiones de máquinas dispares se va a requerir una **interfaz homogénea** de comunicación, de tal forma que máquinas distintas con sistemas operativos diversos puedan acceder u ofrecer recursos al resto de las estaciones del sistema.

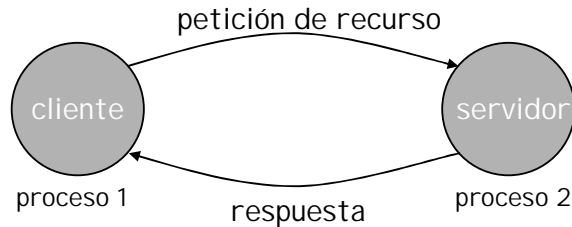
El esquema más utilizado para ofrecer y utilizar recursos se denomina **modelo cliente-servidor**. Lo veremos en la siguiente transparencia.

Dado el auge de la programación orientada a objetos, también se han realizado varias propuestas de estandarización, que aunque basadas en el modelo cliente-servidor, refuerzan y aprovechan las características de los lenguajes orientados a objetos. Algunos ejemplos de estas propuestas son: CORBA, del OMG (Object Management Group); Java-RMI (Remote Method Invocation); y DCOM, de Microsoft.

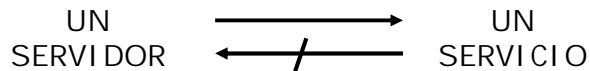
Características de los S.D. ...Compartición de Recursos

Modelo CLIENTE-SERVIDOR

- ✓Proceso Servidor \implies Gestor de Recursos
- ✓Proceso Cliente \implies Usa Hw. y Sw. Compartido



No debe verse al Gestor del Recurso como el proveedor centralizado del recurso.



Se requiere consistencia de interfaz de usuario

No todos los recursos pueden compartirse de igual manera

- ✓RAM
- ✓Procesador
- ✓Interfaz de acceso a la red

Ya hemos dicho que un recurso está encapsulado en una máquina y que está gestionado por un gestor de recurso el cual recibe el nombre de **servidor**. Por otra parte tenemos los procesos que acceden y utilizan el recurso ofrecido por el servidor. A estos procesos, que pueden estar en la misma máquina o en estaciones remotas, les denominaremos **clientes**. Por lo tanto, en un momento dado, en un sistema distribuido habrá una serie de procesos servidores y otro conjunto de procesos clientes. Los ordenadores en los que residan estos procesos pueden ser de distintos tipos, pueden ejecutarse en la misma máquina e incluso un proceso servidor puede, a su vez ser cliente de otro proceso servidor para poder conseguir la totalidad de los servicios que ofrece.

El protocolo de comunicación del modelo cliente-servidor es muy simple. Como podemos ver en la transparencia, en primer lugar el cliente solicita el servicio de un recurso al servidor correspondiente, éste realiza el trabajo solicitado y a continuación le devuelve el resultado o la respuesta al cliente.

El servidor de un recurso es el gestor de ese recurso, y solamente de ese recurso concreto, no de otros de similares características. Es decir, no debe verse al gestor de un recurso como el gestor centralizado de todos los recursos de ese tipo (pues para ello habrá un nivel superior de jerarquía de servidores), pues si el servicio de un recurso está centralizado puede derivarse hacia los problemas típicos de los sistemas centralizados, y que en los distribuidos se quieren evitar.

Por lo tanto, debe hacerse una distinción clara entre el servicio y el servidor de un recurso. El **servicio** de un recurso concreto es la especificación de las operaciones que ofrece ese recurso a sus clientes potenciales. Describe las primitivas disponibles, los parámetros que utiliza y las acciones que lleva a cabo. Es decir, especifica la interfaz de utilización del recurso. Por el contrario, un **servidor** de un recurso es un proceso que se ejecuta en alguna máquina y que ayuda a implementar el servicio de ese recurso. El servicio de un recurso puede disponer de varios servidores, pero los clientes no tienen por qué conocer ni el número de servidores ni su dirección. Todo lo que saben es que al llamar a los procedimientos indicados en el servicio, el trabajo necesario se realiza de alguna manera y se obtienen los resultados solicitados. Debe hacerse notar aquí, la necesidad de la consistencia de la interfaz de usuario para que el sistema funcione correctamente, pues de nada sirve la transparencia si los resultados se obtienen en un tiempo razonable.

Es importante tener en cuenta que las estaciones u ordenadores no son clientes ni servidores, simplemente tienen recursos que comparten unos con otros. Son los procesos de estos ordenadores los que pueden comportarse como clientes o como servidores, dependiendo de si en un momento dado está utilizando u ofreciendo recursos. Por esto, una máquina puede ser cliente y servidora a la vez si tiene procesos clientes y servidores al mismo tiempo.

Como un apunte final, debemos observar que no todos los recursos disponibles en una estación de trabajo van a poderse compartir con la misma facilidad, eficiencia y dedicación. Pensemos, por ejemplo, en el procesador y la memoria central de un ordenador. Aunque veremos que se pueden compartir para la ejecución remota de programas, esto tendrá ciertas limitaciones y carencias que no están presentes cuando se ejecutan sobre la memoria y procesador de la propia máquina. Un ejemplo claro de recurso hardware no compartible es la tarjeta de acceso a la red. Difícilmente vamos a poder utilizar la tarjeta de red de una estación remota si la propia no dispone ya de su propio hardware de comunicaciones. (Aunque sí podría utilizar una máquina remota para acceder a otra red distinta a la que no se tiene acceso directo.)

Sistema Abierto

UN SISTEMA ES ABIERTO SÍ ES FÁCILMENTE AMPLIABLE

ASPECTO HW.

Periféricos
Memoria
Interfaces de com.

ASPECTO SW.

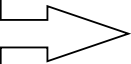
Extensiones del S.O.
Protocolos de comunicación
Nuevos recursos compartidos

¿Cómo conseguir Sistemas Abiertos?

INTERFACES PÚBLICAS

- Mecánicas y Eléctricas
- Del Software

Los Componentes de los S.D. son Heterogéneos



Imprescindible que sean Abiertos

¿Cómo conseguir Sistemas Distribuidos Abiertos?

COMUNICACIÓN UNIFORME Y PÚBLICA ENTRE PROCESOS

NUEVOS RECURSOS COMPARTIDOS

Se dice que un sistema es abierto si puede ampliarse fácilmente y sin dependencias de su constructor o diseñador original.

Un sistema puede ser abierto con respecto al hardware o al software. **Es abierto** respecto al hardware **si permite añadirle nuevos periféricos**, memoria o cualquier elemento hardware en general (claro está, siempre conforme con la arquitectura base del sistema), y sin que los componentes añadidos tengan que ser necesariamente del constructor original del ordenador. Un sistema **es abierto** respecto al software **si se permite añadir o sustituir programas o componentes software** de manera fácil e, igualmente, sin que dichos programas provengan necesariamente del fabricante original.

Bien, hasta ahora hemos dado definiciones, pero **¿cómo se consigue esto** que hemos dicho?, es decir, ¿cómo se consigue que cualquier fabricante pueda construir una tarjeta de comunicaciones que sirva para ese ordenador? o ¿cómo puede una empresa de software ajena ofrecer una extensión de un sistema operativo que no es suyo?

La respuesta solo puede ser de una manera: **haciendo públicas las interfaces de comunicación**; tanto las interfaces del hardware (mecánicas y eléctricas) como las del software. Publicar la interfaz del software de un sistema significa simplemente publicar las especificaciones de las llamadas al sistema operativo (las *system calls*). No obstante, un sistema que simplemente ofrece las especificaciones de las llamadas al sistema no es del todo abierto, porque tiene limitaciones: ¡no se pueden añadir nuevas llamadas al sistema! Para ser completamente abierto (respecto al software) debe publicar incluso la forma o medios por los que puedan incorporar nuevas llamadas al sistema (por ejemplo para cubrir los servicios proporcionados por nuevos tipos de periféricos).

En cuanto a los sistemas distribuidos, está claro que nos interesa que sean abiertos, sobre todo si queremos que puedan estar formados por equipos y máquinas con arquitecturas heterogéneas y con diversos sistemas operativos, de tal forma que siempre podamos ampliarlo con nuevos y diferentes ordenadores y con nuevos servicios que puedan compartirse por los integrantes del sistema.

Entonces, **¿cómo hacer sistemas distribuidos abiertos?** Pues al igual que antes, proporcionando y publicando un **mecanismo uniforme para la intercomunicación entre los procesos**, así como las interfaces de los protocolos necesarios para poder solicitar y acceder a los recursos compartidos.

"Escalabilidad"

AL CRECER UN SISTEMA PUEDEN APARECER PROBLEMAS

EN EL HW.

No se pueden añadir más equipos

Dimensionamiento de las direcciones

EN EL SW.

Se pierden prestaciones

Centralización de Datos y Algoritmos

UN SISTEMA DISTRIBUIDO DEBE SER FÁCILMENTE AMPLIABLE, SIN QUE PARA ELLO LOS USUARIOS DEBAN MODIFICAR SU PROTOCOLO DE COMUNICACIÓN NI LA EFICIENCIA DEL SISTEMA SE VEA AFECTADA.

¿CÓMO?

Sobredimensionando las direcciones

Huyendo de la centralización.

Replicando Datos
(Consistencia de réplica)

Descentralizando Algoritmos

Múltiples Servidores

Caché
(Consistencia de caché)

El sistema informático de una pequeña y nueva empresa puede estar formado por dos ordenadores, un servidor de archivos y una impresora, todos ellos interconectados, teniendo así un pequeño sistema distribuido. En nuestra sociedad actual, con un poco de suerte, y algún ingrediente más, es posible que esta empresa crezca hasta tener una red de área local formada por cientos de ordenadores, varios servidores de ficheros, impresoras diversas, ... Pero nuestra empresa no acaba aquí, pues llega a ser una potente multinacional, con lo que su red de área local está conectada a otras redes de área extensa que alcanzan casi todos los rincones del planeta.

Desde luego no habría estado nada bien que cada vez que la empresa haya crecido un poquito y se haya añadido un nuevo equipo, hubiera que haber cambiado los equipos existentes para adaptarse a la nueva configuración. Parece claro que lo que le gustaría al usuario (el propietario de la empresa) es poder ir añadiendo y añadiendo equipos y componentes aprovechando los anteriores, por supuesto.

Al crecer los sistemas distribuidos pueden aparecer problemas tanto en el ámbito del hardware como en el del software. En el primer caso, nos podríamos encontrar con que llegado cierto momento no se pueden añadir nuevos ordenadores por el simple hecho de que se han acabado las direcciones disponibles, lo que obligaría a rehacer todo el sistema de comunicaciones y actualizarlo en todos los equipos de la red. Este problema ya lo han sufrido, por ejemplo, los habitantes de Londres y París, que en un momento dado sus compañías telefónicas agotaron todos los posibles números para los abonados, con lo que hubo que añadir tres nuevos números para permitir la continuación de la expansión telefónica. En resumen, se debe tener en cuenta el dimensionamiento y crecimiento de la red a la hora de diseñar el sistema de direcciones –algo que no se tuvo en cuenta en el diseño del actual sistema de direcciones IP de Internet (V. 4) y que está resuelto para la siguiente versión (V. 6).

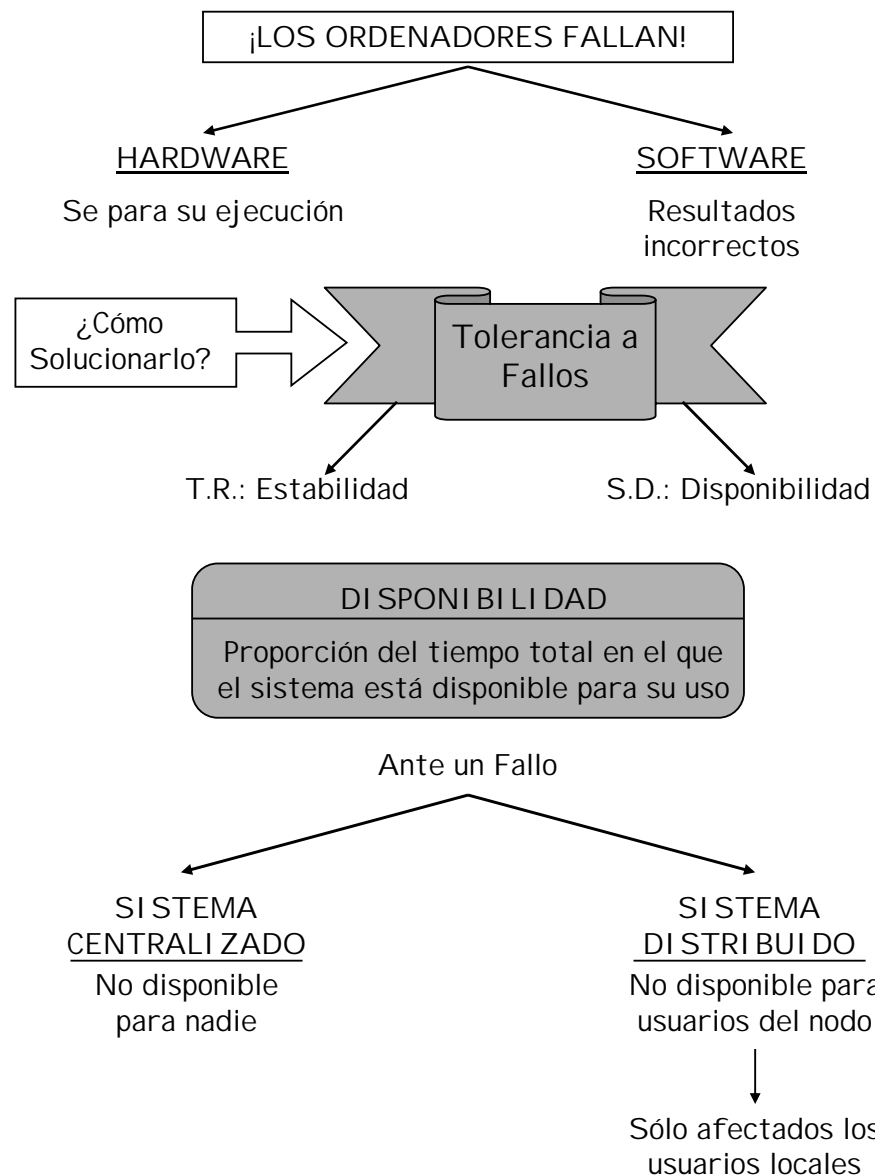
En cuanto a los problemas del ámbito del software que pueden surgir a raíz del crecimiento de los sistemas, está la pérdida de prestaciones. Es decir, el sistema se puede ir convirtiendo en un gigante que cada vez le cueste más moverse. Y esto puede deberse al hecho de que los algoritmos utilizados para la gestión del sistema estén centralizados, con lo que a mayor número de equipos, mayor cuello de botella en el equipo con el algoritmo.

Un sistema distribuido debe ser fácilmente ampliable, sin que para ello los usuarios deban modificar su protocolo de comunicación, ni se vea afectada la eficiencia del sistema.

¿Cómo conseguirlo? Por una parte sobredimensionando las direcciones posibles, o utilizando técnicas como las del código de operación con extensión, que no tienen ninguna limitación lógica. Por otra parte, huyendo de la centralización de servicios y algoritmos que creen cuellos de botella (entre otros problemas, como ya veremos). Por ejemplo, si un sistema tiene un servidor de ficheros, al aumentar significativamente el número de usuarios, seguramente será una buena decisión el replicar el servidor de ficheros para descentralizar el acceso y evitar así el cuello de botella en el servidor original. Si esto se hace así, el sistema crecerá sin que los usuarios noten ningún detrimento de las prestaciones.

Otra forma de replicar datos consiste en el mantenimiento de memorias caché en las estaciones de la red para evitar un acceso tan continuado al servidor, aunque esto conlleva serios problemas de consistencia de datos que deben tenerse en cuenta en el diseño del sistema.

Tolerancia a Fallos



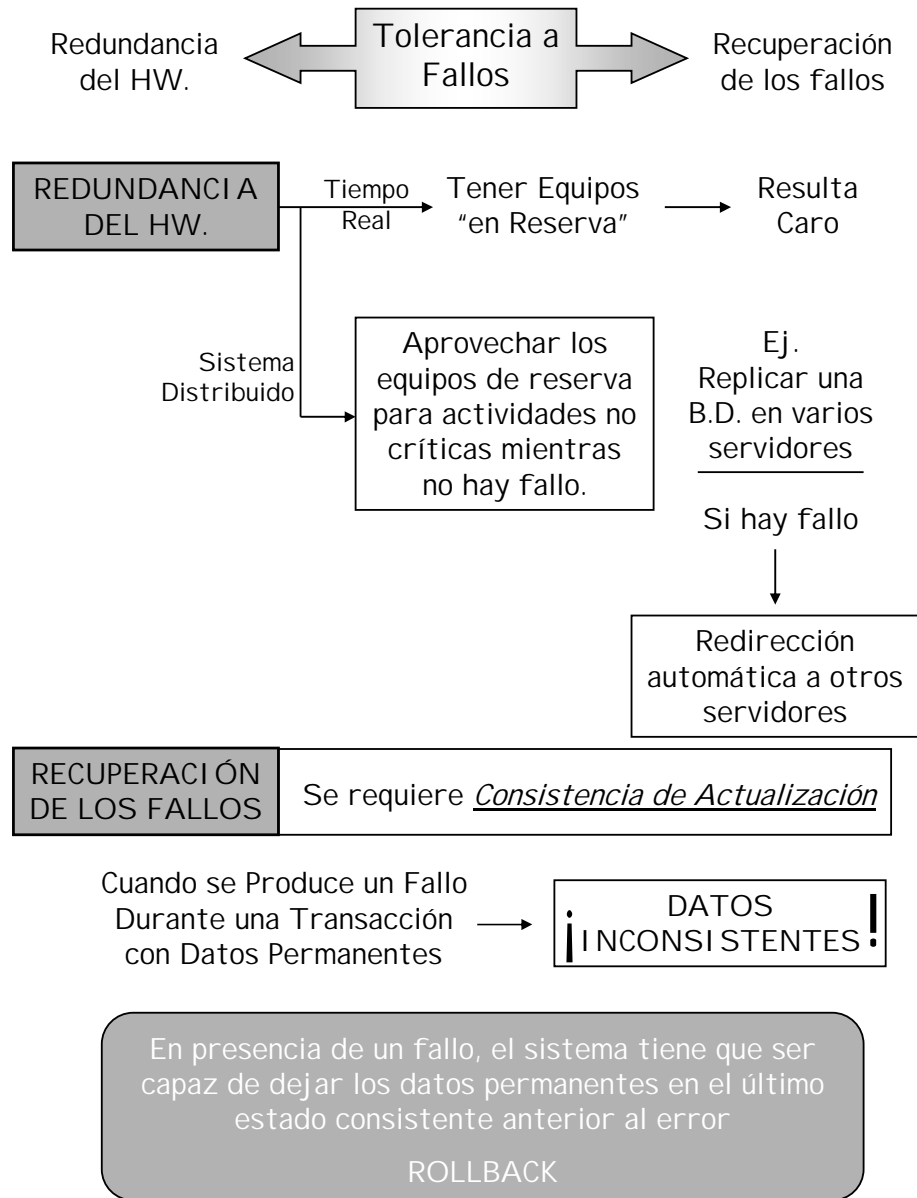
Sabemos que los ordenadores, circunstancialmente, tienen fallos en su comportamiento. Los fallos pueden ser debidos al software o a problemas en el hardware subyacente. Cuando el error se produce en el hardware, normalmente, simplemente se detiene la ejecución del sistema completo, mientras que si los problemas son de naturaleza software el comportamiento del programa o programas en ejecución es indefinido; en el mejor de los casos podría incluso continuar su ejecución hasta finalizar sin que el fallo haya llegado a afectar al resultado final. Sin embargo, esto no es normal, y lo común es que a partir de producirse el fallo, el programa comience a generar resultados o comportamientos incorrectos.

Para evitar estas situaciones, los sistemas deben ser **tolerantes a fallos**, lo cual suele implicar, por una parte, la redundancia del hardware y, por otra, la posibilidad de que los programas puedan recuperarse tras la detección de fallos. En la siguiente transparencia veremos con cierto detalle cada una de estas cuestiones.

En sistemas de tiempo real, la tolerancia a fallos evita las consecuencias catastróficas que podría traer la caída de un sistema. En entornos cuya parada no implique estas catástrofes, la tolerancia a fallos cobra importancia en el sentido de que permite una **alta disponibilidad del sistema**.

La disponibilidad de un sistema es una medida de la proporción del tiempo que está disponible para ser utilizado por el usuario. En un entorno centralizado (sin tolerancia a fallos), ante un fallo del sistema, éste queda inoperativo para todos los usuarios. Pues bien, en un sistema distribuido, su propia naturaleza intrínseca le da ciertas características de tolerancia a fallos, de tal forma que ante el fallo de uno de los nodos, solamente se ven afectados (en primera instancia) los usuarios de ese nodo, pudiendo estos incluso cambiarse, si es factible, a cualquier otra estación y continuar su trabajo mientras se repara la estación en fallo.

Debemos darnos cuenta de que si el nodo en fallo contiene información relevante para el sistema distribuido en su conjunto, dicho nodo deberá estar replicado para que el sistema sea tolerante a fallos. Sin embargo, hay ciertos componentes cuyo fallo es más difícil de solventar. Este es el caso de la red de comunicaciones, que normalmente no está replicada. No obstante, en redes de cierta magnitud, sí es normal disponer de rutas alternativas y de algoritmos de encaminamiento que, aunque con cierta degradación del rendimiento de la red, resuelven ciertos fallos de forma transparente al usuario.



Redundancia del Hardware.

Para disponer de sistemas que sean tolerantes a fallos del hardware, la opción inmediata es la replica del hardware, es decir, cada componente hardware del sistema está replicado (tiene otro componente igual de reserva). Así, en caso de fallo del sistema principal entra en funcionamiento el sistema de reserva. Este es el enfoque normal en sistemas de tiempo real o que por su naturaleza requieren una gran fiabilidad.

Esta solución, aunque trivial, resulta cara, pues además de duplicar el coste del sistema principal, se requiere también el hardware y software adicional para la detección de fallos, y esto es algo que no es fácil de justificar en un sistema distribuido, en el que uno de sus objetivos primordiales es la economía.

Una solución para nuestro sistema distribuido puede consistir en que el hardware replicado no esté simplemente de reserva, sino que se utilice, en condiciones normales, para la realización de otras tareas alternativas que no sean críticas. En caso de fallo, estos equipos alternativos pueden dejar de realizar las actividades normales y suplir a los equipos en fallo. Por ejemplo, en un sistema de consulta de una base de datos muy importante, ésta puede tenerse replicada en varios servidores que dan servicio a un gran número de usuarios, evitando así el cuello de botella que se produciría en el caso de tener un único servidor. En caso de fallo en uno de ellos, no se produce ninguna situación catastrófica, simplemente las consultas que le llegan al equipo en fallo se redirigen automáticamente a otro de los servidores. Simplemente se va a producir una ligera sobrecarga en el servidor que ha asumido las competencias del que ha fallado, pero el sistema en su conjunto sigue operativo.

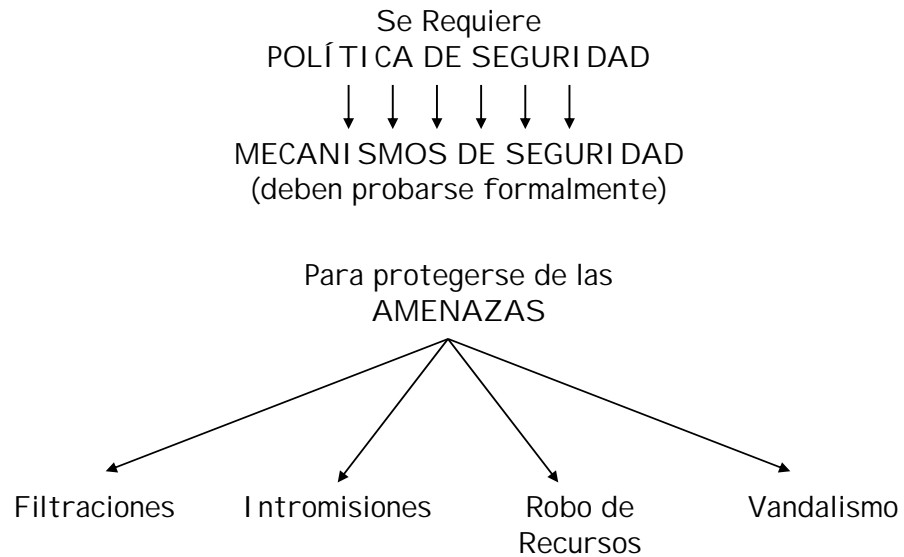
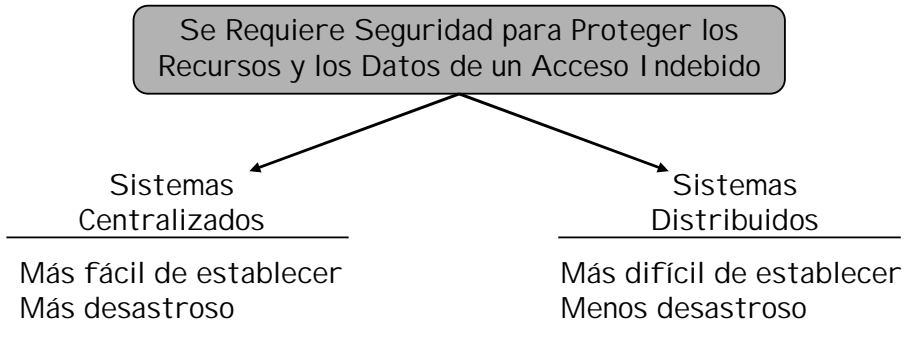
Recuperación de los Fallos.

En algunas aplicaciones, típicamente en bases de datos, la actualización de datos normalmente implica la escritura de un conjunto completo de datos, de tal manera que la actualización de solo una parte del conjunto dejaría inconsistente la información del conjunto de datos. Por ejemplo, si se quiere actualizar una ficha personal de un individuo, se comienza por actualizar el nombre, y si no se actualiza también el DNI, domicilio, etc. la ficha en su conjunto queda inconsistente.

Este tipo de **inconsistencias** se producen durante las transacciones **con datos permanentes**, es decir, datos que residen en una memoria permanente (como el disco).

En los sistemas distribuidos hay que tomar precauciones frente a estos posibles fallos que pueden dejar inconsistentes los datos permanentes de las aplicaciones que se estuvieran ejecutando en el momento del fallo. Es decir, en caso de fallo, el sistema tiene que ser capaz de dejar los datos permanentes en el último estado consistente anterior al error. A esto se le conoce comúnmente con el nombre de **rollback**.

La recuperación de los fallos, debe permitir, por ejemplo, que si se comienza la actualización de una ficha de un cliente, y cuando se ha realizado sólo la actualización parcial de la ficha, el sistema se cae, al reentrancar éste, la ficha debe mostrarse inalterada, es decir, como estaba antes de comenzar la actualización fallida.



En los S.D. las intromisiones suelen realizarse accediendo ilegalmente a los canales de comunicación	<ul style="list-style-type: none"> - Escucha - Suplantación - Alteración de Mensajes - Reenvío
--	--

En la práctica habitual, tanto los sistemas centralizados como los distribuidos tienen problemas para ofrecer una buena seguridad.

Un sistema centralizado tiene un único dominio de seguridad bajo el control de un único administrador, y el mecanismo de protección depende de un único sistema operativo. Parece que cuando todo está bajo el control de un solo sistema operativo, es más fácil de proteger (“casa con muchas puertas, es mala de guardar”). Por otra parte, resulta extremadamente difícil eliminar todos los fallos o debilidades de seguridad de un sistema operativo o de su entorno de trabajo. Así, si un intruso consiguiera acceder al sistema (con un único dominio de protección), le podría resultar fácil acceder a toda la información residente en él.

En los sistemas en red, las interconexiones entre ordenadores son físicamente inseguras, y ofrecen una vía de acceso ilegal a los datos mucho más fácil que en los sistemas centralizados. Sin embargo, los sistemas distribuidos tienen múltiples dominios de protección con diversos mecanismos y políticas de seguridad, y normalmente bajo el control de distintos administradores, por lo que en caso de acceso ilegal a uno de los dominios no se obtiene directamente el acceso a toda la información del sistema, solamente a la de ese dominio.

Para protegerse de los accesos indebidos se requieren diversos mecanismos de seguridad, así como una política clara de utilización de tales mecanismos (no sirve de nada un cerrojo en una puerta si siempre se tiene abierta). Los mecanismos y políticas deberán enfrentarse a los siguientes tipos de acciones ilegales:

- Filtraciones.** Adquisición de información por programas o personas no autorizadas.
- Intromisiones.** Alteración no autorizada de la información.
- Robo de recursos.** Uso de recursos sin autorización (impresoras, procesadores,...).
- Vandalismo.** Interferir la operación de un sistema sin ánimo de lucro.

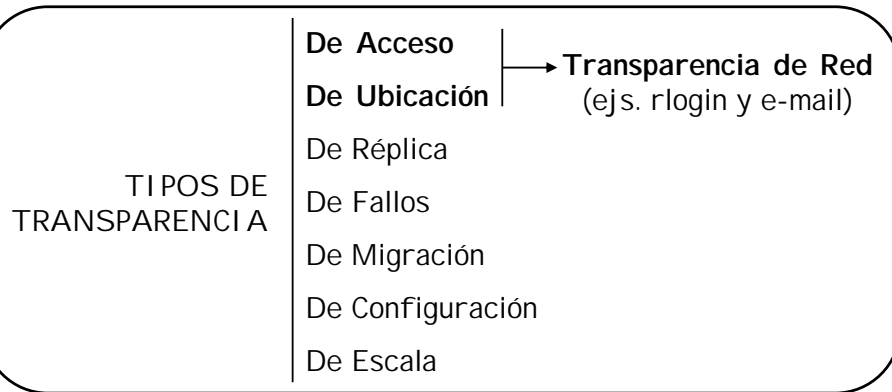
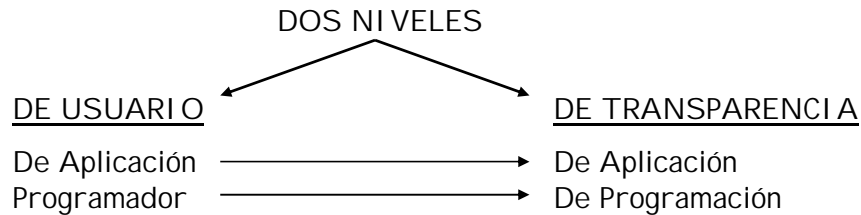
En los sistemas distribuidos estos actos suelen perpetrarse a través de los canales de comunicación realizando las siguientes acciones:

- Escucha.** Obtener copias de mensajes que pasan por la red.
- Suplantación.** Enviar o recibir mensajes utilizando la identidad de otro programa o persona.
- Alteración de Mensajes.** Interceptar mensajes y alterar su contenido.
- Reenvío.** Obtener copias de mensajes para utilizarlas posteriormente.

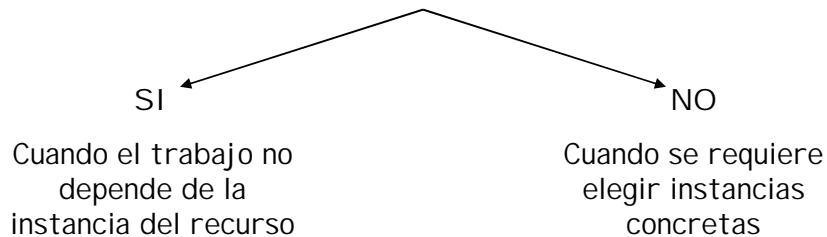
Existen mecanismos y políticas para impedir la realización de estas acciones, o al menos, para detectarlas e invalidarlas.

Transparencia

Consiste en ocultarle al usuario del sistema el hecho de que éste está compuesto por múltiples y heterogéneos equipos separados.



¿Es Recomendable la
Transparencia de Recursos?



La característica más importante de un sistema distribuido consiste en conseguir que el usuario tenga una única imagen del sistema. Es decir, se trata de que los diseñadores del sistema distribuido consigan engañar al usuario, haciéndole creer que el sistema (que en realidad es una colección de máquinas) no es más que otro sistema más de tiempo compartido, pero con muchas más prestaciones. Cuando se consigue esto, se dice que el sistema es transparente.

La **transparencia** se puede conseguir a dos niveles: a nivel **de usuario** y a nivel **de programación**. Lo más fácil es conseguir la transparencia al usuario (ocultarle la distribución al usuario). Por ejemplo, cuando un usuario teclea el comando `make` de Unix, no necesita saber si la compilación de los módulos implicados va a realizarse una tras otra secuencialmente en su misma máquina, o si en realidad se van a compilar simultáneamente en paralelo en múltiples máquinas y utilizando para ello diversos servidores de archivos. El usuario simplemente teclea el comando de igual manera que en un sistema centralizado y le aparece el resultado de la ejecución en su terminal, con lo cual, para él, se comporta como si estuviera en un sistema centralizado (posiblemente más rápido).

Bajando de nivel, llegamos a los programas en sí, y para lograr la transparencia a este nivel se debe ofrecer una interfaz de programación (las llamadas al sistema) que oculten la existencia de múltiples máquinas (con múltiples procesadores, discos, etc.). La solución obvia para que la semántica del comando `make` no varíe para el usuario de un sistema distribuido, consiste simplemente en programar de nuevo el comando `make` teniendo en cuenta la nueva arquitectura, es decir, modificando el nivel inmediato inferior. Si no hubiera que reescribir el programa `make` al pasarlo a un entorno distribuido, se tendría entonces transparencia de programación. Como podemos imaginarnos, esto es bastante más costoso de lograr que la simple transparencia de usuario.

Están identificados siete **tipos de transparencia**:

- de Acceso.** Permite que el acceso a objetos locales o remotos se realice utilizando idénticas operaciones.
- de Ubicación.** Permite el acceso a los objetos sin tener conocimiento de su ubicación.
- de Réplica.** Permite utilizar múltiples instancias de objetos (sin saberlo los usuarios o programas) para mejorar la fiabilidad y rendimiento del sistema.
- de Fallos.** Permite ocultar los fallos, pudiendo así los usuarios o aplicaciones completar sus tareas a pesar de los fallos de componentes hardware o software.
- de Migración.** Permite mover objetos o recursos de una ubicación a otra sin que ello afecte a la operación de usuarios o aplicaciones.
- de Configuración.** Permite la reconfiguración dinámica del sistema para mejorar el rendimiento a medida que varía la carga de trabajo.
- de Escala.** Permite que el sistema o las aplicaciones se expandan sin cambiar la estructura del sistema o los algoritmos de programación

Los dos tipos de transparencia más importantes son la transparencia de acceso y la de ubicación. A las dos juntas se las conoce como **transparencia de red**, pues son las que más influyen en la utilización de recursos distribuidos. La transparencia de red proporciona un tipo de acceso a los recursos similar al de los sistemas centralizados. Un ejemplo de ausencia de transparencia de red es el comando `rlogin`, el cual tiene una interfaz de usuario distinta de `login`. Sería preferible un único comando que permitiese el acceso a todas las máquinas de un dominio concreto del sistema distribuido.

La transparencia oculta cierta información de los recursos que normalmente no es de interés para los usuarios o programas. Esto es deseable cuando se hace referencia a un servicio para el que están provistas varias instancias de recursos o servidores y no nos importa cuál sea la instancia que nos preste ese servicio (por ejemplo, obtener cierta información de una base de datos). Pero hay ocasiones en que no se requiere tanta transparencia, porque se quiere poder especificar el recurso concreto que se desea para prestar un servicio (por ejemplo, cuando se desea imprimir un fichero pero se quiere que salga por la impresora más cercana).

Ventajas e Inconvenientes

<p>VENTAJAS DE LOS S.D. FRENTE A LOS CENTRALIZADOS</p>	<p>Economía: Mejor relación calidad/precio</p> <p>Velocidad: Más potencia que <i>mainframes</i></p> <p>Distribución Inherente: Aplicaciones especiales</p> <p>Fiabilidad: Tolerancia a fallos</p> <p>Escalabilidad: Fácil crecimiento incremental</p>
<p>VENTAJAS DE LOS S.D. FRENTE A LOS INDEPENDIENTES</p>	<p>Comparten Datos: Acceso a B.D. comunes</p> <p>Comparten Dispositivos: Impresoras, Scanners,...</p> <p>Comunicación: Facilita comunicación (e-mail)</p> <p>Flexibilidad: Reparto de carga eficiente</p>
<p>INCONVENIENTES DE LOS S.D.</p>	<p>Software: Hay poco</p> <p>Red: Se puede saturar</p> <p>Seguridad: Más difícil</p>

Ventajas de los Sistemas Distribuidos Frente a los Centralizados.

La principal ventaja de la descentralización es la **economía**. La ley de Grosh decía que "la potencia de un ordenador es proporcional al cuadrado de su precio", aunque esta proporción quizás está mejorando, lo cierto es que hoy en día el último modelo de un cierto microprocesador es el doble de caro que el modelo anterior, que solo es ligeramente más lento. Por esto, es preferible comprar varias CPUs baratas y ponerlas a trabajar juntas. Es decir que en los sistemas distribuidos se obtiene una mejor relación calidad/precio. (¡Ojo! las máquinas paralelas dan peor relación calidad/precio). Otra ventaja está en la **velocidad**, pues mientras que un sistema centralizado (no paralelo) tiene limitaciones físicas en su potencia de cálculo (por la máxima velocidad de la luz), en un sistema distribuido se puede obtener una mayor velocidad que en los sistemas centralizados. No obstante, no hay que olvidarse del problema de la distribución de un trabajo en tareas (no es fácil, y consume tiempo la comunicación y sincronización). Una tercera ventaja es que **se adapta mejor a las aplicaciones que son inherentemente distribuidas**, tales como los robots de una cadena de montaje de una factoría o bases de datos de empresas con múltiples sucursales. Los sistemas distribuidos son **más fiables**, pues al estar distribuida la carga y las labores entre múltiples máquinas, hay menor dependencia del fallo de cualquiera de ellas. Por último diremos que el crecimiento del sistema puede realizarse de manera incremental. Si se dispone de un *mainframe*, las únicas soluciones son cambiarlo por otro modelo mejor o comprar uno más. En un sistema distribuido, como ya veremos, puede que sea tan fácil como añadir más procesadores al sistema, aprovechando los anteriores (en el caso de disponer de *pools* o servidores de procesadores)

Ventajas de los Sistemas Distribuidos Frente a los Independientes.

Una ventaja de los PCs independientes frente a los sistemas centralizados es que los primeros constituyen una forma económica de trabajar. Sin embargo, aparece un nuevo problema: ¿cómo se **comparten** ahora **recursos** típicos de los sistemas centralizados, como bases de datos y periféricos? La solución consiste en conectar los PCs a un sistema distribuido, en los que, como ya sabemos, se pueden compartir todo tipo de recursos. También nos encontramos con que un sistema distribuido ofrece unos buenos mecanismos para la **comunicación entre las personas** (*e-mail, news, WWW*), los cuales, para mucha gente, tienen ciertas ventajas sobre los sistemas convencionales (cartas, fax, teléfono). Una última ventaja consiste en que los sistemas distribuidos son más **flexibles en el reparto de la carga**, pues permiten destinar estaciones de trabajo a la realización de trabajos *batch*, por ejemplo, dejando así libres más estaciones para usuarios.

Inconvenientes de los Sistemas Distribuidos.

A pesar de las ventajas de los sistemas distribuidos, también tienen sus inconvenientes. El más problemático es el del **software**, ya que actualmente no se tiene mucha experiencia con estos sistemas, con lo que no están asentadas las ideas de qué sistemas operativos, lenguajes de programación o aplicaciones son las más apropiadas para cada caso. Ni siquiera está claro cuánto deben saber los usuarios de la distribución (cuánto debe hacer el sistema y cuánto debe hacer el usuario). Esto conlleva además a la escasez de productos de software distribuido. Otro problema que se presenta es el de **la red de comunicaciones**: puede perder mensajes, saturarse, cambio de tarjetas de interfaz (por ej. por tarjetas para fibra óptica). Un último inconveniente es la **seguridad**, pues si se consigue el acceso a la red (de forma ilegal) se consigue ya un acceso sencillo a una gran cantidad de información, lo cual requiere establecer unos mecanismos de acceso muy seguros.

Aplicaciones de los S. D.

✓ APLICACIONES COMERCIALES:

- Reservas de Líneas Aéreas
- Aplicaciones Bancarias
- Cajeros de Grandes Almacenes
- Cajeros y Almacén de Cadenas de Supermercados

✓ APLICACIONES PARA REDES WAN:

- Correo Electrónico
- Servicio de Noticias (NEWS)
- Servicio de Transferencia de Ficheros (FTP)
- Búsqueda de Ficheros (Archie)
- Servicio de Consulta Textual (Gopher)
- World Wide Web (WWW)

✓ APLICACIONES MULTIMEDIA

- Videoconferencia
- Televigilancia
- Juegos multiusuario
- Enseñanza asistida por ordenador

ÁREAS DE LOS SISTEMAS DISTRIBUIDOS:

- Comunicaciones (hw. y sw.)
- Sistemas Operativos Distribuidos
- Bases De Datos Distribuidas
- Servidores Distribuidos de Ficheros
- Lenguajes de Programación Distribuida
- Sistemas Tolerantes a Fallos

La influencia de los sistemas distribuidos enseguida se ha manifestado en diversas áreas de aplicación. Comentaremos algunas de ellas.

Aplicaciones Comerciales. Las reservas de líneas aéreas, aplicaciones bancarias incluyendo cajeros automáticos, así como cajas y gestión de grandes almacenes son aplicaciones típicas históricamente construidas con hardware dedicado y alrededor de sistemas centralizados. Parece que por su inherente distribución geográfica y necesidad de acceso a sistemas distintos del propio centralizado, estas aplicaciones se prestan a implementarse dentro de sistemas distribuidos. Si nos damos cuenta, estas aplicaciones requieren ciertas características de seguridad y protección (no debería ser fácil que cualquiera sacara dinero de nuestra cuenta sin nuestro consentimiento). También se requiere fiabilidad; imaginemos que de repente se deja de poder hacer reservas en todo el mundo para los vuelos de una determinada compañía. Estas y otras características van a tener que tenerse en cuenta en los sistemas distribuidos.

Aplicaciones para Redes de Área Extensa. Dado el espectacular crecimiento de las redes de área extensa (WAN), en concreto Internet, una de las áreas que más auge ha tomado ha sido la dedicada al intercambio de información a través de la red. Así, nos encontramos con que empezó por aparecer el correo electrónico y después un servicio de transferencia de ficheros (FTP) con su correspondiente utilidad de búsqueda (*Archie*). Una variante del correo electrónico es el servicio de noticias *news*, en el que grupos de usuarios con aficiones compartidas utilizan la red como un tablón de anuncios. Un precursor del actual *World Wide Web*, me imagino que bien conocido por todos, fue *Gopher*, que ofrecía el mismo servicio que el WWW, pero en modo textual, es decir sin gráficos ni multimedia.

Aplicaciones Multimedia. Las aplicaciones multimedia y videoconferencia son las últimas en incorporarse a los sistemas distribuidos, pues por ser isócronas imponen ciertas necesidades del hardware, especialmente en lo referente a la velocidad y regularidad de transmisión de una gran cantidad de datos (como supone, por ejemplo, la videoconferencia). Estas aplicaciones incluyen también los conocidos juegos multiusuario, los cuales son muy exigentes en sus requisitos para una adecuada ejecución. Por último haremos alusión a la televigilancia, que permite una atención centralizada de diversos lugares no solamente de un mismo edificio, sino de múltiples edificios repartidos por toda una ciudad o país.

Áreas de la Informática Implicada en los Sistemas Distribuidos. Teniendo en cuenta toda esta variedad de aplicaciones de los sistemas distribuidos, su diseño va a involucrar a muy diversas áreas de la informática, entre ellas: Comunicaciones (tanto hardware como software), diseño de sistemas operativos, bases de datos distribuidas, servidores de ficheros, lenguajes de programación distribuida, y el desarrollo de sistemas tolerantes a fallos.