Arquitecturas de los Sistemas Distribuidos

Diego Alberto Rincón Yáñez MSc
drincony@poligran.edu.co
INSTITUCIÓN UNIVERSITARIA POLITÉCNICO GRANCOLOMBIANO
Facultad de Ingeniería

Agenda

Generalidades de las Arquitecturas

- Ventajas
- Desventajas

Arquitecturas

- Cliente/Servidor
- Objetos Distribuidos
- Orientado a Servicios
- Multiprocesador
- Peer to Peer

Arquitecturas de Sistemas Distribuidos

Prácticamente todos los grandes sistemas informáticos son sistemas distribuidos.

En un sistema distribuido el procesamiento de información se distribuye sobre varias computadoras en vez de estar confinado en una única máquina.

Arquitecturas de Sistemas Distribuidos

El desafío es diseñar software y hardware para proporcionar características deseables a los sistemas distribuidos y minimizar los problemas propios de ellos.

Para eso, debemos comprender las ventajas y desventajas de las diferentes arquitecturas de sistemas distribuidos.

Arquitecturas de Sistemas Distribuidos
Ventajas

1. ***Compartir recursos.*** Un sistema distribuido permite compartir recursos hardware y software (discos, impresoras, ficheros y compiladores) que se asocian con computadoras de una red.

2. ***Apertura.*** Son normalmente sistemas abiertos: se diseñan sobre protocolos estándares que permiten combinar equipamiento y software de diferentes vendedores.

Arquitecturas de Sistemas Distribuidos
Ventajas

3. ***Concurrencia.*** Varios procesos pueden operar al mismo tiempo sobre diferentes computadoras de la red. Hasta pueden comunicarse con otros durante su funcionamiento.

4. ***Escalabilidad.*** Los sistemas distribuidos son escalables mientras la capacidad del sistema pueda incrementarse, añadiendo nuevos recursos para cubrir nuevas demandas sobre el sistema.

En la práctica, si se añaden muchas computadoras nuevas, la capacidad de la red puede saturarse.

Arquitecturas de Sistemas Distribuidos
Ventajas

5. ***Tolerancia a defectos.*** Contar con varias computadoras y el potencial para reproducir información significa que los sistemas distribuidos pueden ser tolerantes a algunas fallas de funcionamiento del hardware y del software.

En la mayoría de los sistemas distribuidos, puede haber un servicio degradado, ante fallas de funcionamiento. Una completa pérdida de servicio sólo ocurre cuando existe una falla de funcionamiento en la red.

Arquitecturas de Sistemas Distribuidos
Desventajas

1. ***Complejidad.*** Los sistemas distribuidos son más complejos que los sistemas centralizados; lo que hace más difícil comprender sus propiedades emergentes y probar estos sistemas.

Por ejemplo, en vez de que el rendimiento del sistema dependa de la velocidad de ejecución de un procesador, depende del ancho de banda y de la velocidad de los procesadores de la red.

Mover los recursos de una parte del sistema a otra puede afectar de forma radical al rendimiento del sistema.

2. ***Seguridad.*** Puede accederse al sistema desde varias computadoras diferentes, y el tráfico en la red puede estar sujeto a escuchas indeseadas.

Es más difícil mantener la integridad de los datos en el sistema y que los servicios del sistema no se degraden por ataques.

Arquitecturas de Sistemas Distribuidos
Desventajas

3. ***Manejabilidad.*** Las computadoras en un sistema pueden ser de diferentes tipos y ejecutar versiones diferentes de sistemas operativos.

Los defectos en una máquina pueden propagarse a otras, con consecuencias inesperadas.

Esto significa que se requiere más esfuerzo para gestionar y mantener el funcionamiento del sistema.

4. ***Impredecibilidad.*** Los sistemas distribuidos tienen una respuesta impredecible.

La respuesta depende de la carga total en el sistema, de su organización y de la carga de la red.

Como todos ellos pueden cambiar rápidamente, el tiempo requerido para responder a una petición de usuario puede variar drásticamente, de una petición a otra.

Modos de Transmisión

MODOS DE TRANSMISIÓN

Una transmisión de datos tiene que ser controlada por medio del tiempo, para que el equipo receptor conozca en que momento se puede esperar que una transferencia tenga lugar. Hay dos principios de transmisión para hacer esto posible:

Transmisión Sincronía.

Transmisión Asíncrona.

Arquitecturas Cliente - Servidor

En una arquitectura cliente-servidor, una aplicación se modela como un conjunto de servicios proporcionados por los servidores y un conjunto de clientes que usan estos servicios.

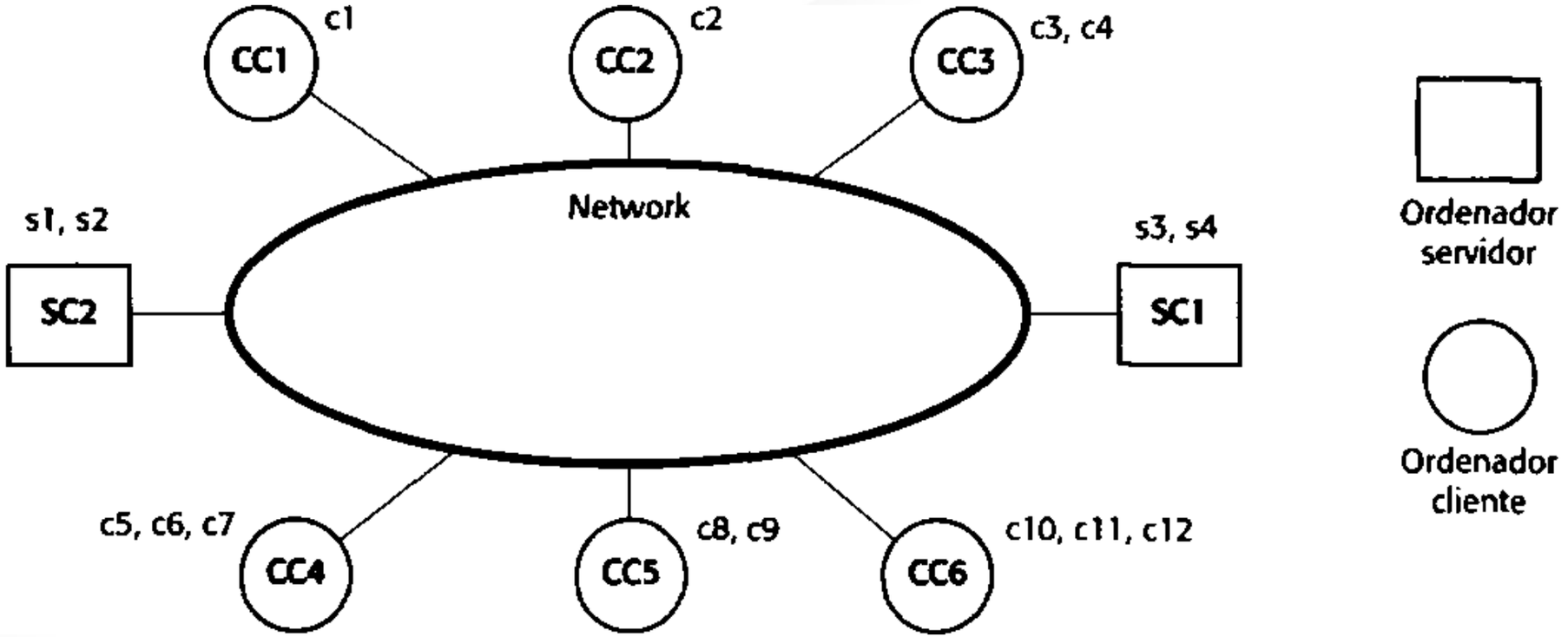
Los clientes necesitan conocer qué servidores están disponibles, pero normalmente no conocen la existencia de otros clientes.

Clientes y servidores son procesos diferentes.

Varios procesos servidores pueden ejecutarse sobre un único procesador servidor; por lo tanto, no hay necesariamente una correspondencia 1:1 entre procesos y procesadores en el sistema.

Arquitecturas Cliente - Servidor

Esta Figura muestra la **arquitectura física** de un sistema con seis **computadoras cliente** y dos **computadoras servidor**.



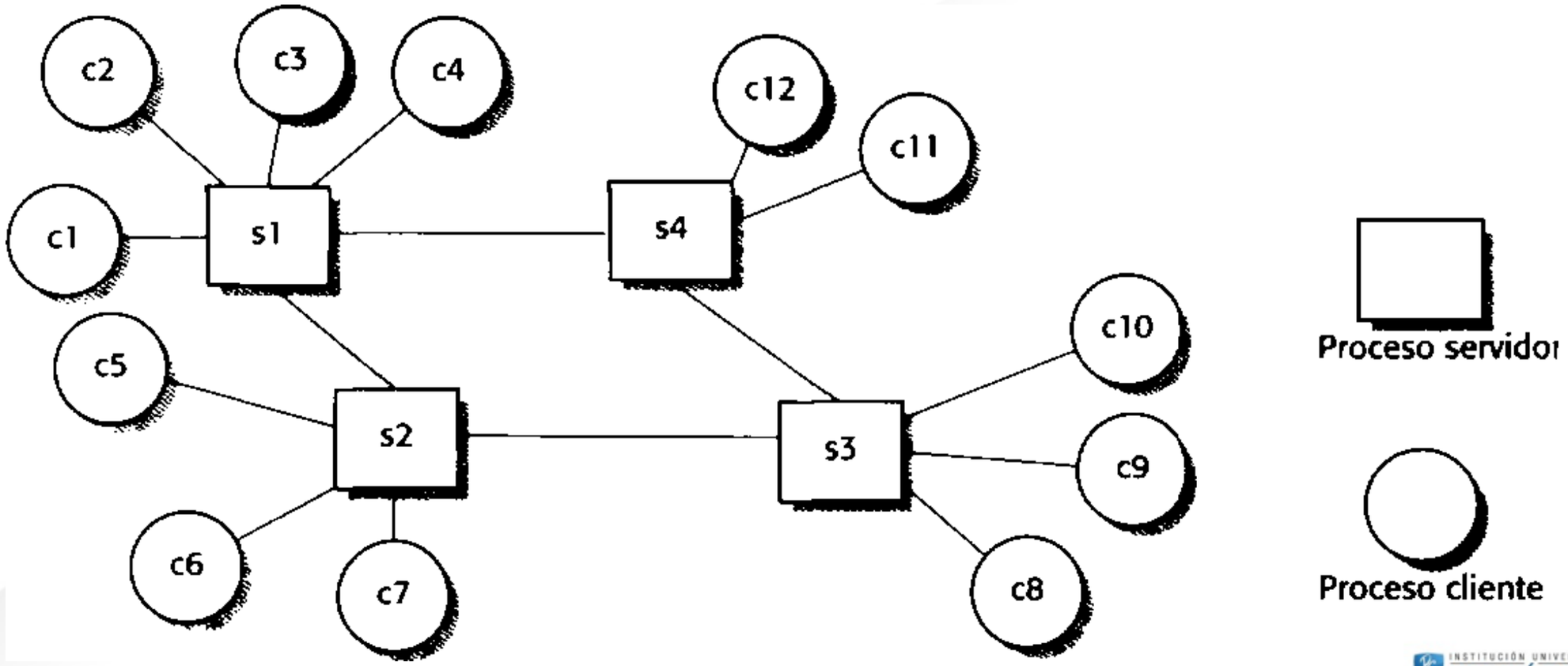
Arquitecturas Cliente - Servidor

Esa arquitectura FISICA puede ejecutar los procesos cliente y servidor que se muestran en la figura siguiente.

Acá, Cuando hablamos de ***clientes*** y ***servidores,*** nos referimos a los **procesos lógicos**, en vez de a las **computadoras físicas** sobre las que se ejecutan, como era el caso anterior.

Arquitecturas Cliente - Servidor

Sistema de "Procesos" Cliente-Servidor:



Arquitecturas Cliente - Servidor

El diseño de sistemas cliente-servidor debería reflejar la estructura lógica de la aplicación que se está desarrollando.

Una forma de ver una aplicación se ilustra en la Figura siguiente, que muestra una aplicación estructurada en tres capas.

La capa de presentación está relacionada con la presentación de la información al usuario y con toda la interacción con él.

Arquitecturas Cliente - Servidor

La capa de procesamiento de la aplicación está relacionada con la implementación de la lógica de la aplicación.

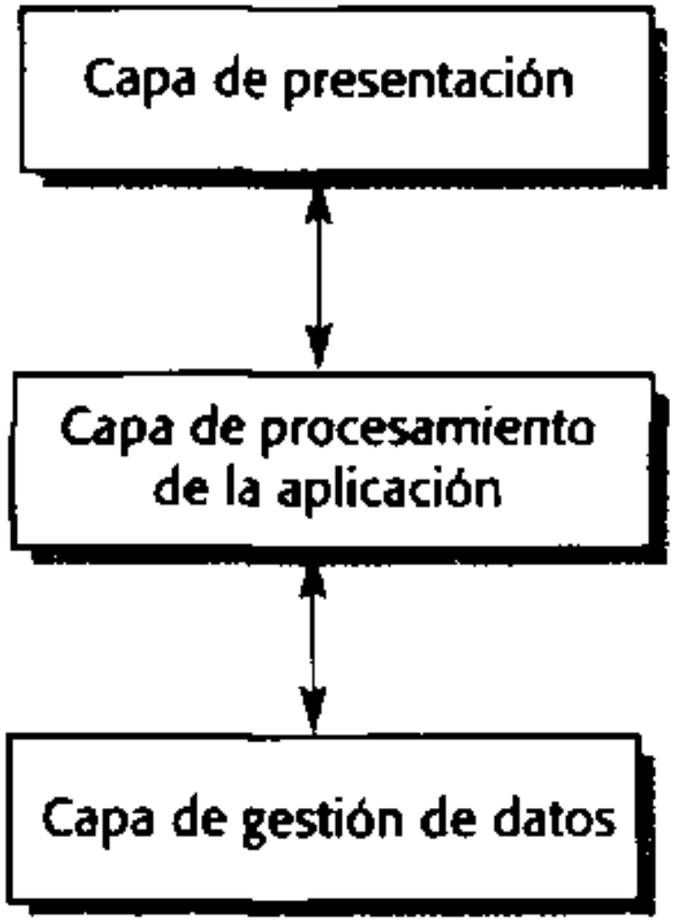
La capa de gestión de datos está relacionada con todas las operaciones sobre la base de datos.

En los sistemas centralizados, no es necesario que estas capas estén claramente separadas.

Cuando se diseñe un sistema distribuido, debería hacerse una clara distinción entre ellas, para que sea posible distribuir cada capa sobre una computadora diferente.

Arquitecturas Cliente - Servidor

Figura de Capas de las aplicaciones:



Arquitecturas Cliente - Servidor

La arquitectura cliente-servidor más simple se denomina arquitectura cliente-servidor de dos capas.

Se organiza como un servidor (o múltiples servidores idénticos) y un conjunto de clientes.

Como se ilustra en la Figura siguiente, las arquitecturas cliente/servidor de dos capas pueden ser de dos tipos:

1. ***Modelo de cliente ligero (thin-client).*** Acá todo el procesamiento de las aplicaciones y la gestión de los datos se lleva a cabo en el servidor.

El cliente simplemente es responsable de la capa de presentación del software.

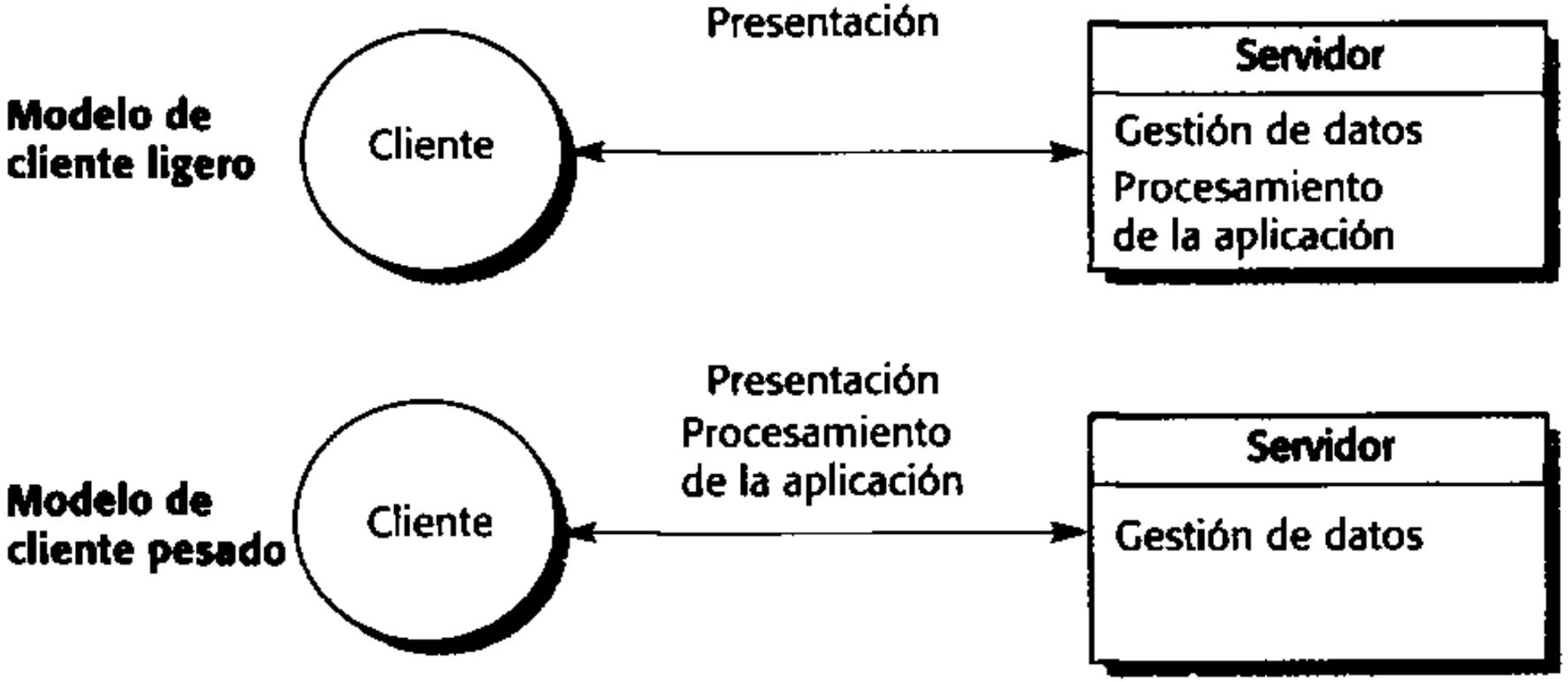
Arquitecturas Cliente - Servidor

2. ***Modelo de cliente rico (fat-client).*** En este modelo, el servidor solamente es responsable de la gestión de los datos.

El software del cliente implementa la lógica de la aplicación y las interacciones con el usuario del sistema.

Arquitecturas Cliente - Servidor

Clientes Livianos y Pesados



Modelo de Cliente Ligero/Pobre/Delgado

Una arquitectura de dos capas con clientes ligeros es la más simple que se utiliza cuando los sistemas heredados centralizados, evolucionan a una arquitectura cliente-servidor.

La interfaz de usuario para estos sistemas se migra a PC's, y la aplicación en sí misma actúa como un servidor y maneja todo el procesamiento de la aplicación y gestión de datos.

Un modelo de cliente ligero también puede implementarse cuando los clientes son dispositivos de red sencillos en lugar de PC's o estaciones de trabajo.

El dispositivo de red ejecuta un navegador de Internet y la interfaz de usuario es implementada a través de ese sistema.

Modelo de Cliente Ligero/Pobre/Delgado

Una gran **desventaja** del modelo de **cliente ligero** es que ubica una **elevada carga de procesamiento**, tanto en el **servidor** como en la **red**.

El servidor es **responsable de todos los cálculos**, y esto puede implicar la **generación de un tráfico significativo en la red** entre el cliente y el servidor.

Los dispositivos de computación modernos disponen de una gran cantidad de potencia de procesamiento, que es desperdiciada en la aproximación de cliente ligero, liviano o pobre.

Modelo de Cliente Rico/Gordo/Pesado

El modelo de cliente rico, pesado o gordo hace uso de esta potencia de procesamiento disponible y distribuye, tanto el procesamiento de la lógica de la aplicación, como la presentación al cliente.

El servidor es esencialmente de transacciones.

Gestiona todas las transacciones de la base de datos.

Ejemplo: arquitectura de los sistemas bancarios ATM (cajeros automáticos), en donde cada Cajero es un cliente y el servidor es un ***mainframe*** que procesa la cuenta del cliente en la base de datos.

Modelo de Cliente Rico/Gordo/Pesado

El **hardware** de los **cajeros automáticos** realiza una gran cantidad de **procesamiento relacionado con el cliente** y asociado a la transacción.

Los **cajeros automáticos** no se conectan directamente con la base de datos de clientes sino con un **monitor de teleproceso**.

Un **monitor de teleproceso** o **gestor de transacciones** es un **sistema middleware** que organiza las **comunicaciones** con los **clientes remotos** y **serializa las transacciones** de los clientes para su procesamiento en la base de datos.

El uso de **transacciones serializadas** significa que el **sistema puede recuperarse de los defectos sin corromper los datos** del sistema.

Modelo de Cliente Rico/Gordo/Pesado

Aunque el modelo de cliente rico distribuye el procesamiento de forma más efectiva que un modelo de cliente ligero, la gestión del sistema es más compleja.

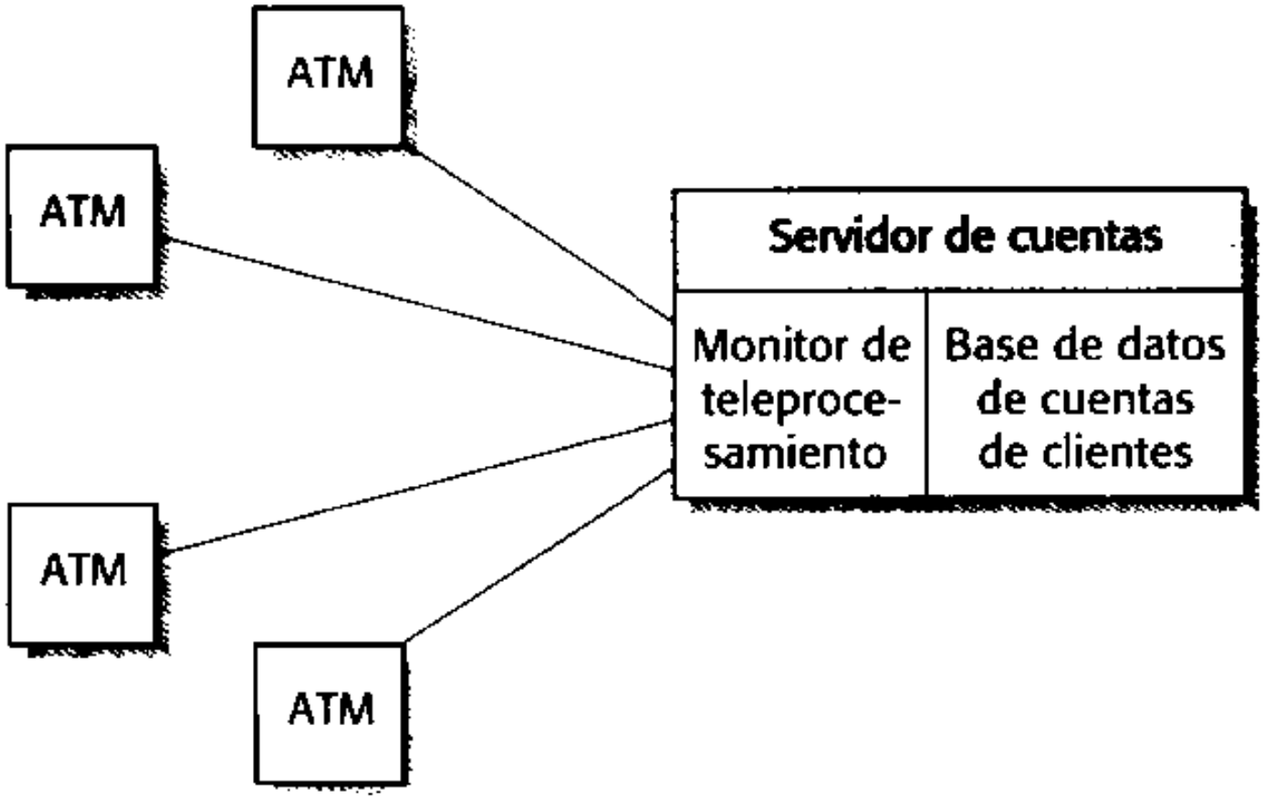
La funcionalidad de la aplicación se expande entre varias computadoras.

Cuando la aplicación software tiene que ser modificada, esto implica la reinstalación en cada computadora cliente.

Esto puede significar un costo importante si hay cientos de clientes en el sistema.

Modelo de Cliente Rico/Gordo/Pesado

Sistema Cliente – Servidor de Cajeros Automáticos (ATM):



Modelo Cliente –Servidor de 2 Capas

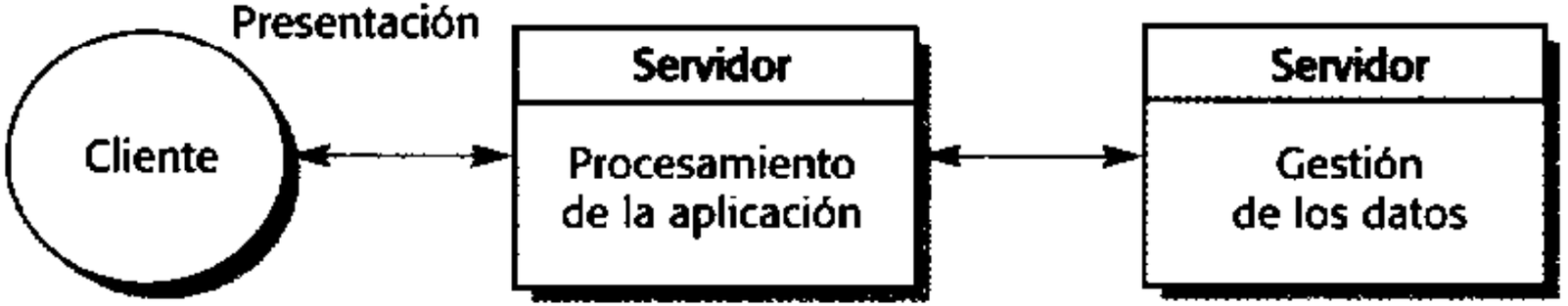
El problema con una aproximación cliente-servidor de dos capas es que las tres capas lógicas —-presentación procesamiento de la aplicación y gestión de los datos— deben asociarse con dos computadoras: el **cliente** y el **servidor**.

Aquí puede haber problemas con la escalabilidad y rendimiento si se elige el modelo de cliente ligero, o problemas con la gestión del sistema si se usa el modelo de cliente rico.

Modelo Cliente –Servidor de 3 Capas

Para evitar estos problemas, una aproximación alternativa es usar una arquitectura ***cliente-servidor de tres capas***.

Aquí, la presentación, el procesamiento de la aplicación y la gestión de los datos son procesos lógicamente separados que se ejecutan sobre procesadores diferentes.



Modelo Cliente –Servidor de 3 Capas

Un sistema bancario por Internet es un ejemplo de una arquitectura cliente-servidor de tres capas.

La base de datos de clientes del banco (usualmente ubicada sobre una computadora mainframe) proporciona servicios de gestión de datos; un servidor web proporciona los servicios de aplicación tales como facilidades para transferir efectivo, generar estados de cuenta, pagar facturas, y así sucesivamente.

La propia computadora del usuario con un navegador de Internet es el cliente.

Modelo Cliente –Servidor de 3 Capas

El sistema es escalable, porque es relativamente fácil añadir nuevos servidores web, a medida que el número de clientes crece.

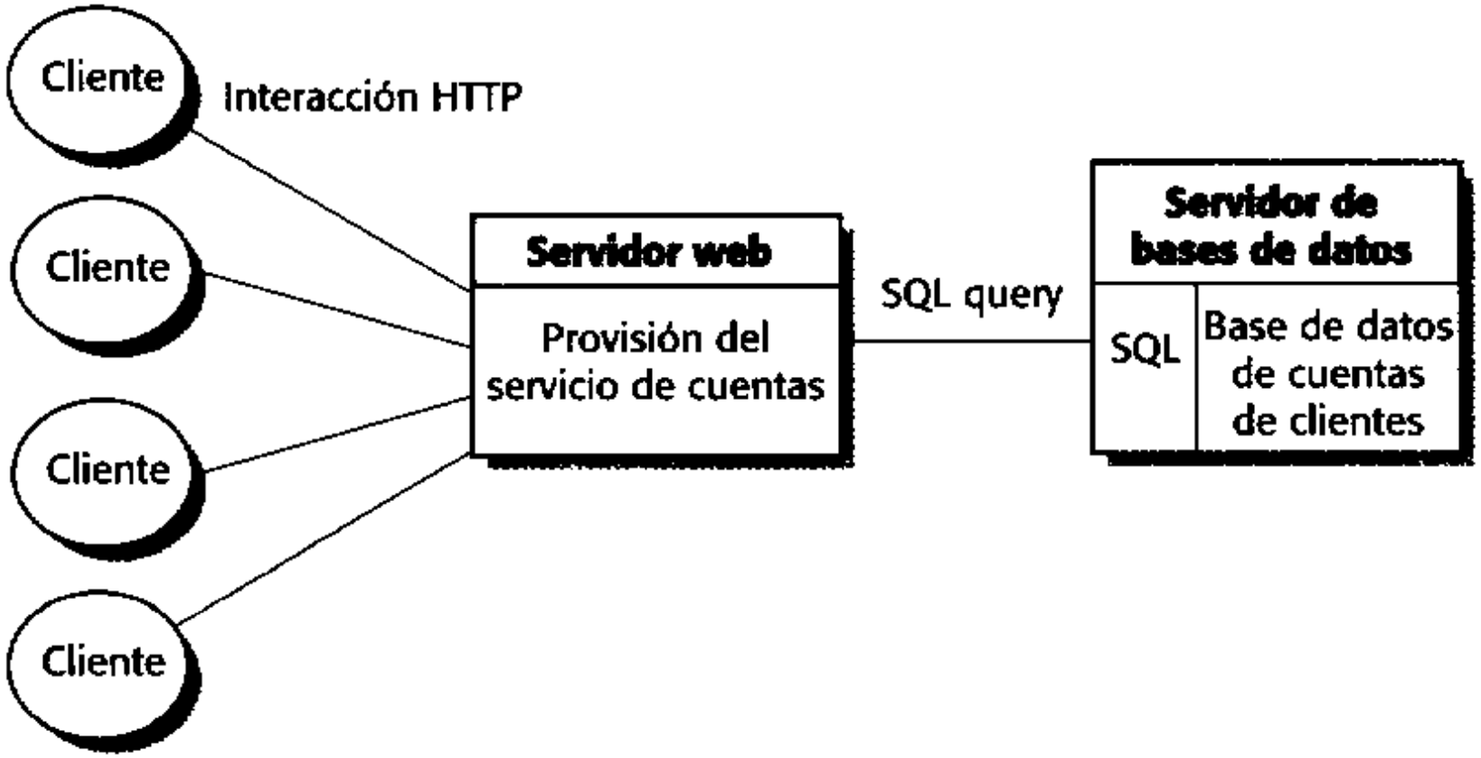
El uso de una arquitectura de tres capas permite optimizar la transferencia de información entre el servidor web y el servidor de la base de datos.

Las comunicaciones entre estos sistemas pueden usar protocolos de comunicación de bajo nivel muy rápidos.

Para recuperar información de la base de datos se utiliza un middleware eficiente que soporte consultas a la base de datos en SQL (Structured Query Language).

Modelo Cliente –Servidor de 3 Capas

Figura de Modelo de Arquitectura Cliente – Servidor de 3 Capas (Sistema Bancario en Internet)



Modelo Cliente –Servidor

Cuadro – resumen del uso de diferentes arquitecturas Cliente – Servidor:

Arquitectura C/S de dos capas con clientes ligeros	Aplicaciones de sistemas heredados en donde no es práctico separar el procesamiento de la aplicación y la gestión de los datos. Aplicaciones que requieren cálculos intensivos tales como compiladores con poca o ninguna gestión de los datos. Aplicaciones que requieran manejar una gran cantidad de datos (navegar y consultar) con poco o ningún procesamiento de la aplicación.
Arquitectura C/S de dos capas con clientes ricos	Aplicaciones en donde el procesamiento de la aplicación se proporciona por software comercial (por ejemplo, Microsoft Excel) sobre el cliente. Aplicaciones que requieren un procesamiento de datos computacionalmente intensivo (por ejemplo, visualización de datos). Aplicaciones con una funcionalidad para el usuario final relativamente estable usada en un entorno de gestión del sistema bien establecido.
Arquitectura C/S de tres capas o multicapa	Aplicaciones a gran escala con cientos o miles de clientes. Aplicaciones en donde tanto los datos como la aplicación son volátiles. Aplicaciones en donde se integran datos de múltiples fuentes.

Modelo Cliente –Servidor

A veces sirve extender el modelo cliente-servidor de tres capas a una variante multicapa en la que se **añaden al sistema servidores adicionales**.

Los sistemas multicapa pueden usarse cuando las aplicaciones necesitan acceder y usar datos de diferentes bases de datos.

Entonces, un servidor de integración se ubica entre el servidor de aplicaciones y los servidores de la base de datos.

El servidor de integración recoge los datos distribuidos y los presenta a la aplicación como si se obtuviesen desde una única base de datos.

Modelo Cliente –Servidor

Las arquitecturas cliente-servidor de tres capas y las variantes multicapa, que distribuyen el procesamiento de la aplicación entre varios servidores, son más escalables que las arquitecturas de dos capas.

El tráfico de la red se reduce, al contrario que con las arquitecturas de dos capas de cliente ligero.

El procesamiento de la aplicación es la parte más volátil del sistema, y puede ser fácilmente actualizada, porque está localizada centralmente.

El procesamiento, en algunos casos, puede distribuirse entre: la lógica de la aplicación y los servidores de gestión de datos, lo que permite una respuesta más rápida a las peticiones de los clientes.

Arquitecturas de Objetos Distribuidos

En el modelo cliente-servidor de un sistema distribuido, los clientes y los servidores son diferentes.

Los clientes reciben servicios de los servidores y no de otros clientes; los servidores pueden actuar como clientes recibiendo servicios de otros servidores, pero sin solicitar servicios de clientes.

Los clientes deben conocer los servicios que ofrece cada uno de los servidores y deben conocer cómo contactar con cada uno de ellos.

Arquitecturas de Objetos Distribuidos

El modelo **Cliente – Servidor** funciona bien para muchos tipos de aplicaciones.

Sin embargo, limita la flexibilidad del diseñador, que debe decidir dónde se proporciona cada servicio.

También debe planificar la escalabilidad y proporcionar algún medio para distribuir la carga sobre los servidores, cuando más clientes se añadan al sistema.

Arquitecturas de Objetos Distribuidos

Una opción superadora es eliminar la distinción entre cliente y servidor y diseñar una **arquitectura de objetos distribuidos**.

Aquí, los componentes del sistema son **objetos que proporcionan y requieren un conjunto de servicios**.

Otros objetos realizan llamadas a estos servicios sin hacer ninguna distinción lógica entre un **cliente** (el **receptor de un servicio**) y un **servidor** (el **proveedor de un servicio**).

Arquitecturas de Objetos Distribuidos

Los objetos pueden distribuirse a través de varias computadoras en una red y comunicarse a través de middleware.

A este middleware se lo denomina intermediario de peticiones de objetos.

Su misión es proporcionar una interfaz transparente entre los objetos.

Proporciona un conjunto de servicios que permiten la comunicación entre los objetos y que éstos sean añadidos y eliminados del sistema.

Arquitecturas de Objetos Distribuidos

Ventajas del modelo de objetos distribuido:

1) Permite al diseñador retrasar decisiones sobre dónde y cómo deberían proporcionarse los servicios.

Los objetos que proporcionan servicios pueden ejecutarse sobre cualquier nodo de la red.

Por lo tanto, la distinción entre los modelos de cliente rico y ligero es irrelevante, ya que no hay necesidad de decidir con antelación dónde ubicamos la lógica de aplicación de los objetos.

Arquitecturas de Objetos Distribuidos

2) Es una arquitectura abierta: permite añadir nuevos recursos si es necesario.

Se han desarrollado estándares de comunicación de objetos, que permiten escribir objetos, en diferentes lenguajes de programación para comunicarse y proporcionarse servicios entre ellos.

3) El sistema es flexible y escalable.

Pueden añadirse nuevos objetos, a medida que la carga del sistema se incrementa, sin afectar al resto de los objetos del sistema.

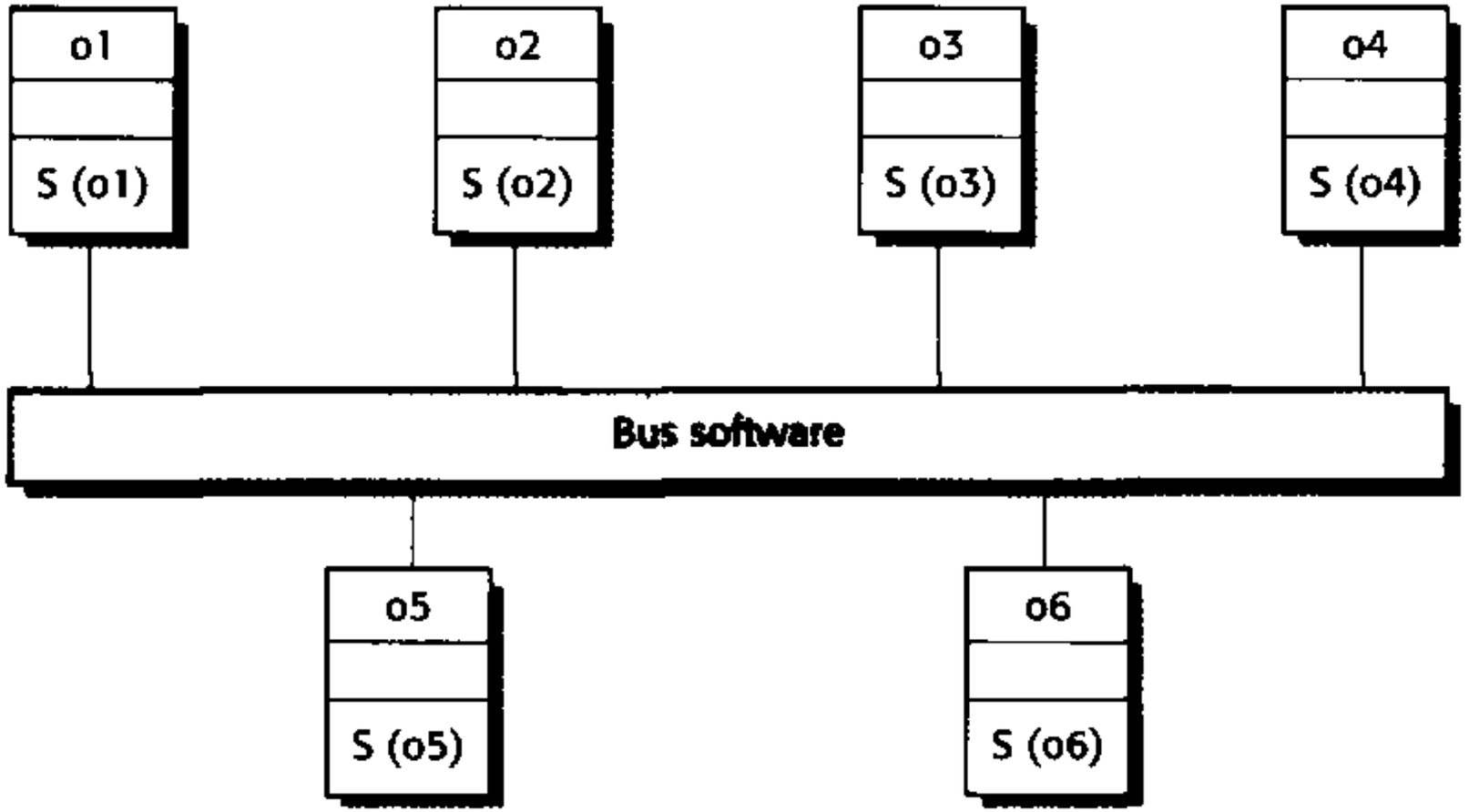
Arquitecturas de Objetos Distribuidos

4) Si es necesario, se puede reconfigurar el sistema, de forma dinámica, mediante la migración de objetos a través de la red.

Esto importa cuando haya fluctuación en los patrones de demanda de servicios.

Un objeto que proporciona servicios puede migrar al mismo procesador que los objetos que demandan los servicios, lo que mejora el rendimiento del sistema.

Arquitecturas de Objetos Distribuidos



Arquitecturas de Objetos Distribuidos

Una arquitectura de objetos distribuidos puede ser usada como un modelo lógico que permita estructurar el sistema.

Entonces, debemos pensar cómo proporcionar las funcionalidades de la aplicación únicamente en términos de servicios y combinaciones de servicios.

Después, debemos identificar cómo proporcionar estos servicios utilizando varios objetos distribuidos.

Arquitecturas de Objetos Distribuidos

La principal desventaja de las arquitecturas de objetos distribuidos es que son mucho más complejas de diseñar que los sistemas cliente-servidor.

Los sistemas cliente-servidor parecen ser la forma más natural de concebir a los sistemas.

Estos reflejan muchas transacciones humanas, en las que la gente solicita y recibe servicios de otra gente especializada en proporcionárselos.

Es más difícil pensar en una provisión de servicios generales.

Arquitectura de Sistemas orientados a Servicios

Noción de Servicio Web: Por ellos, las organizaciones que quieren hacer accesible su información a otros programas, **definen y publican una interfaz de servicio web**.

Esta interfaz define los datos disponibles y cómo se puede acceder a ellos.

Un **Servicio Web** es una **representación estándar para cualquier recurso computacional o de información que pueda ser usado por otros programas**.

La esencia de un servicio, por lo tanto, es que **la provisión de servicio es independiente de la aplicación que utiliza el servicio**.

Arquitectura de Sistemas orientados a Servicios

Los proveedores de servicios pueden desarrollar servicios especializados y ofertarlos a un cierto número de usuarios de servicios de diferentes organizaciones.

Existen varios modelos de servicios, aunque todos ellos operan de acuerdo al modelo de la Figura siguiente.

Un proveedor de servicio oferta un servicio definiendo su interfaz y definiendo la funcionalidad del servicio.

Arquitectura de Sistemas orientados a Servicios

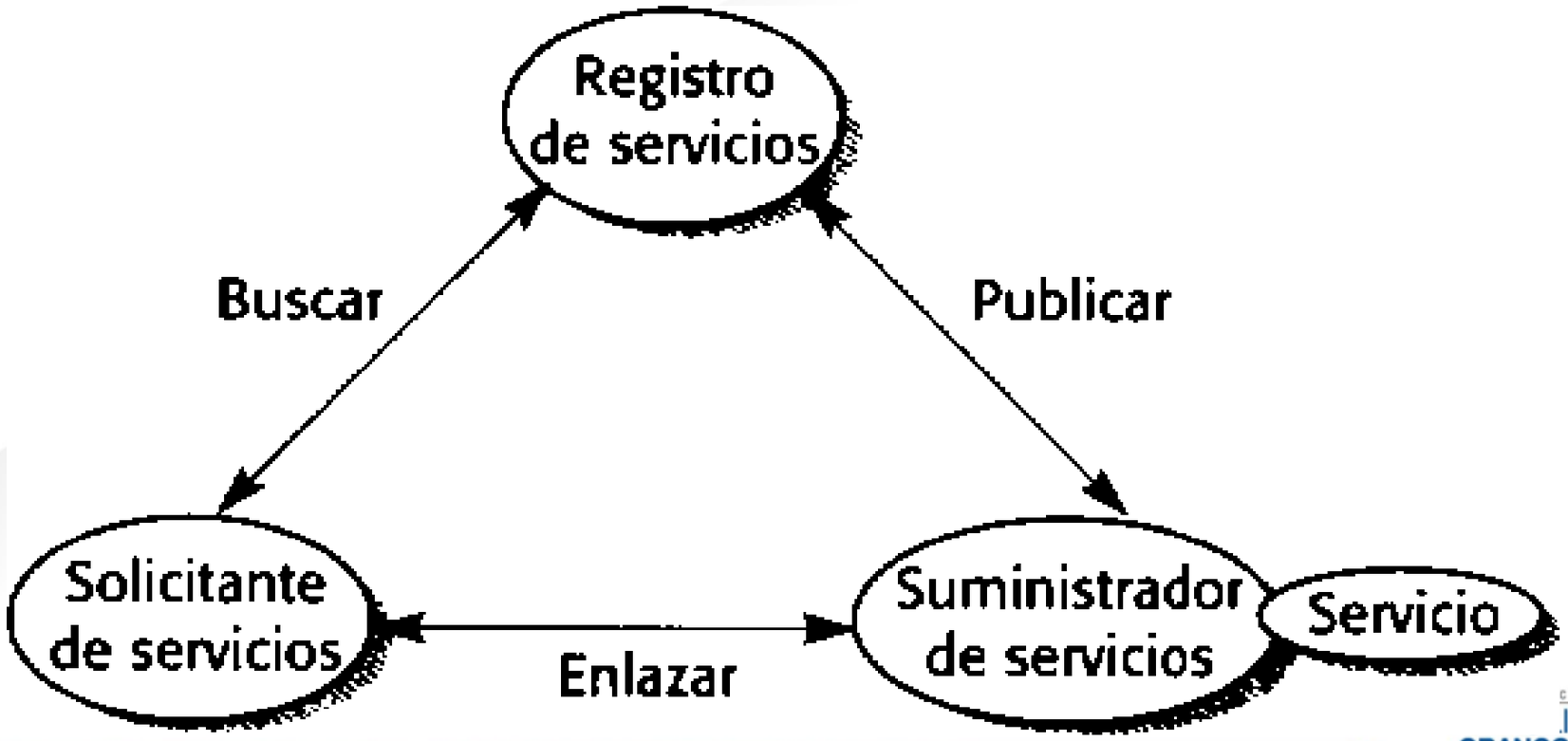
Un solicitante del servicio enlaza este servicio en su aplicación.

Es decir, la aplicación del solicitante incluye un código para llamar al servicio y procesa el resultado de esa llamada al servicio.

Para asegurar que el servicio puede ser accedido por usuarios externos, el proveedor de servicios registra una entrada en el servicio de registro, que incluye información sobre el servicio y lo que hace.

Arquitectura de Sistemas orientados a Servicios

Arquitectura Conceptual de un Sistema orientado a Servicios:



Arquitectura de Sistemas orientados a Servicios

Diferencias entre el **Modelo de Servicios** y la aproximación de **Objetos Distribuidos**:

Los servicios pueden ofertarse por cualquier proveedor de servicio dentro o fuera de una organización.

Las organizaciones pueden crear aplicaciones integrando servicios desde varios proveedores.

Por ejemplo, un compañía industrial puede enlazar directamente a los servicios de sus proveedores.

Arquitectura de Sistemas orientados a Servicios

El proveedor de servicios hace pública la información sobre el servicio para que cualquier usuario autorizado pueda usarlo.

El proveedor de servicios y el usuario de los servicios no necesitan negociar sobre lo que el servicio hace.

Las aplicaciones pueden retrasar el enlace de los servicios hasta que éstas sean desplegadas o hasta que estén en ejecución.

Es posible la construcción de nuevos servicios.

Un proveedor de servicios puede reconocer nuevos servicios que se creen.

Arquitectura de Sistemas orientados a Servicios

De este modo, los usuarios de los servicios pueden pagar por los servicios con arreglo a su uso en vez de a su provisión.

Por lo tanto, en lugar de comprar un componente de precio elevado que se usa muy raramente, el desarrollador puede usar un servicio externo por el que pagará solamente cuando sea requerido.

Las aplicaciones pueden ser reactivas y adaptar su operación de acuerdo con su entorno, enlazando con diferentes servicios según sea cada caso.

Arquitecturas Multiprocesador

El modelo más simple de un sistema distribuido es un sistema multiprocesador donde el software está formado por varios procesos que pueden (aunque no necesariamente) ejecutarse sobre procesadores diferentes.

Este modelo es común en sistemas grandes de tiempo real.

Estos sistemas recogen información, toman decisiones usando esta información y envían señales para modificar el entorno del sistema.

Arquitecturas Multiprocesador

Lógicamente, los procesos relacionados con la recopilación de información, toma de decisiones y control de actuadores podrían ejecutarse todos sobre un único procesador bajo el control de un planificador ***(scheduler).***

El uso de múltiples procesadores mejora el rendimiento y adaptabilidad del sistema.

La distribución de procesos entre los procesadores puede ser predeterminada o puede estar bajo el control de un despachador ***(dispatcher)*** que decide qué procesos se asignan a cada procesador.

Arquitecturas Multiprocesador

Un ejemplo de este tipo de sistemas:

Es un modelo simplificado de sistema de control de tráfico.

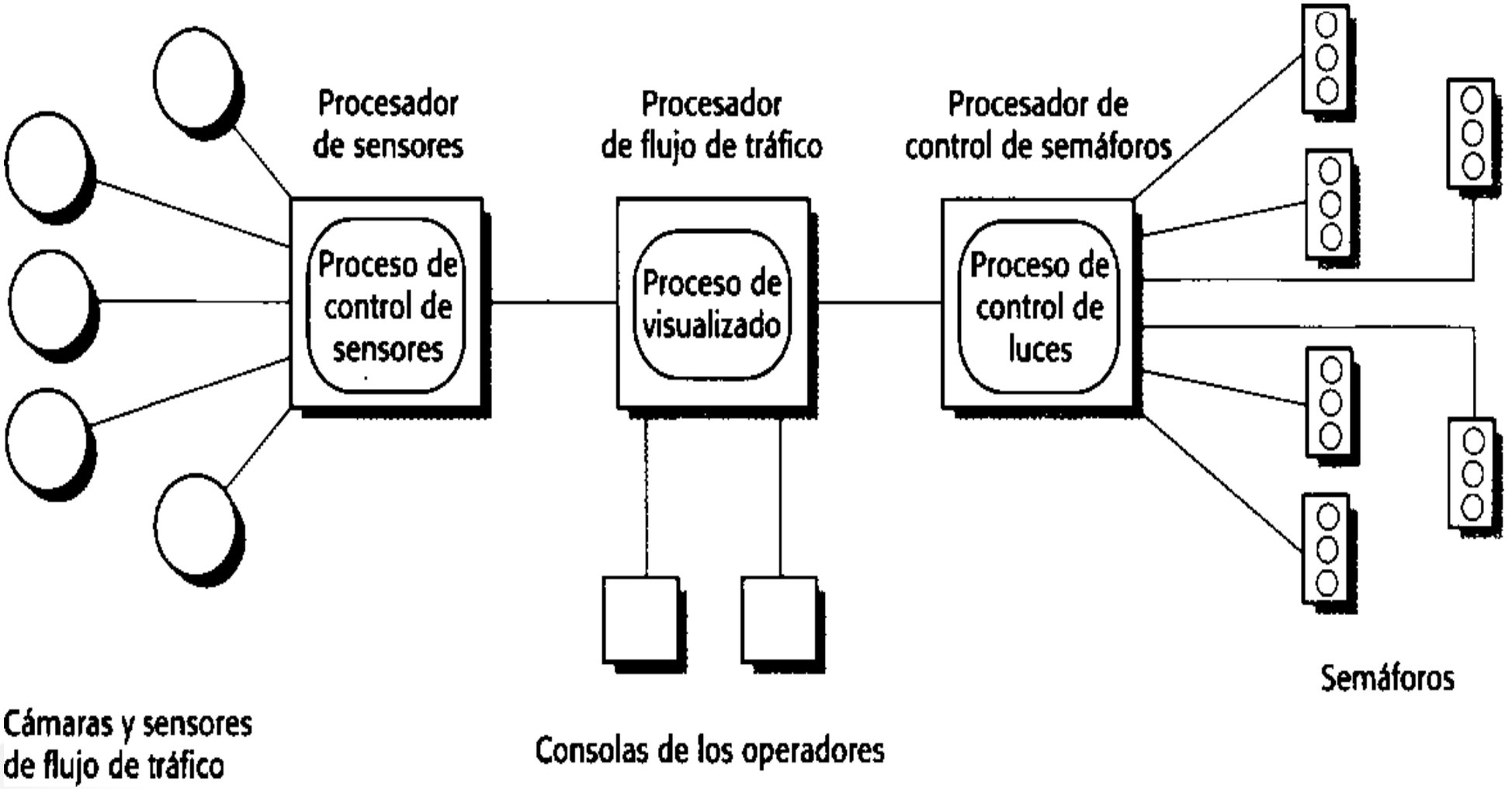
Un conjunto de sensores distribuidos recogen información sobre el flujo de tráfico y la procesan localmente, antes de enviarla a una sala de control.

Los operadores toman decisiones, usando esta información y dan instrucciones a un proceso de control de semáforos.

- ✓ Acá hay varios procesos lógicos para gestionar los sensores, la sala de control y los semáforos.
- ✓ Estos procesos lógicos pueden individuales o un grupo de procesos.
- ✓ En este caso, se ejecutarán sobre procesadores diferentes.

Arquitecturas Multiprocesador

Sistema Multiprocesador de Control de Tránsito



Arquitecturas Peer-to-Peer

Los sistemas peer-to-peer (p2p) son sistemas descentralizados, en los que los cálculos pueden llevarse a cabo en cualquier nodo de la red y, al menos **en principio**, **no se hacen distinciones entre clientes y servidores**.

En las aplicaciones peer-to-peer, el sistema en su totalidad se diseña para aprovechar la ventaja de la potencia computacional y disponibilidad de almacenamiento a través de una red de computadoras potencialmente enorme.

Arquitecturas Peer-to-Peer

Los estándares y protocolos que posibilitan las comunicaciones, a través de los nodos, están embebidos en la propia aplicación.

Cada nodo debe ejecutar una copia de dicha aplicación.

Las tecnologías peer-to-peer han sido mayormente usadas para sistemas personales.

Sin embargo, se está utilizando de forma creciente en empresas con potentes redes de PCs.

Arquitecturas Peer-to-Peer

Intel y Boeing han implementado sistemas p2p para aplicaciones que requieren computaciones intensivas.

Para aplicaciones cooperativas que soportan trabajo distribuido, es la tecnología más efectiva.

Hay dos tipos principales de arquitecturas lógicas de red que se pueden usar: arquitecturas descentralizadas y arquitecturas semicentralizadas.

Arquitecturas Peer-to-Peer

En teoría, en los sistemas peer-to-peer, cada nodo podría conocer cualquier otro nodo, conectarse con él, e intercambiar datos.

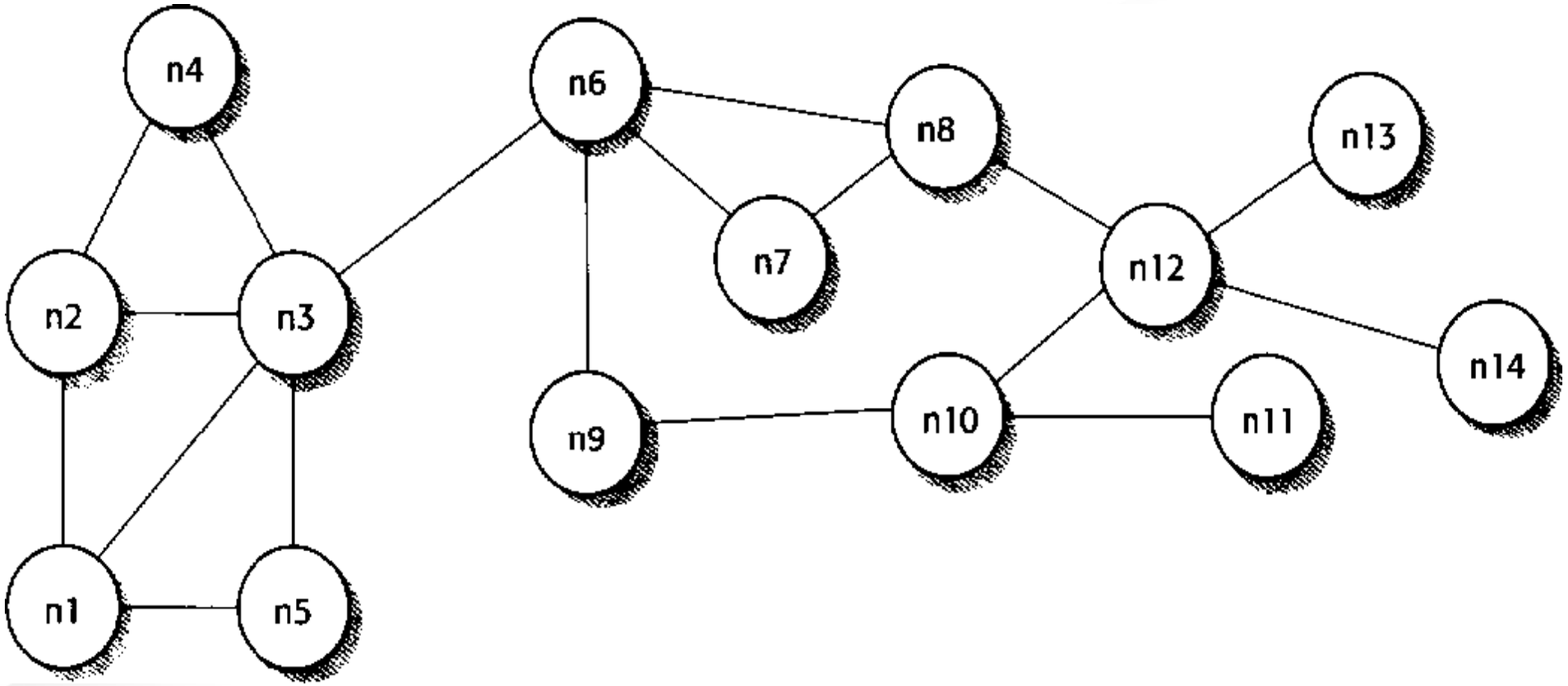
En la práctica, esto es imposible, ya que los nodos se organizan dentro de «localidades» con algunos nodos que actúan como puentes a otras localidades de nodos.

En una arquitectura descentralizada, los nodos en la red no son simplemente elementos funcionales, sino que también son interruptores de comunicaciones que pueden encaminar los datos y señales de control de un nodo a otro.

En la figura siguiente, si **n1** debe buscar un archivo que está almacenado en **n10**, esta búsqueda se encamina a través de los nodos **n3**, **n6** y **n9** hasta llegar a **n10**.

Arquitecturas Peer-to-Peer

Arquitectura p2p descentralizada:



Arquitecturas Peer-to-Peer

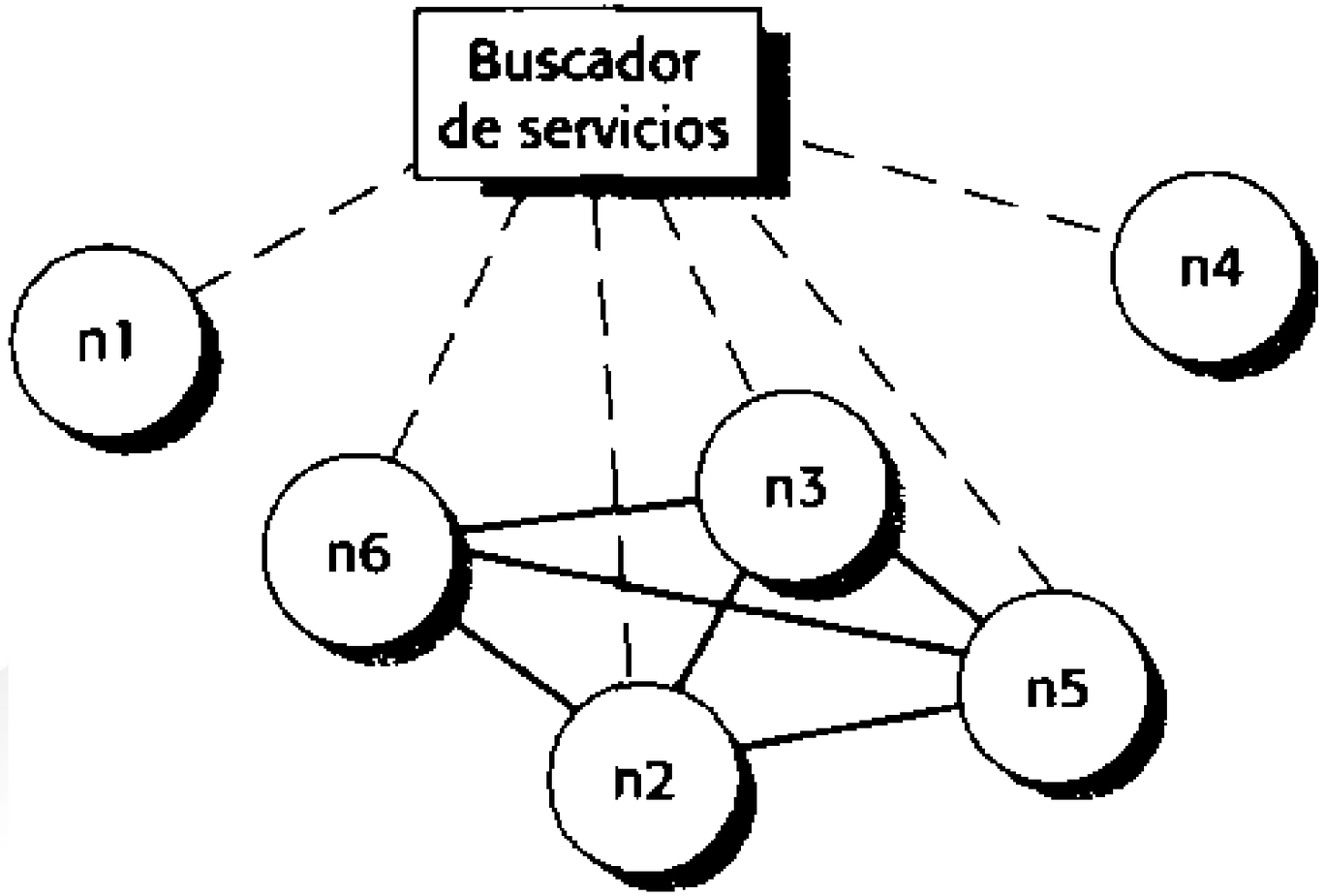
Esta arquitectura descentralizada tiene ventajas: es altamente redundante, y tolerante a defectos y a nodos desconectados de la red.

Sin embargo, existen sobrecargas obvias en el sistema ya que la misma búsqueda puede ser procesada por muchos nodos diferentes y hay una sobrecarga significativa en comunicaciones.

Un **modelo de arquitectura p2p alternativo** que parte de una **arquitectura p2p pura** es una **arquitectura semicentralizada** en la que, **dentro de la red, uno o más nodos actúan como servidores para facilitar las comunicaciones entre los nodos**.

Arquitecturas Peer-to-Peer

Arquitectura p2p SemiCentralizada



Arquitecturas Peer-to-Peer

En una **arquitectura semicentralizada**, el papel del servidor es **ayudar a establecer contacto entre iguales en la red o para coordinar los resultados de un cálculo**.

Por ejemplo, si la Figura anterior representa un **sistema de mensajería instantánea**, entonces los **nodos de la red** se comunican con el **servidor** (indicado por **líneas punteadas**), para encontrar **qué otros nodos están disponibles**.

Una vez que éstos son encontrados, **se pueden establecer comunicaciones directas** y **la conexión con el servidor es innecesaria**.

Por lo tanto, **los nodos n2, n3, n5 y n6 están en comunicación directa**.

Arquitecturas Peer-to-Peer

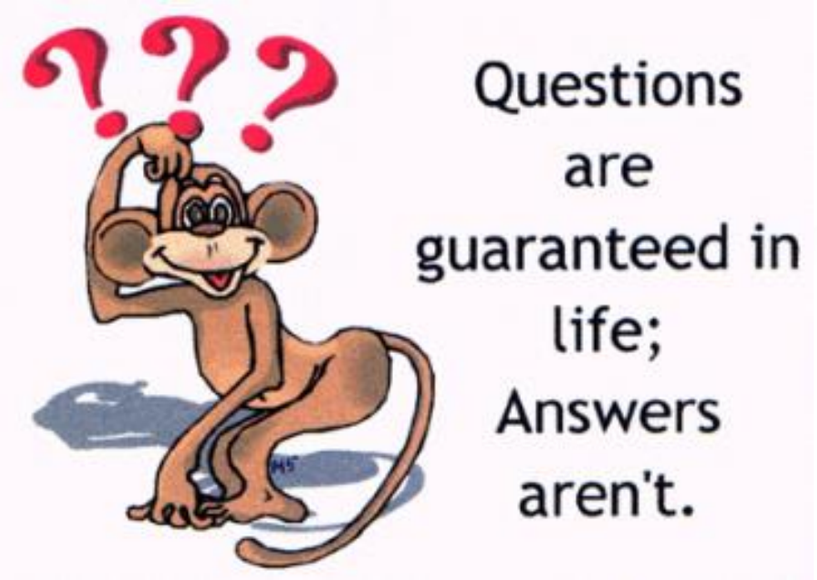
En un sistema p2p, donde un cálculo (que requiere un uso intensivo del procesador) **se distribuye a través de un gran número de nodos**, es normal que se distingan **algunos nodos cuyo papel es distribuir el trabajo a otros nodos y reunir y comprobar los resultados del cálculo**.

Si bien hay **sobrecargas obvias** en los sistemas peer-to-peer, **éstos son una aproximación mucho más eficiente para la computación interorganizacional que la aproximación basada en servicios**.

Arquitecturas Peer-to-Peer

Todavía hay problemas en el uso de las **arquitecturas p2p**, ya que cuestiones tales como la **protección** y la **autenticidad** **no están del todo resueltas**.

Esto significa que los sistemas p2p se usan más en sistemas de información no críticos.



¿Preguntas?

Diego Alberto Rincón Yáñez MCSc.
Twitter: @d1egoprog.


INSTITUCIÓN UNIVERSITARIA
POLITÉCNICO
GRANCOLOMBIANO
¡GRACIAS!
POLI.EDU.CO
Poligran
MIEMBRO DE LA RED
ILUMNO