TEMA 1

MOTIVOS

Aumentar la capacidad de procesamiento subiendo la velocidad del procesador es costoso. Es mejor interconectar más máquinas, pero requiere compartir recursos con sistemas operativos distribuidos.

Progreso Histórico

Sistemas por lotes

Tiempo compartido Mainframe y terminales tontas

Teleprocesos Las consolas son sistemas personales poco potentes

Sistemas personales

Sistemas en Red

Sistemas distribuidos

Sistemas distribuidos

Ordenadores independientes que aparentan ser ser un único sistema. Transparencia de ubicación, local o remoto.

Middleware Capa de abstracción intermedia para proporcionar una interfaz común.

Ventajas	Desventajas
Económico, escalable, flexible, alta disponibilidad, paralelismo, acceso transparente, seguro.	Alto coste de mantenimiento, bajo rendimiento individual, compleja gestión de recursos, la red es siempre el factor limitante.

Sistemas ubicuos Un sistema que no se percibe. EJ: computador de abordo

PROPIEDADES

Transparencias Los recursos parecen de una sola máquina. Tipos:

Por identificación Independencia de rutas de red y sistema de ficheros.
EJ: N1:/usr/R3 => N1:R3

De ubicación Independencia del nodo, pueden moverse.
EJ: N1;/usr/R3 => /usr/R3

De replicación No se conoce cuantas copias hay de un recurso.
Recolocación: Los recursos se mueven sin afectar su uso.
Replicación: No se sabe, pero se hacen copias.

Paralelismo Recursos compartidos sin que el usuario lo sepa. (Compartición)
Concurrencia con equivalencia al uso en serie

Rendimiento El servicio no se degrada sin importar la carga.

Fallos Se ocultan los fallos, causas y recuperaciones.

Persistencia Se oculta si un recurso está en memoria o en disco.

Escalabilidad Capacidad de crecer sin afectar al rendimiento

Espacio de nombres Tamaño suficiente del espacio de nombres (combinaciones para un identificador). La jerarquía hace escalable el espacio de nombres.

Mantener el rendimiento Es costoso.
Mirroring: Duplicado geográfico de recursos.
Caching: Almacenamiento local, reduce el uso de la red.

Fiabilidad Capacidad para realizar en todo momento las funciones para las que se diseñó.

Disponibilidad Medida con el tiempo medio entre fallos o reparaciones.

Tolerancia a fallos Capacidad de seguir funcionando (recuperarse) tras el fallo de un componente. La replicación por si sola no es suficiente, debe ser de forma transparente.

Flexibilidad Capacidad del sistema operativo distribuido para actualizarse al ritmo de la evaluación tecnológica.

Kernel Monolítico	Micro Kernel
El núcleo tiene todos los servicios. Es rápido al no tener que comunicarse entre sí con mensajes. Una actualiazciñon requiere parar el sistema.	El núcleo solo tiene los servicios fundamentales, entre ellos la comunicación de procesos. Son más lentos ya que los servicios se implementan en la capa de usuario. Se puede actualizar rápidamente sin reiniciar.

HARDWARE DE LOS SISTEMAS DISTRIBUIDOS

Clasificación de Flynn

SISD Una sola instrucción, un solo dato. Von Neumann clásico.

SIMD Una sola instrucción, múltiples datos. Computadores vectoriales.
Hay una sola unidad de control, la misma operación se aplica a múltiples datos. Distintas unidades de ejecución y memoria.

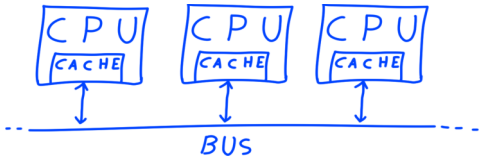
MSID Múltiples instrucciones, un solo dato. Computador segmentado, DLX.
Diferentes unidades de procesos ejecutan instrucciones a la vez en distintos fases de la instrucción que requieren de segmentos independientes de la arquitectura.

MIMD Múltiples instrucciones, múltiples datos. Cada unidad de ejecución con su unidad de control. Para leliza totalmente las instrucciones. Se clasifican según su grado de acoplamiento.

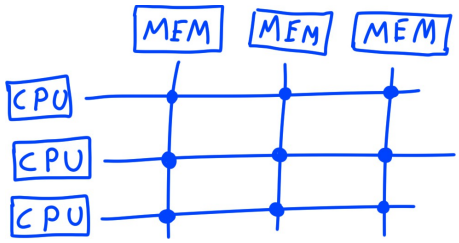
Multiprocesadores

Fuerte acoplamiento. Se diferencian por el método de conexión.
UMA: Memoria y buses compartidos. (Acceso a memoria unificado)
NUMA: Comparten memoria pero no buses (Acceso a memoria no unificado)

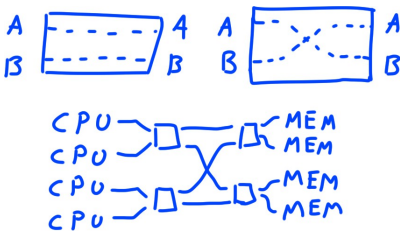
BUS La memoria y los procesadores se conectan a un único BUS. No escala bien. aunque mejora agregando cache.



Conmutados Se usan a partir de 64 procesadores. Divide la memoria y se conecta en malla con conmutadores. Si dos CPUs necesitan el mismo módulo de memoria, deben esperar. Es posible que cada uno acceda a un módulo de memoria diferente.

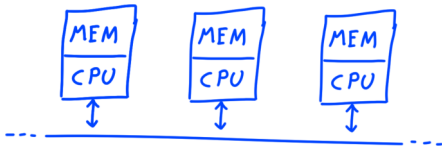


Conmutadores OMEGA Tardan un tiempo y se requieren en grandes cantidades y altas velocidades. No escalan bien. Se necesitan nºCPU x nºMEM. Alto coste. Pero no siempre se requiere conectividad completa.



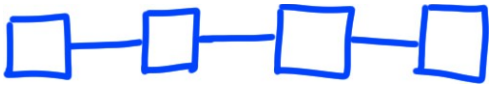
Multicomputadores

Cada CPU tiene una memoria, BUS compartido Se usa cuando quieren compartir información. EJ: Estaciones de trabajo en red Ethernet.

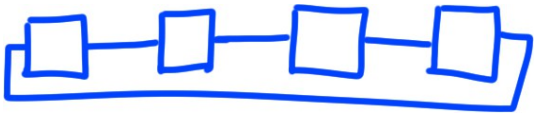


Conmutados Los nodos se conectan entre sí en una estructura regular. Permite delimitar el retorno de los mensajes. Depende del número de enlaces a atravesar como máximo, el diámetro de la red. Topologías:

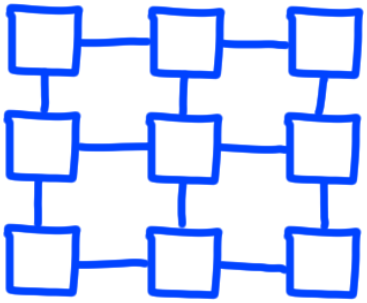
Lineales: Diámetro nºNodos – 1. EJ: 3



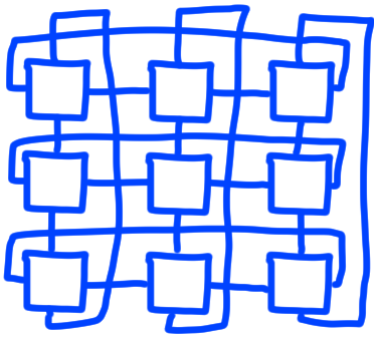
Anillo: Diámetro [nºNodos / 2] EJ: 2 ([] = floor (truncar))



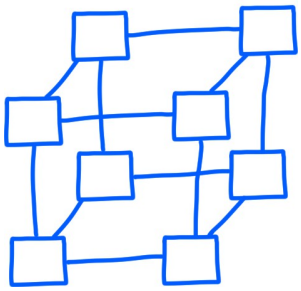
Bidimensional (Solo cuadrados) Diámetro: 2(nºNodosLado – 1) EJ: 4



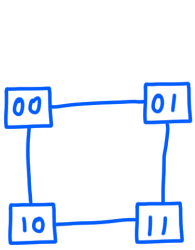
Toro (Solo cuadrados) Diámetro: 2[nºNodosLado / 2] EJ: 2

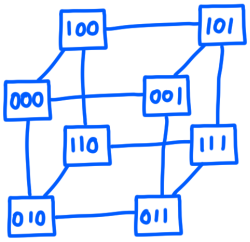


Malla N-Dimensional Diametro: nºDimensiones * (nºNodosDimension – 1)
EJ: 3 dimensiones, 3 nodos por dimensión, diámetro 6.



Hipercubo: Malla N-Dimensional. Es un hipercubo de orden n. Cada nodo tiene n conexiones. Hay 2^n nodos. Propiedad: Si cada nodo se identifica con bits de 0 a 2^n -1, un nodo se conecta con los que tienen un identificador que difiere en un solo bit.





SOFTWARE DE LOS SISTEMAS DISTRIBUIDOS

Heterogeneidad

Es complicado ofrecer un servicio único cuando los nodos cuentan con Hardware y Software distintos y están en diferentes partes del mundo. Peor si los componentes y el SO son propietarios.

Sistema abierto Sistema abierto: La especificación es pública, los fabricantes lo implementan. EJ: Convergencia al protocolo TCP/IP.

Características

Interoperatividad Capacidad para mover información entre nodos.
Protocolo estándar: TCP/IP
Lenguaje de definición de interfaces: IDL, SOAP

Transportabilidad de aplicaciones Capacidad para mover programas (no ejecutables) entre máquinas.
Posix: Código fuente compatible entre máquinas UNIX.

Java: Microcódigo entre máquinas virtuales (JVM)

Transportabilidad de usuarios Un usuario puede cambiar de nodo de acceso sin tener que conocer sus detalles.

Comunicación

Diferencia entre la distribución lógica y física. Depende del acoplamiento.

Variables compartidas MEM Física compartida y modelo de MEM compartida.

Buzones, pipes MEM Física compartida. Requiere paso de mensajes.

RMI MEM Física distribuida. Se comporta como si fuera compartida.

RPC MEM Física distribuida. Requiere paso de mensajes. No enmascara la MEM.

Implementación de la comunicación

Diferencia entre la distribución lógica y física. Depende del acoplamiento.

Modelo Cliente-Servidor El cliente envía un mensaje al servidor solicitando un servicio, el servidor espera peticiones de los clientes y las satisface. La distribución de mensajes la realiza el núcleo (Primitivas).

Bloqueantes: El envio espera hasta la recepción y liveración del canal.
No bloqueantes: Los mensajes se almacen, continua sin comprobar el éxito.

Fiable: TCP orientado a conexión. Tiene sobrecarga y es costoso. Pero el emisor puede estar seguro de que lo que envía se recibo sin pérdidas y en el orden correecto.
No fiable: UDP basado en datagramas y sin conexión. Es rápido y ligero. No garantiza la llegada ni el orden. (Eso es cosa del usuario)

Punto a punto: Solo dos nodos conocidos.
Broadcast: Una dirección especial que llega a todos los nodos de la red.
Multicast: Una dirección que engloba un conjunto de dispositivos.

Comunicación de grupos Se pretende poder enviar un mensaje a un conjunto de nodos que forman un grupo. El mensaje llega a todos los miembros a la vez o no llega a ninguno. Es atómico y con ordenación temporal. No resulta tan sencillo:
Los grupos pueden ser dinámicos, el mecanismo debe ser transparente, un proceso puede pertenecar a varios grupos. Se basa en broadcast o multicast.

SINCRONIZACIÓN DE SISTEMAS DISTRIBUIDOS

Se debe evitar la centralización pero es necesario tener una noción del tiempo para ordenar eventos. Es dificil obtener un tiempo exacto en distintos nodos.

Todos los relojes de los nodos se adelantan o atrasan.

Centralizar el servicio de tiempo en un nodo

Un servidor fiable que se sincroniza con UTC provee del tiempo a los demás. Parte del retardo es constante pero otra parte depende de la carga. La latencia de la red hace que sea impreciso. Si cae todo el sistema distribuido se bloquea.
Algoritmo de Cristian: Suponer que la hora recibida fue obtenida en la mitad entre que se enviá y que se recibe la solicitud y respuesta.

Tiempo distribuido

Los nodos se sincronizan entre ellos. Se usa el tiempo de cada nodo para generar un nuevo tiempo global que es distribuido. Solo es válido para ordenación.
Principio estadístico, la media de todos los relojes es más precisa que cualquiera de los relojes por separado.
EJ: Reloj lógico.

Sincronización externa Se usan tiempo de freferencia, proporciona el instante preciso en el que ocurre un evento.

Sincronización interna Se usan los relojes de cada nodo para medir el tiempo entre eventos en nodos distintos.

La sincronización externa requiere de la interna. Puede haber interna sin externa pero no externa sin interna. Igualar la hora local con la UTC real (Reloj físico).

Un receptor UTF en cada nodo es demasiado caro, se usan algoritmos de sincronización interna.

Algoritmo de Berkeley

El coordinador solicita el tiempo de todos los nodos.

Se calcula la media desechando las respuestas demasiado desviadas o que llegaron con demasiada latencia.

Se calcula y envía la desviación de cada nodo.

$$T = \frac{\sum_{i=1}^{n} \left(t_i - \frac{D_i}{2}\right)}{n} \quad ; \quad \Theta_i = t_i - \frac{D_i}{2} - T$$

Compensación de desvíos

Negativa El reloj está atrasado, se puede adelantar.

Positiva El reloj está adelantado, no se puede atrasar. Hay que relentizarlo poco a poco para ajustarlo. Un reloj no puede retroceder.

DTS: Distributed Time Server. Algunos nodos se sincronizan con UTC. Otros obtienen la hora de los que tienen sincronización externa. Se toma la intersección de los tiempos (con más o menos tolerancia) y se usa el tiempo medio de la mayor intersección.

Tiempo lógico

Para la ordenación de eventos. Cuando la precisión del reloj físico no es suficiente, puede que se distorsione la causalidad.

Propiedades Forma de ordenar eventos.

Si X e Y son eventos del mismo proceso, se comparan por el tiempo físico. $X \rightarrow Y \Leftrightarrow T(X) < T(Y)$

Si X es el evento de envió de un mensaje e Y es el evento de recepción de dicho mensaje. $X \rightarrow Y$

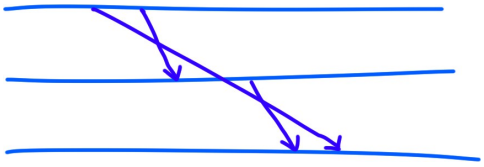
Transitividad. Si $X \rightarrow Z$, $Z \rightarrow Y$ entonces $X \rightarrow Y$.

Cuando no hay relaciones de causalidad son concurrentes $X \parallel Y$.

Reloj de Lamport Todos los procesis tienen un reloj que comienza en 0. Cuando producen un evento incrementan el contador. Al recibir un mensaje toman el valor del reloj del proceso que enviá el mensaje solo si este es mayor y lo incrementan el cualquier caso. No siempre mantene el orden entre procesos.

Vectores de tiempoo de Lamport Todos los procesos tienen un vector con tantos elementos como procesos hay en el sistema. Inicialmente todo el vector de todos los procesos está a 0. Un proceso incrementa la posición de su vector de tiempo correspondiente a su número de proceso cuando realiza un evento. Al recibir un mensaje se actuaran las posiciones del vector que sean menores con respecto al recibido (en los mensajes se incluye el vector de tiempo del proceso emisor). Un evento es anterior a otro si al menos una de sus posiciones del vector de tiempo es menor y el resto es menor o igual. Son concurrentes cuando todas son iguales o una es mayor y otra es menor son.

Entrega casual: No se puede recibir un mensaje anterior después de otro posterior. Se soluciona con una entrega FIFO o vectores de tiempo.



EXCLUSIÓN MUTUA DISTRIBUIDA

Algoritmos de exclusión mutua

Evitar que se produzca inconsistencia por que múltiples procesos accedan a la zona crítica a la misma vez. Con memoria compartida se utilizan semáforos o monitores.
Para sistemas distribuidos se necesitan algoritmos que no requieran memoria compartida.

Algoritmo centralizado

Un proceso controlador gestiona el recurso. Los procesos solicitan el recurso al coordinador. Si está libre se concede, si no se encola en una FIFO. Puede haber cuello de botella por el coordinador. Es fácil de implementar pero si el coordinador falla o el cliente muere antes de liberar el recurso se bloquea el acceso.

Algoritmo de Ricart-Agrawala

Se trata de un orden total de eventos. Cuando un proceso quiere acceder a la sección crítica notifica de sus intenciones a todos los procesos. Al recibir una notificación:

Si el proceso está en la SC: No confirma, no enviar nada es negar una confirmación.

Si el proceso está esperando para la SC: Solo confirma si el tiempo del solicitante es menor que el suyo, si no encola la petición y la contesta cuando salga de la SC.

Si no está en la SC y no quiere entrar: confirma inmediatamente.

Un proceso solo entra en la SC si todos los procesos le dan confirmación.

Algoritmo en anillo

Los nodos se conectan en Token Ring, todos tienen un anterior y un siguiente. El token solo se mueve en un sentido. Un proceso solo accede a la SC mientras esté en posesión del token.

Cuando un proceso recibe el token pero no necesita entrar en la SC lo envía al siguiente. Inicialmente cualquiera puede tener el token.

Requiere menos mensajes que Ricart-Agrawala.

Pero se producen muchos mensajes cuando nadie quiere entrar.

El fallo de un nodo provoca el bloqueo al detenerse el paso de testigo. No se puede detectar si es un fallo o hay un proceso acaparando la SC.

Corrección del Algoritmo en anillo con numeración del token

En el token se guarda el valor del reloj lógico de cada proceso de la última vez que

obtuvieron el token. Las peticiones se realizan a todos los nodos, se envía el reloj lógico de ese nodo que solicita. Cuando un proceso sale de la SC envía el token al primer proceso esperando. Un proceso está esperando si el valor de su reloj lógico en el token es menor que el de su petición.
No se rompe a menos que falle el nodo en posesión del token. Solo se necesitan tantos mensajes como nodos haya para entrar en la SC.

Algoritmos de elección

Cuando se necesita que un proceso coordine una operación. La elección requiere el conceso de todos los procesos del sistema. La elección se repetirá si el coordinador falla. Debe existir una ordenación total de procesos y los mensajes tienen un tiempo máximo para ser entregados.

Bully (Matón)

Comienza una nueva elección cuando se sospecha que el controlador ha fallado, cuando el tiempo de espera excede dos veces el tiempo de propagación más del de procesamiento.
Un proceso envía un mensaje a todos los procesos con un identificador superior al suyo. Si un proceso no recibe ningún mensaje avisa al resto que él es el nuevo coordinador. Todos los mensajes se confirman, puede necesitar nºProcesos ^ 2 mensajes.

Anillo

Cuando un proceso detecta que el coordinador ha fallado envía un mensaje de elección al siguiente. Si es menor, solo envía su código si no se trata de un nodo parcipante. Un proceso se autoselecciona y avisa a los demás cuando recibe un mensaje con su identificador. (Porque nadie ha podido superarlo en una vuelta completa). Necesita 2 * nºProcesos o 3 nºProcesos mensajes.

Mejora del anillo

Se envía un mensaje con el identificador de todos los participantes. Al llegar de vuelta al origen se selecciona al de mayor identificador. No falla si varios nodos comienzan el proceso porque solo se envía el más alto y el menor sale de circulación.

TRANSACCIONES ATÓMICAS

Evita dejar el sistema en un estado inconsistente tras el fallo de una operación no indivisible.
No se ven afectadas por otras operaciones concurrentes, equivalencia con serie.
Si terminan con un fallo no se mantienen los efectos que haya tenido. Nunca se quedan a medias y nunca se interrumpen. (Esto segundo es solo a efectos prácticos)
Se requieren primitivas para comenzar, finaliza y abortar una operación.

Se deben cumplir las reglas ACID

Atomicity No existen estados intermedios porque la operación se realiza completamente o no se realiza.

Consistency Los datos nunca quedan inconsistentes. Si una operación falla se deshacen los cambios.

Isolation El aislamiento garantiza que la ejecución concurrente de transacciones es equivalente a una ejecución serie. No puede ser serie por la perdida de rendimiento.

Durability Los cambios confirmados son permanentes al finalizar una operación.

Implementación con un log de eventos

Se guarda un registro de los valores antes y después de los cambios.
Al empezar una transacción se marca con una etiqueta. Si se finaliza correctamente se cierra con otra etiqueta. Si se produce un error, se deshace a los valores anteriores hasta llegar al comienzo de la transacción.

También es posible realizar una copia de los datos, trabajar sobre la copia y volcar la a la original solo si la transacción termina correctamente.

Transacciones distribuidas

Se necesita un coordinador. Los éxitos y fracasos son globales.
El compromiso se realiza en dos fases.

> El coordinador envía un mensaje “Preparate” a cada trabajador.
> Los trabajadores responde con un “Preparado” o “Abortar”
>
> Un solo “Abortar” es suficiente para abortar la transacción.
> El coordinador informa a los trabajadores con “Comprometido” o “Abortado”.
> Los trabajadores responde con “Terminado” indistintamente.

Gracias al registro es posible recuperar el estado aunque falle el coordinador o un trabajador.

TEMA 2

TIPOS DE ARQUITECTURAS

Una arquitectura es un diseño conceptual la estructura fundamental seleccionada para interconectar el hardware.

Clasificación por topología

Centralizados Los componentes tienen roles.
Cliente-Servidor: Prestan y solicitan servicios.
Jerárquica: Control de recursos y acceso por capas.

Descentralizados No hay roles, todos comparten obligaciones.
Peer-to-Peer: Los participantes se solicitan recursos entre ellos.

Clasificación por estructura

Capas Los elementos se especializan en capas. Sólo se comunican las capas contiguas. EJ: OSI, TCP/IP

Objetos Los elementos son objetos autónomos que invocan los métodos remotamente en una red.

Eventos El sistema tiene una cola donde se agregan las peticiones y se van desplazando en orden.

CLIENTE-SERVIDOR

Dos entidades diferenciadas. La aplicación se distribuye en componentes y estos en varios equipos.

Servicio Modelo de interacción. Basado en dialogo Petición ↔ Respuesta.

El cliente inicia la comunicación solicitando un servicio y el servidor responde concediendolo. El servidor es un cuello de botella y no escala bien solo.
No se usan recursos de los clientes.
El cliente debe conocer los servicios ofertados por el servidor.

Cliente-Servidor con Cache

Mejora la latencia al poder usar copias locales y reduce el uso de la red y de los recursos, lo que ayuda a la escalabilidad.

Cliente-Servidor con Proxy

Agrega un intermediario entre el cliente y el servidor (Tubería).

Pueden procesar, filtrar o cachear información.
Se pueden encadenar tantos proxies como sean necesarios.
Funcionan mejor si la interfaz del proxy es la misma que la del servicio. Tipos:

Forward El proxy está en la misma red que el cliente.

Reverse El proxy está en la misma red que el servidor.

Open El proxy está fuera de las redes del cleinte y del servidor.

Cliente-Servidor con Jerarquía/Multicapa

Los servidores son clientes de otros servidores.

2-Tier Tradicional. Un servidor multicliente.
Dificultades para escalar, alta dependencia con el servidor. Hay que modificar a los clientes cuando cambia la lógica de negocio.

3-Tier Extensión del modelo tradicional.
El cliente se dedica exclusivamente a la interfaz de usuario. La lógica de negocio se realiza en el servidor de datos y el intermedio.

N-Tier Extensión del 3-Tier.
La lógica de negocio se ubica en servicios entre la interfaz y los datos. Las capas sólo se comunican con sus capas contiguas. Es complejo de gestionar pero es el más flexible, modular y seguro. Hace un uso más intensivo de la red.

Cliente-Servidor con Replicación

Para reducir los cuellos de botella con el servidor.

Reparto de servicios Se clona el servidor. Atienden las peticiones individualmente, se reparten el trabajo. Ayuda a la escalabilidad. Reduce latencia.

Cooperar en el servicio Se reparten las tareas del servicio. No mejora la escalabilidad ni la latencia, pero sí la tolerancia a fallos.

Todas las réplicas del servicio deben actualizarse simultáneamente.

TIPOS DE CLIENTE

Depende de cómo se repartan el trabajo el cliente con el servicio.

Cliente ligero

No implementa ninguna lógica. Es una interfaz que recoge y presenta los datos. El servidor tiene más trabajo. EJ: El servidor genera HTML y el cliente lo representa.

Clientes pesados

Implementan gran parte de la lógica, procesos o almacenamiento de datos. El servidor es sencillo, poca responsabilidad. EJ: Cliente y servidor de BD que sólo guarda los datos.

Clientes híbridos

La implementación de la lógica se reparte entre cliente y servidor. EJ: Cliente y servidor con BD y procedimientos almacenados.

SISTEMAS ENTRE IGUALES

Todos los nodos tienen la misma funcionalidad.

Se aprovechan los recursos de todas las máquinas y no hay cuellos de botella. Los nodos pueden entrar y salir sin problemas. Se autogestiona al no estar centralizado.

La administración es compleja y presenta fallos de seguridad debido a la arquitectura.
Son útiles para almacenamiento distribuido, comunicación de grupos, computación distribuida. Siempre y cuando compartan interfaz.

SISTEMA EDITOR-DISTRIBUIDOR

No tiene nada que ver con Cliente-Servidor.

Los suscriptores se suscriben por intereses y los editores generan eventos. Estos eventos son enviados a los suscriptores. Es asíncrono y desacoplado. Se puede acceder desde

Push El suscriptor revive el evento.

Pull El suscriptor pregunta cuando quiere obtener eventos.

Híbrido El suscriptor revive las notificaciones pero tiene que solicitarlas previamente.

Puede haber intermediarios para que no sean los editores los que hablen directamente con los suscriptores.

COMUNICACIÓN ENTRE PROCESOS

Todo sistema distribuido se fundamenta en la comunicación entre procesos. Estos han venido mejorando y abstractamente cada vez más en el tiempo.

TEMA 3

INTRODUCCIÓN A STREAMS (SOCKETS)

La comunicación permite que los procesos colaboren, se realiza mediante el paso de mensajes. Todos los mecanismos cuentan con las siguientes primitivas.

Enviar datos a otro proceso.
Recibir datos del emisor.

Solo si el sistema está orientado a coneión
Iniciar conexión emite una petición de conexión. (Cliente)
Esperar conexión escucha a la espera de conexiones entrantes. (Servidor)
Aceptar conexión acepta las conexiones pendientes de realizar. (Servidor)

Esquemas de sincronización

Primitivas bloqueantes	Primitivas no bloqueantes
EL proceso que la invoca espera hasta que la primitiva termine completamente.	El proceso que la invoca continúa sin esperar a que la primitiva finalice completamente.

Enviar bloqueante El emisor espera hasta que el receptor reciba el mensaje.
Enviar no bloqueante El emisor coloca los datos en el buffer y continúa.
Recibir bloqueante El receptor espera hasta que haya datos que leer.
Recibir no bloqueante Si no hay datos el receptor continúa, si los hay los recoge.
Inicio de conexión bloqueante Espera hasta que la conexión sea aceptada.
Aceptar conexión no bloqueante Espera hasta que haya una petición que aceptar.

Tipos de comunicación

Comunicación síncrona Enviar y recibir son bloqueantes. La sincronización está garantizada. Es más fácil de usar pero no tiene un buen rendimiento.

Comunicación asíncrona Enviar es no bloqueante, mejora el rendimiento y evita bloqueos indefinidos. Recibir suele ser bloquearte, aunque puede no serlo.

Buena practica La habitual es poner las operaciones bloqueantes en un hilo a parte. EJ: en los servidores multihilo se crea un hilo por cada petición.

MODELO OSI Y TCP/IP

Capas del modelo OSI

Física conexión binaria. Transmisión y modulación.
Enlace Acceso al medio y comunicación entre nodos de una red.
Red Elige rutas y direccionamiento lógico jerarquizo.
Transporte Comunicación punto a punto libre de errores.
Sesión Multiplexa entre aplicaciones.
Presentación Adapta el formato de los datos.
Aplicación Suministra los servicios.

Comparación OSI y TCP/IP
La sesión, presentación y aplicación de OSI se resumen en la capa aplicación de TCP/IP. Las capas de enlace de datos y física de OSI de combinan en el acceso a red de TCP/IP.

SOCKETS

API de nivel de transporte, permite la conexión de procesos en un punto común, Es parte del kernel de un sistema operativo.

> Protocolo TCP Fiable, orientado a conexión (Stream).
> Protocolo UDP No fiable, no orientado a conexión (Datagrama).

Puertos
Usados tanto en TCP como UDP (Son independientes). Número de 16 bits que identiofica un servicio.

Socket
Un socket consta de dirección IP y un puerto. Los sockets no son compartidos. La implementación suele ser asíncrona. Enviar no bloqueante y recibir bloqueante. Aunque sea orientado a conexión o no.

SOCKETS EN JAVA

UDP
DatagramSocket es un socket que envia DatagramPacket. Necesita un DatagramPacket para enviar y recibir. Para enviar se crean con InetAddress, puerto, un array de bytes y la cantidad de bytes a enviar de ese buffer. Para recibir se necesita un array y la longitud máxima.

TCP
Socket es un stream de una conexión, de él se obtien los InputStream y OutputStream. El socket se puede obtener por conexión mediante InetAddress y puerto o por escucha con un ServerSocket que recibe el puerto de escucha y acepta conexiones creando sockets.

SOCKETS EN C#

UDP
Se usa la clase UdpClient para enviar y recibir. Enviar requiere un array de bytes, la longitud y un IPEndPoint que se genera con un IPAdress y un puerto. Para recibir se pasa una referencia a un IPEndPoint,. Any sifnifica que escucha en todas las direcciones de la máquina. Devuelve un array de bytes.

TCP
Se usa Socket para simbolizar la conexión. A partir de ahí se obtienen NetworkStream para escribir y un Stream normal para leer. Se lee o escribe con StreamReader o StreamWriter. El socket se obtiene de TcpListener en caso del servidor, requiere un IPEndPoint con la dirección y el puerto de escucha. TcpClient para el cliente, también requiere una dirección de conexión y el puerto.

TEMA 4

INTRODUCCIÓN AL RPC

La llamada a procedimiento remoto se implementa sobre streams, abstrae los detalles de la conexión. Abstrae:

- Aplanamiento (Serialización, marshaling) de los datos.
- Desaplanamiento (Deserialización, unmarshaling) de los datos.
- Formato de los datos, tipos complejos, endian.
- Gestión de diálogos por el socket.

Es un entorno que genera código de apoyo automaticamente.
Ofrece la ilusión (transparencia) de que se realizan llamadas a las funciones de forma remota, del mismo modo que si fueran locales. Se ocultan los detalles de la comunicación e implementación.

FUNCIONAMIENTO

Cliente

Cuando el proceso llama a una función:

- Se envía un mensaje con ID de función y los argumentos.
- Al recibir la respuesta desempaqueta el valor devuelto y lo pasa el proceso.

Servidor

Cuando recibe un ID de función y argumentos:

- Invoca la función.
- Envía un mensaje al cliente con la respuesta devuelta.

DIFERENCIAS CON UNA LLAMADA LOCAL

Pueden producirse errores en el servidor, en el cliente o en la red.
No hay acceso a variables globales ni al resto de la memoria o sistema. No se pueden usar punteros, eso prohíbe las estructuras de datos enlazados, entre otras. Tampoco se permite el paso por referencia, solo paso por valor simulado.
EL rendimiento es mucho menor que una llamada local por la red y por el aplanamiento de datos. Se puede imponer un límite a la complejidad de las estructuras aplanables.

COMPORTAMIENTO ANTE FALLOS

En las llamadas locales hay garantía de que el procedimiento se llama exactamente una vez y se obtiene el valor devuelto o se recibe la excepción que identifica el error.
En RPC se puede recibir un código de error en caso de fallo o puede no resibirce nada y quedar en unas espera indefinida. Causas frecuentes:

Fallo en el procedimiento remoto

Puede ser por un problema de hardware, red o una excepción no controlada el código

Fallo de comunicación

Perdida, desorden o corrupción de datos.

El cliente no tiene forma de saber exactamente cual es la causa del problema. Puede que la petición no llegara, que hubiera un fallo al procesarla o que la respuesta no haya llegado.
Se intenta imitar el funcionamiento de las llamas locales.

Semántica de las llamas de red

No es posible lograr la semántica “Exactamente una vez” de las llamadas locales pero se intenta imitar con aproximaciones.

Tal vez El procedimiento puede realizarse una o ninguna vez. Solo se envía una petición, si no llega una respuesta en un tiempo máxima continúa sin esperar. No se sabe qué es lo que pudo ocurrir ni se intenta repetir. Es útil solo si la perdida de peticiones no ocasiona problemas.

Al menos una vez El procedimiento se ejecuta una vez o más. El cliente envía de nuevo las peticiones si no se recibe respuesta. El servidor no elimina los duplicados, el procedimiento se puede ejecutar más de una vez y el cliente puede recibir más de una respuesta. Solo es viable cuando la idemtpotencia está garantizada en las operaciones.

Idempotencia: $f(x) = f(f(x))$

Como máximo una vez Se ejecuta una vez o ninguna. El cliente reenvía las pteiciones si no recibe contestación pero el servidor guarda las operaciones ejecutadas para asegurarse que solo se ejecutan una sola vez.

IMPLEMENTACIÓN DE RPC

Para ofrecer la idea de que los procedimientos se ejecutan desde el cliente, se crea un stub. Es un representante del servidor en el cliente. Y un Skeleton, que es un representante del cliente en el servidor.
Stub y Skeleton se generan automáticamente a partir de la definición de la interfaz.

Stub Tiene los procedimientos definidos en la interfaz.

Skeleton Realiza las llamadas al servidor de forma local como si fuera el cliente.

Binding Ofrece la transparencia de localización. Gestiona el nombre del servicio y su asociación con el skeleton.

Compilador de interfaces Recibe la definición de los procedimientos y estrucutas de datos de la interfaz, es quien genera el Stub y el Skeleton.

TEMA 5

INTRODUCCIÓN A RMI

RMI persigue el mismo objetivo que RPC pero integra la orientación a objetos.
Sistema de objetos distribuidos.

Permite llamadas a métodos de objetos remotos y paso de referencias a objetos remotos de forma transparente.

INTERFAZ REMOTA

Para que un objeto ofrezca método fuera de su JVM local, debe implementar una interfaz que herede de Remote. La interfaz recoge las operaciones expuestas por el servicio. Los parámetros de entrada siempre se pasan por copia (real).

Los métodos pueden lanzar escepciones, al menos deben poder lanzar RemoteException. Si solo tienen esta excepción la semántica es “Exactamente una”, si pueden lanzar más excepciones es “Como mucho una”.

Stub

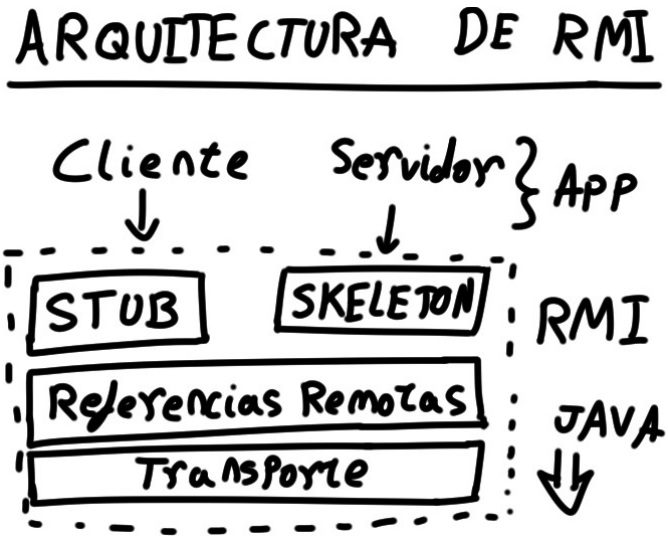
Implementa la interfaz remota, maneja las referencias a otros objetos remotos.

Servicio de nombres

El servidor de nombres registra y distribuye referencias remotas. Los clientes reviven una copia del stub.

PROPIEDADES DE RMI

Los clientes interactúan sólo con las interfaces del servicio, nunca con las clases que lo implementan. A estas referencias sólo se las puede castear.
Los objetos locales funcionan únicamente en la JVM que los creó, por eso los parámetros y valores de vueltos se pasan clonandolos binariamente y sus referencias también. Toda la serialización es transparente. La serialización es genérica mediante una interfaz obligatoria si el objeto tiene que ser transmitido por RMI.



TEMA 6

INTRODUCCIÓN A NET REMOTING

Dominio de aplicación

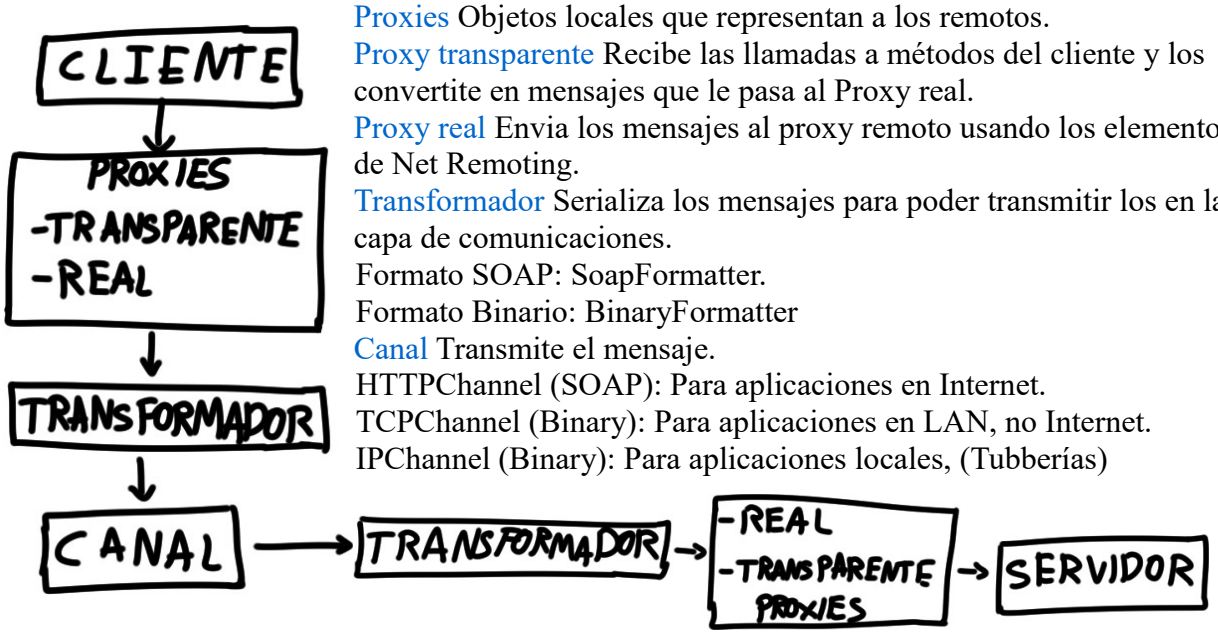
Un dominio de aplicación es el entorno en el que se ejecuta la aplicación. Los objetos de distintos dominios de aplicación no se pueden comunicar directamente. Requiere el paso de mensajes.

Net Remoting

Net Remoting es la solución dentro del dotNet Framework de Microsoft para que objetos de distintos dominios de aplicación puedan comunicarse aunque esos dominios de aplicación estén en maquinas distintas.

Dicen que es independiente del lenguaje (Aunque te fuerzan a usar C#)

ARQUITECTURA



Proxies Objetos locales que representan a los remotos.
Proxy transparente Recibe las llamadas a métodos del cliente y los convertite en mensajes que le pasa al Proxy real.
Proxy real Envia los mensajes al proxy remoto usando los elementos de Net Remoting.
Transformador Serializa los mensajes para poder transmitir los en la capa de comunicaciones.
Formato SOAP: SoapFormatter.
Formato Binario: BinaryFormatter
Canal Transmite el mensaje.
HTTPChannel (SOAP): Para aplicaciones en Internet.
TCPChannel (Binary): Para aplicaciones en LAN, no Internet.
IPChannel (Binary): Para aplicaciones locales, (Tubberías)

OBJETOS REMOTOS

Un objeto remoto es aquel que reside en un dominio de aplicación diferente al dominio desde el que es llamado.

Paso de un dominio de aplicación a otro

Hereda de MarshalByReference El cliente obtiene un proxy que hace referencia al objeto real. (MBR)

Implementar Serializable El cliente obtiene una copia binaria del objeto (MBV)

Activación del objeto remoto

Siempre se crea un objeto remoto en el servidor y el cliente obtiene un proxy, pero hay matices.

Activación en servidor

El servidor expone un objeto. Es el servidor quien controla los tiempos de vida. La sustanciación ocurre cuando el primer cliente llama a un método. SI el objeto a ha exipirado, se vuelve a crear otro nuevo en la siguiente llamada.

Singletron: Tan solo existe un objeto remoto a la vez. La primera llamada lo crea y el resto acceden al mismo. Todos los clientes comparten el mismo objeto remoto.

SingleCall: EL objeto es creao cuando lega una llama y destruido al terminal. Cada llamada se ejecuta sobre un objeto remoto distinto.

Publicación: El servicio instancia él mismo el objeto remoto y lo publica. Todos los clientes accedenb a ese objeto remoto ya existente. Es método equivalente a RMI de Java.

Activación en cliente

Es el cliente quien hace new del objeto remoto. Aunque cada cliente tiene un solo objeto remoto distinto. Este objeto no se comparte entre clientes. Es el cliente quien controla el tiempo de vida.

CONCESIONES DE TIEMPO

Los objetos remotos activados por el cliente y los activados por el servidor en modo Singleton tienen un tiempo de vida.

Si el tiempo de vida se acaba el objeto es marcado para su destrucción.
Inicialmente cuentan con 5 minutos.
Cada vez que el objeto recibe una invocación se renueva el tiempo 2 minutos.
Antes de destruir el objeto se busca al esponsor del objeto que decidirá durante cuanto tiempo se mantiene al objeto remoto con vida. Si el esponsor no contesta en dos minutos o no exixte el objeto es destruido inmediatamente.

Todos estos tiempos se pueden momificar junto con otros aspectos de la concesión del tiempo de las siguientes formas.

Actualizar el fichero de configuración

Permite modificar el tiempo de concesión inicial, renovación por invocación, espera a la respuesta del esponsor y tiempo de búsqueda de concesiones.

Sobrecargar el método InitialLifetimeService()

Puede devolver null si el objeto no tiene límite o un Ilease modificado que permite cmabiar los mismos aspectos que el fichero de configuración.

Usando GetLifetimeService()

Puede ser llamado por el cliente, se modifican los valores del objeto devuelto. Permite registrar el objeto esponsor. La clase esponsor debe implementar la interfaz ISponsor y heredar de MarshalByReferenceObject. Tiene que implementar el método Reneval que recive un ILease y devuelve un TimeSpan.

TIPOS DE LLAMADAS

Síncrono

El cliente espera hasta tener el resultado. Las llamadas son indepndientes y se ejecutan secuencialmente. Se puede crear un hilo por llamada pero es mejor asíncrono.

Asíncrono

Se usan delegados que son punteros a un método. El delegado se utiliza para realizar llamadas asíncronas. Se puede esperar al resultado cuando interese.

Asíncrono sin retorno (OneWay)

No comprueba si el método se ejecutó o no realmente, no puede devolver ningún valor. Se debe reservar para operaciones dónde no importe si se realiza o no. En la definición se marca con OneWay en el método.

TEMA 7

INTRODUCCIÓN A WCF

Un servicio web según la W3C es un sistema software diseñado para permitir la interoperatividad en al red.

WCF es la continuación de Net Remoting. La comunicación se realiza en XML con estándares SOAP transmitiendo por HTTP. Los servicios se definen en WSDL. El motivo principal HTTP es el puerto 80.

Tienen un bajo acoplamiento al no depender de DLLs compartidas.

Organización en capas:

Capa de contratos

Define los puntos en común entre cliente y servidor para permitir la comunicación.

- Contrato de servicios Metadatos.
- Contrato de datos Estructuras de datos, meros contenedores.
- Contrato de mensajes define los campos de cabecera y cuerpo.
- Política y registro Características del canal de comunicación.

Capa de ejecución del servicio

Describe el funcionamiento del servicio en operación.

- Throttling Behavior Cantidad de mensajes que puede procesar.
- Error Behavior Comportamiento en caso de error y cómo informar de ello.
- Metadata Behavior Define la definición del servicio.
- Instance Behavior Forma de instanciar la clase que implementa la interfaz.
- Transaction Behavior Operaciones de transacciones y caso de rollback.
- Dispatch Procesado de mensajes.
- Message Inspector Las partes de un mensaje.
- Parameter Filtering Acciones predeterminadas en función del contenido del mensaje.

Capa de mensajería

Canales de comunicación y la estructura del mensaje.

- WS-Security Channel Implementa seguridad a nivel del mensaje.
- WS-Reliable Messaging Channel Asegura que los mensajes llegan.
- Encoders Formato de representación de los datos.
- HTTPChannel Transporte de mensajes por HTTP.
- TCPChannel Transporte de mensajes por TCP.
- Transaction Flow Channel Mensajes para las transacciones.
- NamedPipe Channel Comunicación entre procesos mediante tuberías.
- MSMQ Comunicaciones mediante colas de mensajes.

Hosting y activación

Formas de ejecutar y activar un servicio WCF.

- Self-Hosted Un ejecutable exe.
- Administrado A través de un agente extorno (IIS).
- Servicio de Windows

Los componentes COM+ pueden ser alojados como WCF.

UTILIZACIÓN DE LOS SERVICIOS WCF

Dada la estandarización de WCF, hay un cliente capaz de explotar todos los servicios.
Visual Studio permite agregar una referencia WCF y acomodar el proyecto para que lo utilice.

DISEÑO

Es un DLL normalmente con la definición del servicio (métodos) con sus contratos de operaciones y datos y miembros. El servidor hace accesible los servicios.
App.config define como se exponen los metadatos del servicio. En producción es recomendable quitarlo.
El cliente se comunica con el servidor mediante un proxy. El proxy se forma con sólo los métadatos del servicio cuando están expuestos.

El host puede ser un ser vidor web o el publicador de Windows.
El servicio puede definir el nivel de concurrencia. Simple, Múltiple o reentrante (bloqueo de concurrencia por operaciones).