## Unidad 4: Objetos Distribuidos e Invocación Remota

Los sistemas distribuidos permiten que diferentes programas trabajen juntos como si fueran una sola aplicación, aunque se ejecuten en diferentes computadoras. Esta colaboración se logra mediante **invocación remota**, donde un programa puede pedirle a otro que ejecute ciertas operaciones. Para hacer esto, se usan dos técnicas principales: **RPC** y **RMI**.

## 1. Middleware en Aplicaciones Distribuidas

El **middleware** es una capa de software que facilita la comunicación entre los diferentes procesos de un sistema distribuido, manejando aspectos como:

- **Transparencia de ubicación**: el cliente no necesita saber si la operación ocurre en su propia máquina o en otra.
- **Protocolos de comunicación**: son independientes de los protocolos de red subyacentes.
- **Soporte de múltiples lenguajes**: el middleware permite que programas en distintos lenguajes trabajen juntos.

Esta capa proporciona una interfaz uniforme para la comunicación, de modo que los programas distribuidos puedan funcionar como si estuvieran en el mismo equipo.

## 2. Protocolos de Petición-Respuesta

Los sistemas distribuidos operan con **modelos de petición-respuesta**. Básicamente, el cliente pide una operación, y el servidor responde con los resultados. Esto puede ocurrir de dos maneras:

- **Sincrónica**: el cliente espera la respuesta del servidor antes de continuar.
- **Asíncrona**: el cliente envía una petición y sigue trabajando, revisando la respuesta más tarde.

### Primitivas de Comunicación en Petición-Respuesta

Se usan tres comandos principales para manejar esta comunicación:

- **doOperation(s, args)**: inicia la operación en un servidor.
- **getRequest()**: el servidor recibe la solicitud del cliente.
- **sendReply(r, c)**: el servidor envía la respuesta al cliente.

## 3. Llamada a Procedimiento Remoto (RPC)

La **RPC** permite que una función de un programa local llame a otra función en un programa remoto como si fuera una función normal, proporcionando **transparencia de distribución**. Para que esto funcione:

- El cliente usa un **procedimiento de resguardo** (o proxy), que representa la función remota y empaqueta los datos en un mensaje para el servidor.

- El servidor tiene un **esqueleto** que recibe el mensaje, desempaqueta los datos y llama a la función correspondiente.

Estos elementos en el cliente y el servidor garantizan que la comunicación entre procesos remotos sea casi idéntica a una función local.

## 4. Invocación de Método Remoto (RMI)

La **RMI** es una extensión de RPC que funciona con **objetos distribuidos**. Aquí, los métodos que un objeto remoto puede ejecutar se definen en una **interfaz remota**.

- Un **proxy** en el cliente actúa como un sustituto del objeto remoto, manejando la comunicación.
- El **esqueleto** en el servidor toma los mensajes y llama al método correspondiente en el objeto remoto.

Además, **referencias a objetos remotos** se utilizan para que cualquier programa distribuido pueda identificar y llamar a un objeto específico en otra máquina.

## 5. Manejo de Fallos y Semánticas de Invocación

En los sistemas distribuidos, existen varias estrategias para manejar los fallos durante la comunicación:

- **Reintento de mensajes**: el sistema sigue enviando la solicitud hasta recibir una respuesta o confirmar que el servidor no responde.
- **Filtrado de duplicados**: se eliminan peticiones repetidas.
- **Retransmisión de resultados**: se almacena el resultado de una operación para reenviarlo si se pierde la respuesta inicial.

Estas técnicas ayudan a garantizar que los mensajes se envíen correctamente y que los clientes sepan si una operación se realizó, si puede haber sido duplicada o si falló.

**Semánticas de Invocación**

- **"Pudiera ser"**: la operación puede no haberse ejecutado en absoluto, o puede haber fallado sin que el cliente lo sepa.
- **"Al menos una vez"**: la operación se ejecutará al menos una vez, aunque esto puede causar duplicación.
- **"Como máximo una vez"**: la operación se ejecutará una vez o ninguna, garantizando que no haya duplicados.

## 6. HTTP como Ejemplo de Protocolo de Petición-Respuesta

El protocolo **HTTP** es una implementación común del modelo de petición-respuesta, usado en la web. Cada operación en HTTP se define por métodos específicos:

- **GET**: obtener datos.
- **POST**: enviar datos.
- **PUT**: actualizar datos.
- **DELETE**: eliminar datos.

HTTP permite que los navegadores web y los servidores intercambien información de manera segura y confiable.

## Unidad 5: Tiempo y Estados Globales.

### 1. Introducción al Tiempo en Sistemas Distribuidos

En sistemas distribuidos, cada proceso tiene su propio **reloj** y ejecuta eventos que pueden ser:

- **Eventos de comunicación** (enviar/recibir mensajes).
- **Eventos de transformación del estado** (cambios en las variables del proceso).

Cada proceso tiene una secuencia ordenada de eventos, pero al no compartir un reloj global, surgen problemas para ordenar los eventos en diferentes procesos.

---

### 2. Sincronización de Relojes

Existen tres métodos principales para sincronizar relojes en sistemas distribuidos:

- **Método de Cristian**: Un proceso pide la hora a un servidor de tiempo y ajusta su reloj sumando la mitad del tiempo de ida y vuelta del mensaje. Este método es ideal en redes con baja latencia.
- **Algoritmo de Berkeley**: Un nodo se designa como **master** y coordina el ajuste de los relojes de los demás nodos (slaves).
- **Protocolo de Tiempo de Red (NTP)**: Usado en Internet, este protocolo organiza los servidores en **estratos** (niveles jerárquicos), donde los servidores primarios se sincronizan con relojes de alta precisión, y sincronizan a los servidores de nivel inferior.

---

### 3. Relojes Lógicos

Para ordenar eventos sin depender del tiempo físico, se usan **relojes lógicos**, que funcionan con contadores en lugar de tiempo real.

- **Relojes de Lamport**: Capturan la relación “sucedió antes” entre eventos. Cada proceso tiene un reloj lógico que se incrementa con cada evento. Cuando un mensaje es enviado,

el reloj lógico del emisor se incluye en el mensaje, y el receptor actualiza su reloj para reflejar la mayor marca de tiempo entre ambos.
- **Relojes Vectoriales**: Superan la limitación de los relojes de Lamport al usar un vector de enteros, donde cada proceso tiene su propio contador en el vector. Esto permite conocer si un evento realmente sucedió antes que otro, incluso en procesos distintos.

---

## 4. Estados Globales y Cortes Consistentes

El **estado global** de un sistema distribuido incluye el estado de cada proceso y los mensajes en tránsito. Para analizarlo, se utiliza el concepto de **cortes consistentes**, que son subconjuntos de eventos tomados de cada proceso en un punto en el tiempo.

- **Cortes consistentes**: Un corte incluye eventos que respetan la causalidad, es decir, si un evento de envío de mensaje está en el corte, también lo estará el evento de recepción correspondiente.
- **Algoritmo de Chandy y Lamport**: Este algoritmo captura una instantánea del estado global enviando un **marcador** entre los procesos para registrar su estado y el de los canales de comunicación.

---

## 5. Depuración Distribuida y Predicados Globales

La **depuración distribuida** evalúa condiciones en el sistema mediante **predicados globales**, los cuales se aplican a los estados globales para identificar errores o asegurar condiciones de seguridad.

- **Predicados de estabilidad**: Si un predicado es verdadero en un momento dado, se mantendrá verdadero.
- **Evaluación de predicados**: Existen métodos para evaluar si un predicado es verdadero "posiblemente" (puede ser verdadero en algún estado) o "sin duda alguna" (es verdadero en todos los estados posibles).

**Unidad 6: Web Services**

Esta unidad explora los **Servicios Web (Web Services)**, fundamentales en la comunicación y la interoperabilidad entre aplicaciones en la web. Los temas abarcan conceptos clave, protocolos, funcionamiento, y casos de uso en diferentes industrias. A continuación se detallan cada uno de los temas del documento:

---

## 1. Introducción a los Servicios Web

Los **Servicios Web** son interfaces de programación diseñadas para la comunicación entre aplicaciones en internet. Permiten la interoperabilidad entre plataformas y frameworks diferentes, usando **protocolos abiertos** y estándares. Esto es especialmente importante para integrar sistemas de diferentes empresas o realizar aplicaciones innovadoras basadas en servicios ya existentes. Son también una base para tecnologías como la computación en la nube y el grid computing.

- Los servicios web usan XML para el formato de mensajes, y comúnmente se transmiten mediante HTTP/S.
- Las interfaces de los servicios se describen en **lenguajes de descripción** (como WSDL) y usan protocolos de comunicación como SOAP o REST.

## 2. Protocolos y Tecnologías Clave en los Servicios Web

Los servicios web se basan en varios protocolos para definir, descubrir, y asegurar la comunicación entre aplicaciones:

- **URI, URL y URN**: El **Uniform Resource Identifier (URI)** identifica recursos en la web. Los URIs pueden ser URLs, que incluyen la ubicación del recurso, o **URNs**, que son nombres globales que no dependen de la ubicación (como el ISBN de un libro).
- **XML (Extensible Markup Language)**: Un lenguaje flexible y extensible usado para intercambiar datos estructurados. XML permite compatibilidad entre diferentes plataformas y es la base de muchos otros protocolos de servicios web.
- **SOAP (Simple Object Access Protocol)**: Define cómo los datos se intercambian en XML y puede utilizar varios protocolos de red como HTTP y SMTP. SOAP ofrece:
    - **Reglas de codificación** para definir tipos de datos.
    - Convenciones para **llamadas a procedimientos remotos (RPC)** y sus respuestas.
    - Reglas específicas para uso con HTTP.
- **WSDL (Web Services Description Language)**: Describe los servicios web, definiendo las operaciones, los tipos de mensajes y los protocolos que se deben usar para acceder al servicio.
- **UDDI (Universal Description, Discovery, and Integration)**: Un registro que permite a las empresas descubrir y registrar servicios web, facilitando la integración entre sistemas.

## 3. Funcionamiento de los Servicios Web

Los servicios web funcionan mediante el intercambio de mensajes entre el cliente y el servidor. El cliente realiza solicitudes de operación, y el servidor responde de acuerdo con las definiciones del servicio. Las tecnologías de SOAP y REST estructuran esta interacción de manera diferente:

- **SOAP** estructura los mensajes en XML y es ideal para entornos que requieren seguridad y manejo complejo de datos.
- **REST**, en cambio, se basa en la arquitectura **HTTP** y usa URL para interactuar con recursos representados en XML o JSON. REST es sencillo y escalable, por lo que se utiliza ampliamente en aplicaciones ligeras, como las interfaces móviles.

## 4. REST y sus Operaciones Principales

REST (Representational State Transfer) utiliza métodos HTTP para manipular recursos:

- **GET**: Recupera un recurso.
- **POST**: Crea un nuevo recurso.
- **PUT**: Actualiza un recurso existente.
- **DELETE**: Elimina un recurso.

REST se enfoca en la manipulación de recursos y es más adecuado para la mayoría de las aplicaciones modernas debido a su simplicidad y menor consumo de ancho de banda en comparación con SOAP.

## 5. WebSocket

**WebSocket** permite **comunicación bidireccional** en tiempo real entre un cliente y un servidor. A diferencia de las conexiones HTTP tradicionales, WebSocket mantiene una conexión persistente sobre TCP, lo cual es útil para aplicaciones de chat, videojuegos y otros sistemas que requieren respuesta inmediata. WebSocket funciona bajo el modelo de seguridad de origen, que suele ser implementado por navegadores web para mantener la seguridad en la comunicación.

## 6. Casos de Uso de los Servicios Web

Los servicios web han permitido a diversas industrias automatizar e innovar en sus sistemas. Algunos ejemplos destacados son:

- **Google Web API**: Permite a desarrolladores acceder a datos de la web de Google mediante SOAP y WSDL.
- **Amazon Web Services (AWS)**: Ofrece servicios mediante SOAP y REST, permitiendo a aplicaciones de terceros crear servicios de valor agregado sobre los servicios de Amazon, como inventario y pedidos.
- **Dollar Rent-a-Car y Southwest Airlines**: Integraron sus sistemas para que los clientes puedan reservar autos al mismo tiempo que compran pasajes aéreos en la web de Southwest.
- **Microsoft MapPoint.Net**: Proporciona información de ubicación y tráfico en tiempo real, ayudando a los usuarios a navegar y acceder a información geográfica precisa.
- **Líneas Aéreas Escandinavas**: Han desarrollado servicios que permiten a los clientes comprar boletos y revisar el estado de sus vuelos desde el móvil.

# Unidad 7: Transacciones y Control de Concurrencia en Sistemas Distribuidos

---

### 1. Concepto de Transacción

- **Definición**: Secuencia de operaciones atómicas realizadas en un sistema.
- **Propiedades ACID**:
    1. **Atomicidad**: Todo o nada.
    2. **Consistencia**: Las transacciones llevan el sistema de un estado válido a otro.

3. **Aislamiento**: Las operaciones intermedias no son visibles para otras transacciones.
4. **Durabilidad**: Los cambios realizados son permanentes.

---

## 2. Problemas en Concurrencia

- **Actualizaciones perdidas**: Dos transacciones leen un mismo dato y lo modifican sin coordinación, sobrescribiendo los cambios.
- **Recuperaciones inconsistentes**: Una transacción lee un estado intermedio modificado por otra, generando cálculos incorrectos.
- **Lecturas sucias**: Una transacción lee datos no confirmados de otra que podría ser abortada.
- **Abortos en cascada**: Si una transacción aborta, todas las que dependen de sus datos también deben abortar.

---

## 3. Técnicas de Control de Concurrencia

a) Bloqueos

- **Tipos**:
    1. **Bloqueo de Lectura (Compartido)**: Permite que varias transacciones lean simultáneamente.
    2. **Bloqueo de Escritura (Exclusivo)**: Restringe el acceso total a un recurso.
- **Bloqueo en Dos Fases (2PL)**:
    - **Fase de Crecimiento**: Se adquieren bloqueos.
    - **Fase de Disminución**: Se liberan bloqueos.
- **Deadlock (Bloqueo indefinido)**:
    - Dos o más transacciones esperan indefinidamente por recursos bloqueados por las otras.
    - **Solución**: Detección mediante gráfos de espera, timeouts o abortos.

b) Control Optimista

- Se asume que los conflictos son poco frecuentes.
- **Fases**:
    1. **Fase de Trabajo**: Operaciones sobre versiones tentativas.
    2. **Fase de Validación**: Verificación de conflictos.
    3. **Fase de Actualización**: Cambios tentativos se hacen permanentes.

---

## 4. Recuperabilidad y Manejo de Fallos

- **Imágenes Previas**: Guardan estados anteriores para restaurar en caso de abortos.
- **Abortos en Cascada**: Evitar haciendo que solo se lean datos confirmados.

- **Ejecuciones Estrictas**: Retrasan las operaciones hasta que las transacciones previas hayan terminado.

---

### 5. Transacciones Anidadas

- Permiten dividir una transacción en **subtransacciones**.
- Ventajas:
    - Mayor **concurrencia** y **robustez**.
    - Subtransacciones pueden abortarse o consumarse de forma independiente.

---

### 6. Esquemas de Bloqueo Avanzados

- **Bloqueo de dos versiones**:
    - Transacciones pueden escribir versiones tentativas mientras otras leen la versión confirmada.
    - Utiliza bloqueos de lectura, escritura y consumación.

---

### 7. Ejemplos Clásicos

a) Transferencia Bancaria

- Secuencia de operaciones: `extrae()` de una cuenta, `deposita()` en otra.
- Problemas si no se usan bloqueos adecuados:
    - **Actualización perdida**: Ambas transacciones usan valores iniciales.
    - **Recuperación inconsistente**: `totalSucursal()` interrumpe la transferencia.

b) Productor-Consumidor

- Productores llenan un buffer; consumidores lo vacían.
- Coordinación mediante `wait()` y `notify()` en métodos sincronizados.

## Unidad 8: Transacciones Distribuidas

---

### 1. Definición de Transacciones Distribuidas

- **Concepto**: Son transacciones que involucran a múltiples servidores.
    - **Atomicidad global**: O todas las operaciones de la transacción se completan correctamente en todos los servidores o todas se abortan.

- **Coordinador**: Un servidor asume el rol de coordinador para gestionar la transacción distribuida.

---

## 2. Componentes Clave

- **Coordinador**:
  - Responsable de iniciar y finalizar la transacción.
  - Asigna un identificador único (TID) a la transacción.
- **Participantes**:
  - Servidores que manejan objetos involucrados en la transacción.
  - Cooperan con el coordinador para implementar protocolos de finalización.

---

## 3. Protocolo de Consumo en Dos Fases

1. **Fase de Preparación**:
   - El coordinador solicita a cada participante que prepare sus cambios.
   - Los participantes responden con un "listo" o "no listo".
2. **Fase de Consumo**:
   - Si todos los participantes están listos, el coordinador envía un mensaje de "consumar".
   - Si algún participante no está listo, se envía un mensaje de "abortar".

---

## 4. Interbloqueos Distribuidos

- **Interbloqueos**: Situación en la que varias transacciones quedan esperando indefinidamente por recursos bloqueados entre sí.
- **Grafo Espera-Por**:
  - Representa relaciones de espera entre transacciones.
  - Los ciclos en el grafo indican un interbloqueo.
- **Técnicas de Detección**:

1. **Detección Global**:
   - Construcción de un grafo espera-por global.
   - Detecta ciclos y resuelve interbloqueos abortando transacciones.
2. **Captura de Arcos**:
   - Envío de sondas entre servidores para detectar ciclos en el grafo distribuido.

- **Interbloqueos Fantasmas**:
  - Ocurren cuando la información está desactualizada y se detecta erróneamente un ciclo.

---

## 5. Recuperación de Transacciones

- **Objetivo**: Restaurar el estado del servidor tras un fallo, manteniendo los últimos datos confirmados.
- **Técnicas**:
    - **Registro Histórico**:
        - Archivo que contiene el historial de transacciones.
        - Incluye valores de objetos, estados de transacciones y listas de intenciones.

---

### 6. Ejemplo de Protocolo Distribuido

- **Transacción Bancaria**:
    - Operaciones de transferencia entre cuentas gestionadas por distintos servidores.
    - El coordinador garantiza que todas las operaciones se consuman o aborten de manera consistente.

---

## Diagramas Clave:

- **Figura 8.1**: Transacciones distribuidas (planas y anidadas).
- **Figura 8.5**: Protocolo de consumo en dos fases.
- **Figura 8.13**: Detección de interbloqueo distribuido con grafos espera-por.