



UNIVERSIDAD **CATÓLICA**
de Colombia

Transacciones y Control de Concurrencia en los Sistemas Distribuidos

Diego Alberto Rincón Yáñez MSc

darincon@ucatolica.edu.co

Sistemas Distribuidos

Departamento de Ingeniería de Sistemas y Computación

Afiliada a la Federación Internacional de Universidades Católicas (FIUC)

w w w . u c a t o l i c a . e d u . c o

Introducción



UNIVERSIDAD CATÓLICA
de Colombia

- Primer acercamiento: Algoritmos “**Semi-Centralizados**”
- Objetivo de las **transacciones**:
 - Asegurar que todos los objetos gestionados por **un servidor** permanecen en un estado (c) cuando dichos objetos son **accedidos por múltiples transacciones** y en **presencia de fallos**



Sincronización Sencilla

- Las operaciones realizadas en nombre de **diferentes clientes** pueden **interferir** a veces unas con otras.
- La **interferencia** puede producir valores **incorrectos en los objetos**.
- Frecuentemente se usan **hilos** para permitir concurrentemente las operaciones de varios clientes y aun accediendo, posiblemente, a los **mismos objetos**.



Sincronización Sencilla

- Si los métodos no están **diseñados** para su utilización en un **programa multi-hilo**, es posible que las acciones de dos o más ejecuciones concurrentes del método puedan combinarse arbitrariamente y tener efectos extraños.
- Ejemplo en java:

```
public synchronized void deposita(int cantidad) throws RemoteException{  
  
}
```

Sincronización Sencilla



UNIVERSIDAD CATÓLICA
de Colombia

- En varias ocasiones es **necesario** hacer que los **hilos se comuniquen**.
- Ejemplo en java:
 - Metodos: **wait** y **notify**.
 - Wait: Un hilo llama a wait en un objeto para suspenderse él mismo y permitir a otro hilo ejecutar un método en ese objeto.
 - Notify: Un hilo llama a notify en un objeto para informar a cualquier hilo que esta esperando en el objeto que ha cambiado alguno de sus datos.



Modelos de fallos para transacciones

- En este modelo (**Lampson**) se intenta que los algoritmos trabajen correctamente en presencia de fallos predecibles, pero no se hacen consideraciones sobre su comportamiento cuando ocurre un desastre.
- El modelo establece:
 - Las **escrituras/lecturas** pueden fallar.



Modelos de fallos para transacciones

- Los **servidores** pueden fallar **ocasionalmente**.
- Puede existir un **retardo arbitrario** antes que llegue un mensaje.

Transacciones



UNIVERSIDAD CATÓLICA
de Colombia

- ¿Qué es?

- ACID

- Atomicidad

Todo o nada: Una transacción finaliza correctamente, y los efectos de todas sus operaciones son registrados en los objetos, o si fallan no tienen ningún efecto.

Transacciones



UNIVERSIDAD CATÓLICA
de Colombia

- Isolation
 - Cada transacción debe realizarse sin interferencia de otras transacciones.
- Consistency
 - La transacción en el caso que se haga debe hacer pasar el sistema de un estado consistente a otro.
- Durability
 - Los cambios registrados en los recursos debido a transacciones que hagan COMMIT deben soportar fallos siguiendo el modelo de lampson.

Transacciones



UNIVERSIDAD CATÓLICA
de Colombia

<i>Con éxito</i>	<i>Abortado por el cliente</i>	<i>Abortado por el servidor</i>
<i>AbreTransacción</i>	<i>AbreTransacción</i>	<i>AbreTransacción</i>
<i>Operación</i>	<i>Operación</i>	<i>Operación</i>
<i>Operación</i>	<i>Operación</i>	<i>Operación</i>
•	•	El servidor aborta la transacción → •
•	•	
<i>Operación</i>	<i>Operación</i>	<i>ERROR en la operación informado al cliente</i>
<i>CierraTransacción</i>	<i>AbortaTransacción</i>	

Transacciones



UNIVERSIDAD CATÓLICA
de Colombia

- Acciones de servicio relacionadas con la ruptura del proceso
 - Si un proceso **servidor** falla, se reemplaza en algún momento.
 - El nuevo proceso aborta todas las transacciones no finalizadas
 - Usa un proceso de **recuperación** para restablecer los valores de los objetos producidos por la transacción finalizada de forma correcta.

Transacciones



UNIVERSIDAD CATÓLICA
de Colombia

- Acciones de un **cliente** relativas a la ruptura del proceso servidor
 - Si un servidor falla mientras una transacción esta en progreso, el cliente será consciente de ello cuando una de las operaciones devuelve una excepción.



Control de concurrencia

- Para describir los problemas de transacciones concurrentes se usa el contexto de un ejemplo bancario.
 - Tres UP (objetos), el estado solo consiste en una variable entera. $A=100$, $B=200$, $C=300$
- Los problemas teniendo en cuenta el **solapamiento de las operaciones** son:
 - Actualizaciones perdidas
 - Recuperaciones inconsistentes

Control de concurrencia

(Actualizaciones perdidas)



UNIVERSIDAD CATÓLICA
de Colombia

Transacción <i>T</i> :	Transacción <i>U</i> :
$balance = b.obtenBalance();$ $b.deposita(balance/10);$ $a.extrae(balance/10)$	$balance = b.obtenBalance();$ $b.deposita(balance/10);$ $c.extrae(balance/10)$
$balance = b.obtenBalance();$ 200 \$	$balance = b.obtenBalance();$ 200 \$
$b.deposita(balance/10);$ 220 \$	$b.deposita(balance/10);$ 220 \$
$a.extrae(balance/10)$ 80 \$	$c.extrae(balance/10)$ 280 \$



Control de concurrencia (Recuperaciones inconsistentes)

Transacción V :		Transacción W :	
<i>a.extrae(100);</i> <i>b.deposita(100)</i>		<i>unasucursal.totalSucursal()</i>	
<i>a.extrae(100);</i>	0 \$	<i>total = a.obtenBalance()</i>	0 \$
		<i>total = total + b.obtenBalance();</i>	200 \$
		<i>total = total + c.obtenBalance()</i>	500 \$
<i>b.deposita(100)</i>	300 \$	• •	



Control de concurrencia

- Para describir los problemas de transacciones concurrentes se usa el contexto de un ejemplo bancario.
 - Tres UP (objetos), el estado solo consiste en una variable entera. $A=100$, $B=200$, $C=300$
- Los problemas teniendo en cuenta el **estado final de las transacciones** son:
 - Lecturas sucias
 - Escrituras prematuras

Control de concurrencia

(Recuperabilidad de transacciones abortadas)

Lecturas sucias



UNIVERSIDAD CATÓLICA
de Colombia

Transacción <i>T</i> :		Transacción <i>U</i> :	
<i>a.obtenBalance()</i>		<i>a.obtenBalance()</i>	
<i>a.ponBalance(balance + 10)</i>		<i>a.ponBalance(balance + 20)</i>	
<i>balance = a.obtenBalance()</i>	100 \$	<i>balance = a.obtenBalance()</i>	110 \$
<i>a.ponBalance(balance + 10)</i>	110 \$	<i>a.ponBalance(balance + 20)</i>	130 \$
		<i>consumar transacción</i>	
<i>abortar transacción</i>			

Control de concurrencia

(Recuperabilidad de transacciones abortadas)

Escrituras prematuras



UNIVERSIDAD CATÓLICA
de Colombia

Transacción T :			Transacción U :		
<i>a.deposita(5)</i>			<i>a.deposita(10)</i>		
	100 \$				
<i>a.deposita(5)</i>	105 \$				105 \$
			<i>a.deposita(10)</i>		115 \$
CT	AT	AT	AT	CT	AT



Control de concurrencia (Equivalencia secuencial)

Transacción *T* :

balance = b.obtenBalance();
b.deposita(balance/10);
a.extrae(balance/10);

balance = b.obtenBalance(); 200 \$

b.deposita(balance/10); 220 \$

a.extrae(balance/10); 80 \$

Transacción *U* :

balance = b.obtenBalance();
b.deposita(balance/10);
c.extrae(balance/10);

balance = b.obtenBalance(); 220 \$

b.deposita(balance/10); 242 \$

c.extrae(balance/10) 278 \$



Control de concurrencia (Equivalencia secuencial)

Transacción V :		Transacción W :	
<i>a.extrae(100);</i> <i>b.deposita(100)</i>		<i>unasucursal.totalSucursal();</i>	
<i>a.extrae(100);</i>	0 \$		
<i>b.deposita(100)</i>	300 \$		
		<i>total = a.obténBalance()</i>	0 \$
		<i>total = total + b.obténBalance();</i>	300 \$
		<i>total = total + c.obténBalance();</i>	600 \$
		...	



Control de concurrencia (Operaciones conflictivas)

Operaciones de diferentes transacciones		Conflicto	Causa
<i>Lee</i>	<i>Lee</i>	No	Porque el efecto de un par de operaciones de <i>lectura</i> no depende del orden en el que son ejecutadas.
<i>Lee</i>	<i>Escribe</i>	Sí	Porque el efecto de una operación de <i>lectura</i> y una de <i>escritura</i> dependen del orden de su operación.
<i>Escribe</i>	<i>Escribe</i>	Sí	Porque el efecto de un par de operaciones de <i>escritura</i> depende del orden de su ejecución.