

Transacciones Distribuidas

Basado en el cap. 13 del texto de G. Coulouris, J. Dollimore y T. Kindberg. Sistemas Distribuidos. Conceptos y Diseño.



Contenido

- Transacciones Distribuidas
 - Transacciones Planas y Anidadas
 - Commit y Abort en un Sistema Distribuido
 - Control de Concurrencia
 - Por Bloqueo
 - Optimista
 - Marcas Temporales
 - Recuperación



Transacciones Distribuidas

- Sus actividades **involucran múltiples servidores.**
- Se usa el término *transacciones distribuidas* para referirse a transacciones planas o anidadas que acceden a objetos administrados por múltiples servidores.

- Las **transacciones distribuidas** pueden ser planas o anidadas.

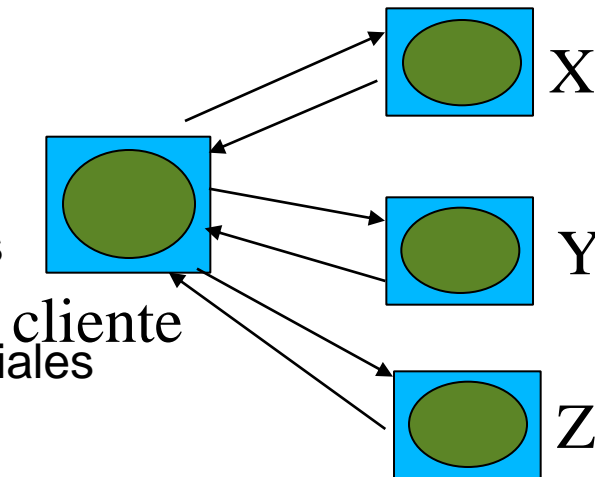
Cliente:

```
begin_transaction  
    call X.x  
    call Y.y  
    call Z.z  
end_transaction
```

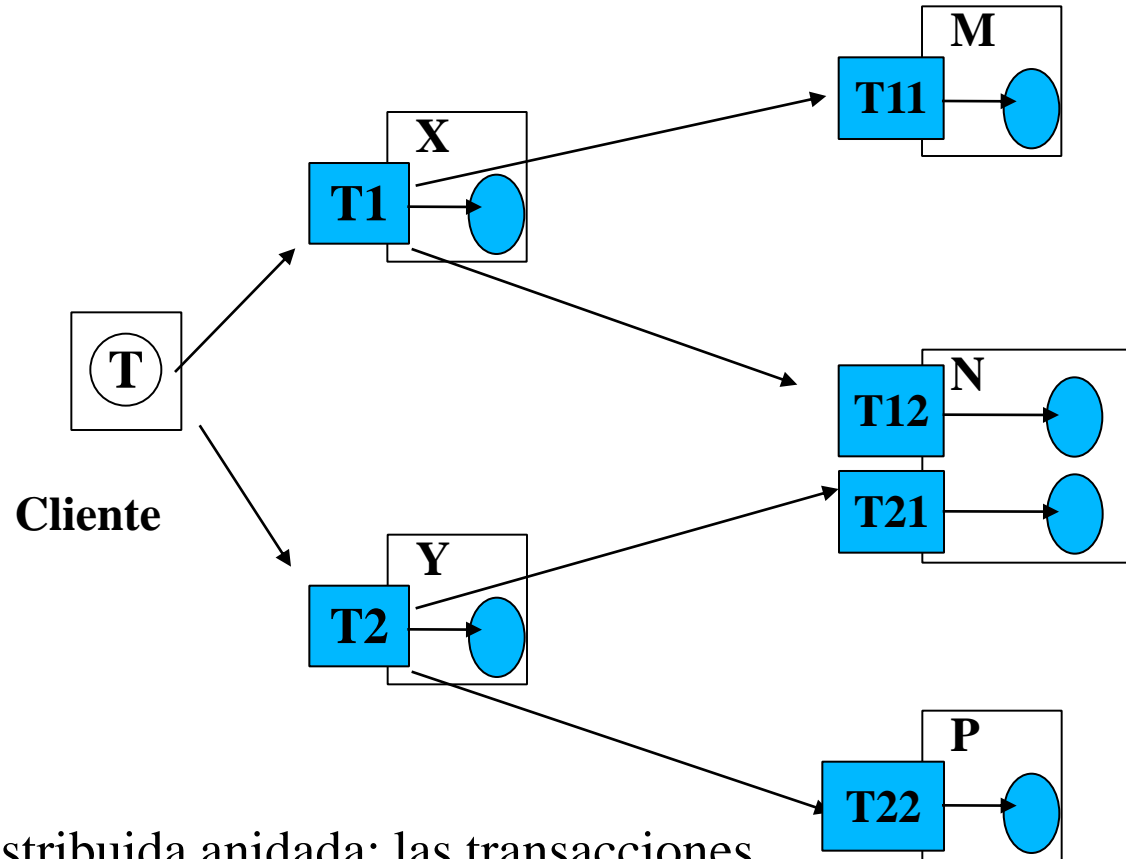
Transacción Plana:

Un cliente realiza peticiones
a más de un servidor.

Las peticiones son secuenciales



Transacciones Distribuidas

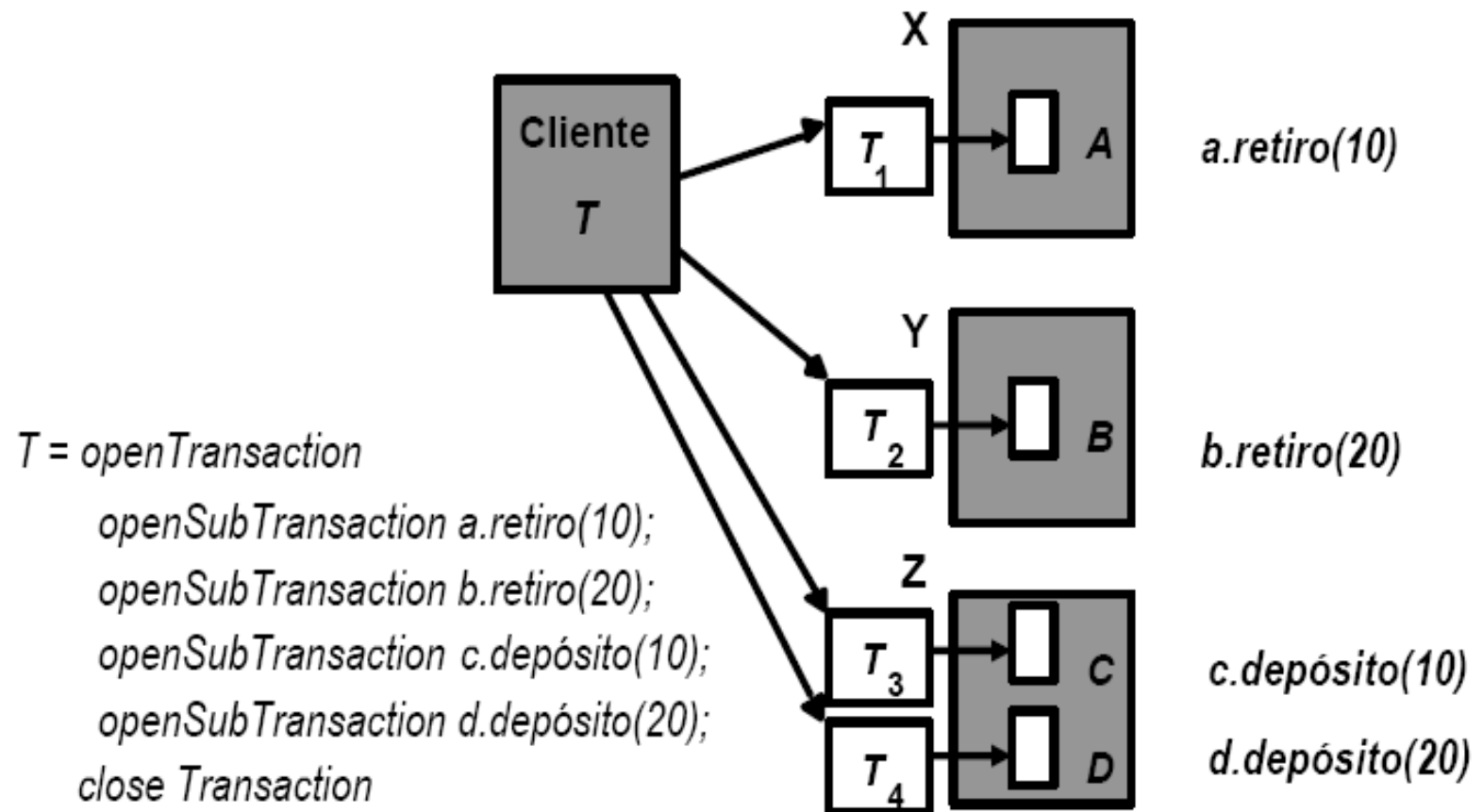


Transacción distribuida anidada: las transacciones del mismo nivel son concurrentes. **Si están en Servidores distintos pueden ejecutarse en paralelo.**

Transacciones Anidadas

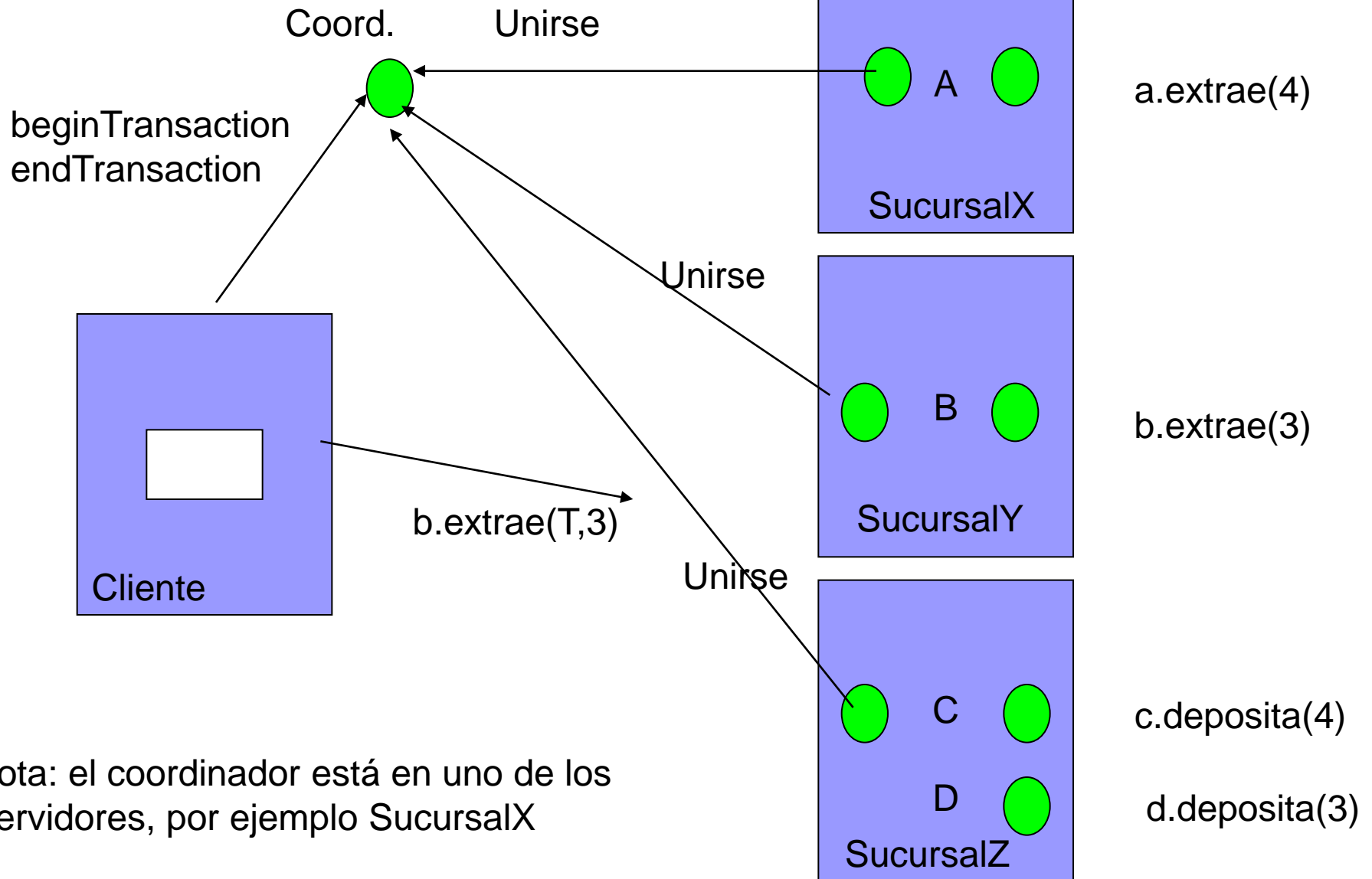
Sea una transacción distribuida donde el cliente transfiere \$10 de la cuenta *A* a *C* y \$20 de *B* a *D*. Las cuentas *A* y *B* están separadas en servidores *X* e *Y* y las cuentas *C* y *D* están en el servidor *Z*.

Si la transacción se estructura como un conjunto de cuatro transacciones anidadas, los cuatro requerimientos (dos depósitos y dos retiros) pueden correr en paralelo y el efecto total es lograr mayor rendimiento que una transacción simple ejecutando las cuatro operaciones secuencialmente.



Procesamiento de transacciones distribuidas

- Un cliente comienza una transacción enviando un *begin_transaction* al **coordinador** en cualquier servidor TPS. Este coordinador inicia la transacción y **devuelve el ID respectivo**.



Nota: el coordinador está en uno de los Servidores, por ejemplo SucursalX

Procesamiento de transacciones distribuidas

- ❖ El **coordinador** que inició la transacción es el **responsable final de consumirla o abortarla**.
- ❖ Durante el progreso de una transacción el coordinador registra la lista de referencias de participantes, y cada participante registra una referencia hacia el coordinador.

Procesamiento de transacciones distribuidas

- ♦ La operación *BeginTransaction* devuelve el **TID**. Los identificadores de las transacciones deben ser **únicos dentro del sistema distribuido**.
- ♦ Una forma sencilla de obtener identificadores únicos esto es que cada TID tenga dos partes: el identificador del servidor (e.g. dirección IP) y un número único dentro del servidor.

Procesamiento de transacciones distribuidas

- Existen dos aspectos importantes a considerar en las transacciones distribuidas:
 - La consumación de una transacción
 - Control de concurrencia
- Se supone que existe una comunicación entre TPSs a través de un **protocolo** de aplicación. Estos protocolos **se implementan sobre RPC o pase de mensajes.**

Transacciones Distribuidas

- ❖ Cuando una transacción distribuida termina, la propiedad de atomicidad exige que todos los servidores acuerden lo mismo (*commit*) o todos aborten (*abort*). Existen **protocolos para llegar a compromisos** (Two-Phase-Commit)
- ❖ Las transacciones distribuidas deben ser globalmente serializadas. Existen protocolos de **control de concurrencia distribuida**.

Protocolos de Consumación Atómica

- ♦ Cuando el coordinador recibe un requerimiento *Commit* de una transacción, tiene que asegurar:
 - ♦ *Atomicidad*: Todos los nodos se comprometen con los cambios o ninguno lo hace y cualquier otra transacción percibe los cambios en todos los nodos — o en ninguno.
 - ♦ *Aislamiento*: Los efectos de la transacción no son visibles hasta que todos los nodos hayan tomado la decisión irrevocable *commit* o *abort*.

Consumación en una fase

■ ***Commit*** de una fase atómico

Una manera simple de completar una transacción en forma atómica es que el cliente comunica el requerimiento de *commit* o abort al coordinador. El coordinador envía a todos los participantes la decisión de consumir o abortar y se mantiene enviando el requerimiento hasta que todos ellos respondan con un ACK indicando que han realizado la tarea.

El protocolo no contempla que la decisión de abortar venga de uno de los participantes o del coordinador. Sólo puede venir del cliente

Consumación en Dos Fases

- Permite que cualquier participante aborte su parte de la transacción.
- En la **primera fase** del protocolo cada participante vota para que la transacción sea consumada o abortada. Una vez que el participante ha votado **commit**, no se le permite que aborte. Por lo tanto, **antes de un participante votar por commit debe asegurarse de que será capaz de llevar a cabo su parte del protocolo de consumación, incluso si falla.**
- Se dice que un participante está en estado “preparado” si finalmente será capaz de consumir la transacción en proceso.

Consumación en Dos Fases

- En la **segunda fase** el coordinador recoge el resultado de las votaciones. Si alguno de los participantes vota por abortar, la decisión es abortar. Si todos los participantes votan consumir, ésta será la decisión.



Consumación en Dos Fases

- Si uno de los participantes o el Cliente decide abortar (antes de la solicitud del coordinador) se le informa al resto. No se activa el protocolo de consumación.

Protocolo de Consumación en dos Fases

Fase 1 (votación)

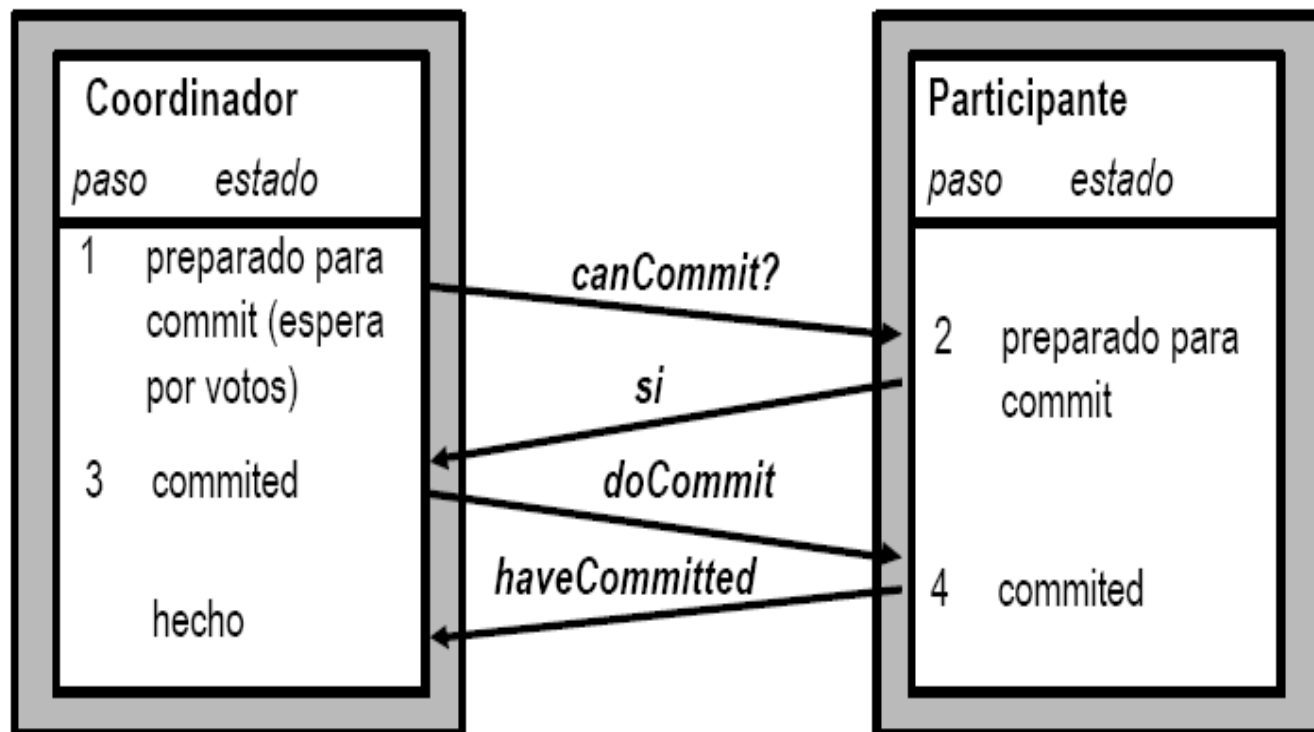
- 1. El Coordinador envía una petición (commit?) a cada participante en la transacción
- 2. Cuando un participante recibe una petición commit? Responde al Coordinador con su voto (sí o no). Antes de votar Sí se prepara para hacer commit **guardando los objetos en almacenamiento permanente**. Si el voto es No. El participante aborta de forma inmediata.

Protocolo de Consumación en dos Fases

Fase 2 (finalización en función del resultado de la votación)

- 3. El coordinador recoge los votos (incluyendo el propio)
 - Si no hay fallos y todos los votos son Sí, el coordinador decide consumir la transacción y envía peticiones de **COMMIT** a cada uno de los participantes.
 - En otro caso, el Coordinador decide abortar la transacción y envía peticiones **Aborta** a todos los que votaron Sí.
- 4. Los participantes que han votado Sí están esperando por una petición de Commit o Abort por parte del Coordinador. Cuando se reciben uno de estos mensajes, se actúa en función de ellos. En caso de COMMIT se retorna al servidor: *haveCOMMITTED*.

Comunicación en el protocolo de dos fases



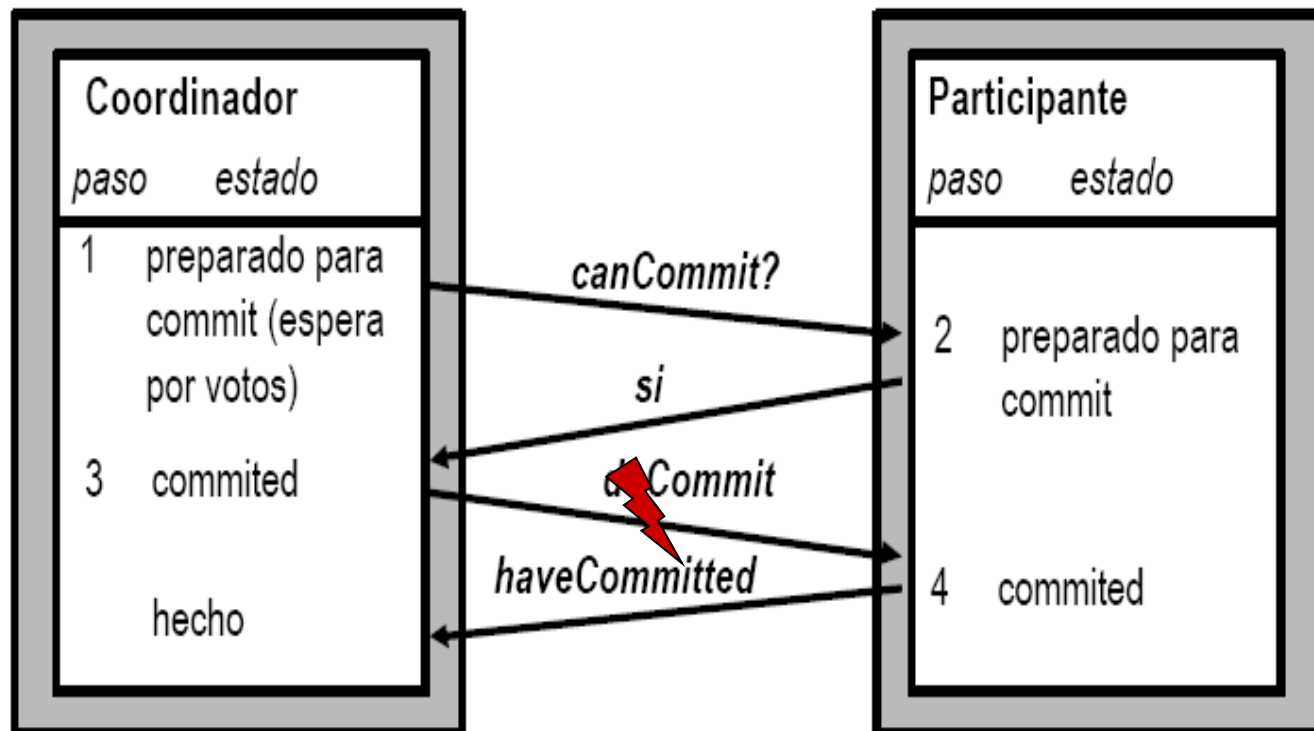


Protocolo de Consumación en dos Fases

- El protocolo podría fallar debido a la caída de uno o más servidores o debido a un corte en la comunicación.
- Para cubrir la posibilidad de caídas, cada servidor guarda la información correspondiente al protocolo en un dispositivo de almacenamiento permanente. Esta información la puede recuperar un nuevo proceso que se inicie para reemplazar al servidor caído.

Situación en la que un participante ha votado Sí y está esperando para que el coordinador le informe el resultado de la votación

Comunicación en el protocolo de dos fases



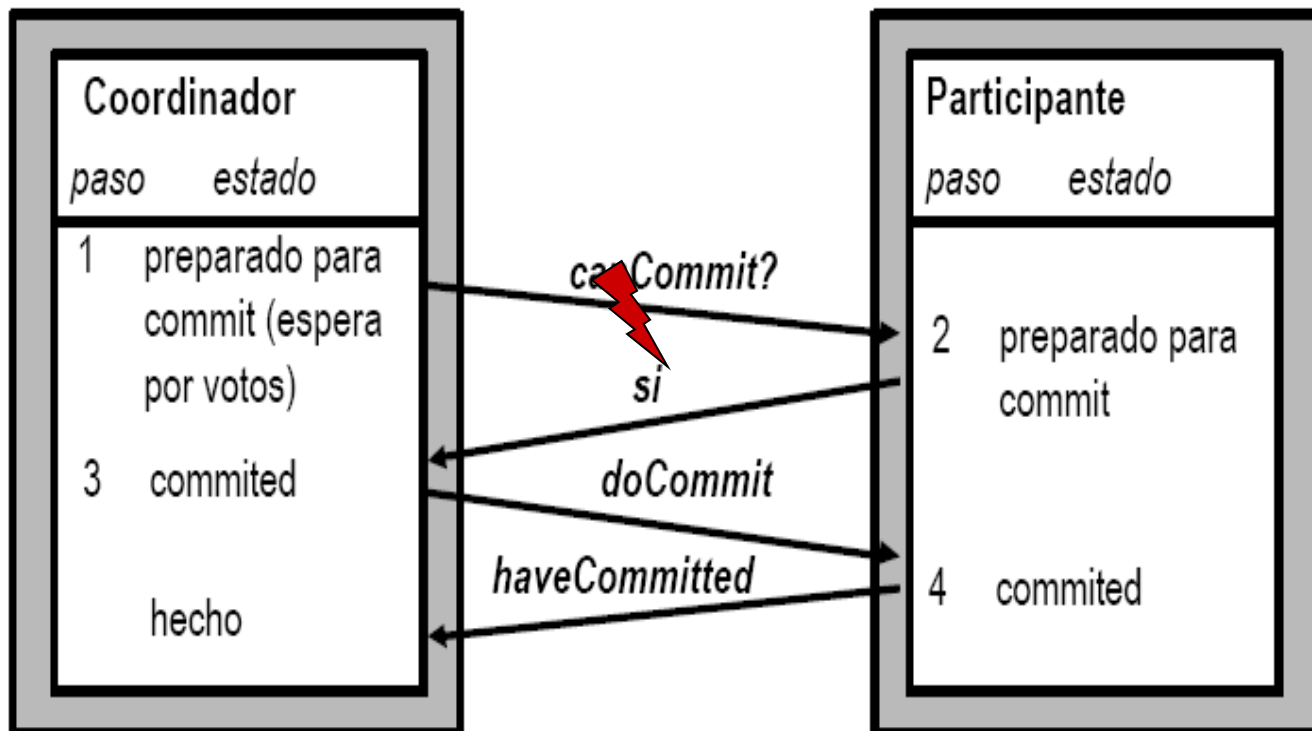
Fallas en la Comunicación

El **participante** está **incierto** frente al resultado y no puede seguir adelante hasta que obtenga los resultados de la votación por parte del coordinador.

No **puede decidir unilateralmente**, debe **mantener los objetos**. Envía un mensaje al Coordinador de *DameDecision*. Cuando obtiene la respuesta continua en el paso 4 del protocolo. Si el Coordinador ha fallado el participante no podrá obtener una decisión hasta que el servidor sea reemplazado.

Situación en la que un participante no ha recibido la solicitud por parte del Coordinador para consumir una transacción.

Comunicación en el protocolo de dos fases





Acciones frente a un *timeout*

Ya que el cliente envía el *endTransaction* sólo al Coordinador, el participante sólo puede detectar esta situación, cuando se da cuenta de que **no ha recibido ninguna petición asociada a una transacción por un largo periodo de tiempo (*timeout*)**

En este caso, como no se ha tomado ninguna decisión, **el participante puede decidir unilateralmente abortar.**



Acciones frente a un *timeout*

El coordinador puede sufrir retrasos cuando está esperando por los votos de los participantes.

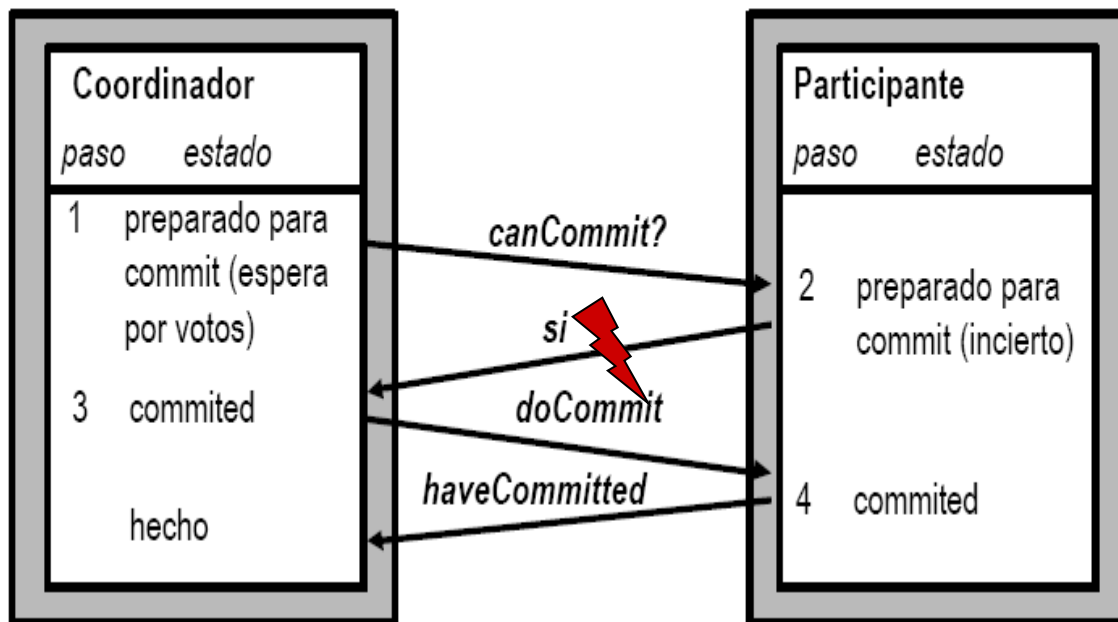
Dado que no está decidido el destino de la transacción, tras cierto periodo de tiempo puede abortar. En ese momento, el Coordinador anuncia su decisión a todos los participantes que ya habían votado. Algunos participantes retrasados pudieran votar Sí, sin haber recibido el último mensaje del coordinador. El coordinador ignora este Sí y el participante pasa al estado incierto.

El coordinador puede sufrir retrasos cuando está esperando por los votos de los participantes.

Dado que no se ha tomado una decisión, el coordinador puede decidir abortar unilateralmente.

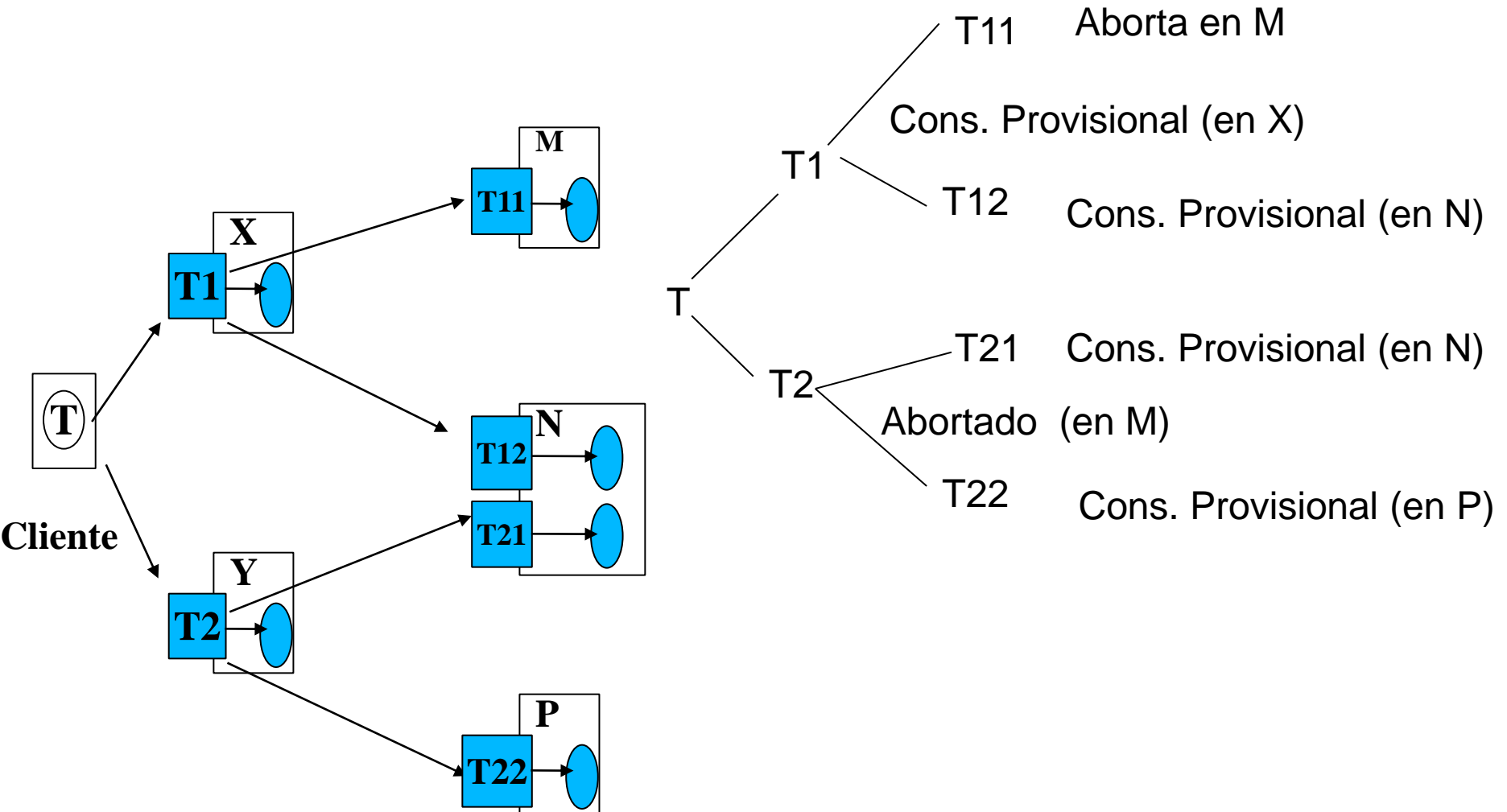
Envía Abortar a los participantes

Comunicación en el protocolo de dos fases



Si llega un Sí, de los que estaban atrasados, se ignora.
El participante queda en el estado incierto descrito Anteriormente.

Transacciones Anidadas



Transacciones Anidadas

- Cuando finaliza una subtransacción toma una decisión independiente sobre si consumarse de forma provisional (no es lo mismo que estar preparado) o abortar. *Una **consumación provisional** es simplemente una decisión local y no se guarda copia en un dispositivo de almacenamiento permanente.*
- Las transacciones trabajan y, cuando terminan, el servidor donde se encuentran registra información sobre si se han consumado provisionalmente o han abortado.



Transacciones Anidadas

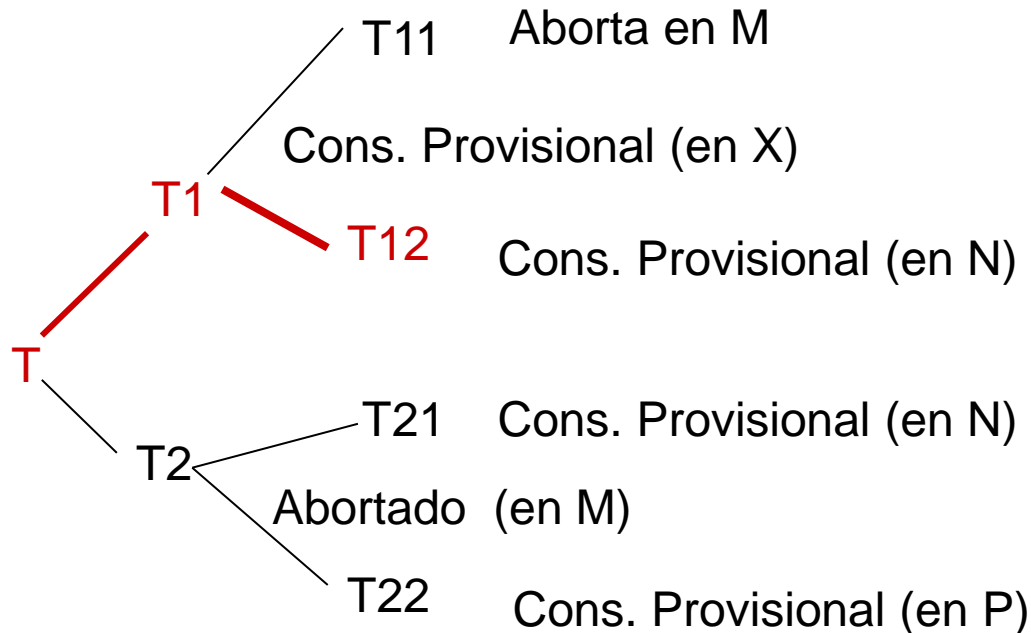
- Cuando una transacción anidada se consuma en forma provisional informa de su estado y del estado de sus descendientes a su madre.
- Cuando una transacción aborta, simplemente informa de haber abortado a su madre.
- Al final, la transacción a nivel superior recibe una lista de todas las subtransacciones en el árbol junto con el estado de cada una de ellas.



Transacciones Anidadas

- La transacción a nivel superior juega el papel de Coordinador en el protocolo de consumación en 2 fases.

Transacciones Anidadas



La lista de participantes consta de los servidores de todas las subtransacciones que se hayan consumado provisionalmente pero no tienen ascendentes que hayan abortado: T, T1, T12



Transacciones Anidadas

- La transacción a nivel superior debe intentar consumir lo que quede del árbol. A T1 y T12 se les pedirá que voten para obtener el resultado.
- Si votan consumir deben preparar las transacciones guardando el estado de los objetos en el dispositivo de almacenamiento permanente.
- La segunda fase es idéntica: se recogen los votos y se informa a los participantes en función del resultado.



Control de Concurrency: Bloqueos

- En una transacción distribuida los bloqueos se mantienen localmente (en el mismo servidor).
- El administrador local de bloqueos puede decidir si otorga un bloqueo o hace que la transacción que lo requirió espere.
- Sin embargo no puede liberar ningún bloqueo mientras la transacción que los tiene no haya terminado (commit o abort) en todos los servidores involucrados en la transacción.
- Cuando se usa el bloqueo para control de concurrencia, los objetos permanecen bloqueados y no están disponibles para otras transacciones durante el protocolo de commit atómico, aunque una transacción abortada libere sus bloqueos durante la primera fase del protocolo.

Bloqueos

Dado que los administradores de bloqueos en diferentes servidores otorgan bloqueos independiente de los demás, es posible que diferentes servidores impongan diferente orden sobre las transacciones.

Considérese el siguiente entrelazado de las transacciones T y U en los servidores X e Y :

T	U
$Write(A)$ en X lock A	
	$Write(B)$ en Y lock B
$Read(B)$ en Y espera U	
	$Read(A)$ en X espera por T

El objeto A está en el servidor X y el objeto B en el servidor Y

Bloqueos

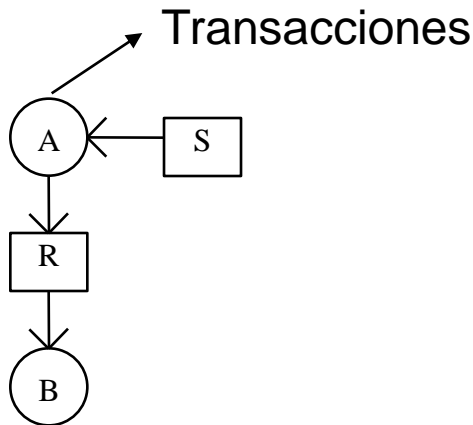
Se tiene T antes que U en un servidor y U antes que T en el otro.

- Cuando se detecta una condición de interbloqueo las transacciones son abortadas.
- En este caso el coordinador será informado y abortará las transacciones en los participantes involucrados.

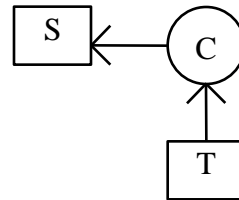
Algoritmos de Detección

1.- Centralizado: basado en grafos de espera

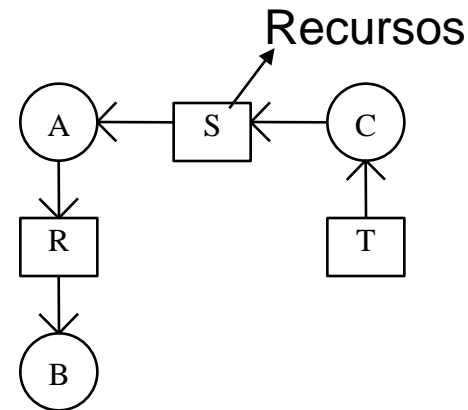
Máquina 0



Máquina 1



Coordinador



Cómo se mantiene el grafo en el coordinador ?

- Cada vez que ocurra una variación en su grafo notifica al coordinador .
- Periodicamente cada máquina notifica sus últimos cambios .
- Periodicamente el coordinador solicita la información .

Problema: Los 3 casos pueden conducir a un Deadlock falso.

Ejemplo: Si se pierden mensajes

Si B solicita a T y C libera a T y llega primero el mensaje de B al coordinador, entonces el cree que hay deadlock.



Algoritmo Distribuido (Caza de Arcos)

- En esta aproximación el grafo no se construye en forma global, sino que cada uno de los servidores implicados posee información sobre alguno de sus arcos.
- Los servidores intentan encontrar ciclos mediante el envío de mensajes denominados sondas.

Pasos del Algoritmo

- Iniciación: Cuando un servidor percibe que una transacción T, espera por un recurso que tiene una transacción U (que está en otro servidor), inicia el algoritmo enviando una sonda que contiene el arco $\langle T \rightarrow U \rangle$ al servidor que contiene el objeto por el cual está bloqueada la transacción U.
- Si U está compartiendo el bloqueo (varias transacciones acceden al mismo objeto), se envía la sonda a los servidores responsables de estas transacciones

Pasos del Algoritmo

- Detección: consiste en recibir sondas y decidir si se ha producido inter-bloqueo.

Por ejemplo, si un servidor recibe la sonda $\langle T \rightarrow U \rangle$ (T espera por U, que tiene el bloqueo de un objeto local), comprueba si U está también esperando. Si es así, se añade a la sonda la transacción por la que está esperando, ej, V. $\langle T \rightarrow U \rightarrow V \rangle$, y si V está esperando por un objeto en otro sitio se vuelve a reenviar la sonda.

Pasos del Algoritmo

- Detección: Antes de reenviar la sonda, el servidor comprueba si la transacción que ha sido añadida, ej, T

□ $\langle T \rightarrow U \rightarrow V \rightarrow T \rangle$

Ha ocasionado un ciclo, si es así se ha detectado un interbloqueo.



Pasos del Algoritmo

- Resolución: cuando se detecta un ciclo, se aborta una transacción en el ciclo para romper el interbloqueo.



Algoritmo de Detección de Bloqueos Distribuido

El coordinador de una transacción es responsable de registrar si la transacción está activa o está esperando por un objeto concreto, y los participantes pueden obtener esta información desde su coordinador. Los gestores de bloqueos informan a los coordinadores cuando las transacciones comienzan a esperar por objetos, y en el momento que adquieren dichos objetos para comenzar a estar activas.



Algoritmo de Detección de Bloqueos Distribuido

Cuando se aborta una transacción para romper un inter-bloqueo, el coordinador informará a los participantes y se eliminarán todos sus bloqueos, con el efecto de que todos los arcos relacionados con esta transacción se eliminarán de los grafos *espera-por* locales.



Control de Concurrency Optimista

■ Recordar:

- ☐ Cada transacción se valida antes de que se le permita consumarse.
- ☐ Se asignan unos números de transacción al comienzo de la validación y se establece un orden o secuencia de acuerdo a estos números.



Control de Concurrency Optimista

- Una transacción Distribuida es validada por una colección de servidores independientes, cada uno de los cuales valida las transacciones que acceden a sus propios objetos.
- La validación de todos los servidores tiene lugar durante la primera fase del protocolo de consumación de dos fases.



Control de Concurrency Optimista

En el caso de transacciones distribuidas optimistas, cada servidor aplica en paralelo un protocolo de validación. Esta es una extensión de la validación hacia delante o hacia atrás para permitir que varias transacciones estén en la fase de validación (por lo que pueden tardar estas fases en un entorno distribuido).

En esta extensión se debe comprobar tanto la regla 3 como la regla 2 ó 1.

Control de Concurrency Optimista

Sean las transacciones T y U entrelazadas, las cuales acceden a los objetos A y B en los servidores X e Y respectivamente:

T

Read(A) en X

Write(A)

Read(B) en Y

Write(B)

U

Read(B) en Y

Write(B)

Read(A) en X

Write(A)

Control de Concurrency Optimista

- Las transacciones acceden a los objetos en el orden T antes que U en el servidor X y U antes que T en el servidor Y .
- Si se supone que T y U empiezan la validación al mismo tiempo, el servidor X valida T primero y el servidor Y valida U primero. No se cumple la equivalencia secuencial

Control de Concurrency Optimista

- Los servidores de transacciones distribuidas deben evitar que suceda T antes que U en un servidor y U antes que T en otro.

Sol: después de una validación local, se llevará a cabo una validación global (antes de la consumación). La validación global comprueba que el orden en los servidores sea secuencialmente equivalente.



Control de concurrencia con ordenación de Marcas Temporales.

- En transacciones distribuidas se requiere que cada coordinador genere una única marca de tiempo global.
- Se consigue la equivalencia secuencial consumando las versiones de los objetos en el orden de las marcas temporales de las transacciones que acceden a ellos.
- Se requiere que cada coordinador genere marcas temporales que sean globalmente únicas.
- Esta marca de tiempo es dada al cliente por el primer coordinador accedido por la transacción.



Time Stamps

- La marca de tiempo de la transacción se pasa al coordinador de cada servidor en los cuales se realizan operaciones de la transacción.
- Una marca temporal consta de: <marca temporal local, identificador local del servidor>



Time Stamps

- Se requiere que las marcas de tiempo proporcionadas por un coordinador estén aproximadamente sincronizadas con aquellas que emiten otros coordinadores.
- Las marcas se pueden mantener sincronizadas mediante la utilización de algoritmos de sincronización de relojes.



Recuperación Distribuida

- Hemos supuesto que: Cuando un servidor está en funcionamiento, mantiene todos sus objetos en la memoria volátil y registra sus objetos consumados en un archivo o archivos de recuperación.
- La recuperación consiste en restaurar el servidor a partir de los dispositivos de almacenamiento permanente y dejarlo con las últimas versiones consumadas de los objetos.



Recuperación Distribuida

- Los requisitos de persistencia y atomicidad ante fallos se tratan mediante un único mecanismo: el gestor de recuperación. Las tareas de un gestor de recuperación son:
 - Guardar los objetos de todas las transacciones consumadas en dispositivos de almacenamiento permanente.
 - Restaurar los objetos del servidor tras una caída.



Recuperación Distribuida

- ☐ Reorganizar el archivo de recuperación para mejorar el rendimiento de la recuperación.
- ☐ Reclamar espacio de almacenamiento.
- Se requiere que el gestor de recuperación sea resistente a los fallos en los medios (discos, etc). Se debe manejar al menos una copia del archivo de recuperación.



Recuperación Distribuida

Listas de Intenciones:

- Durante el progreso de una transacción, las operaciones de actualización se aplican sobre un conjunto privado de versiones tentativas de los objetos que pertenecen a la transacción.
- En cada servidor se guarda una lista de intenciones para cada una de sus transacciones activas.



Recuperación Distribuida

- La lista de intenciones de una transacción contiene una lista de las referencias y los valores de los objetos que son alterados durante la transacción.
- Cuando una transacción se consuma se utiliza la lista de intenciones para identificar los objetos afectados. Se reemplaza la versión tentativa del objeto por una versión consumada.




Recuperación Distribuida

- En el protocolo de consumación de dos fases un participante dice que está preparado para hacer COMMIT.
- Cuando un participante dice que está preparado para hacer un commit, su gestor de recuperación debe haber guardado en su archivo de recuperación su lista de intenciones, de tal forma que después pueda llevar a cabo el commit, aún si fallara el servidor donde reside.



Recuperación Distribuida

- Entradas del Archivo de Recuperación:
 - Objeto: el valor del objeto
 - Estado de la transacción: Identificador de la transacción, estado de la transacción (preparada, consumada, abortada)
 - Lista de intenciones: Identificador de la transacción y una secuencia de intenciones, cada una de las cuales consiste en: identificador del objeto, posición en el archivo de recuperación.



Registro Histórico

- El archivo contendrá una instantánea reciente de los valores de todos los objetos seguido de un historial de las transacciones.
- Cuando el servidor está preparado para consumir una transacción, el gestor añade al archivo todos los objetos en su lista de intenciones seguido por el estado actual de la transacción (preparada).
- Cuando una transacción se consuma o aborta, se añade un nuevo registro con el estado de la transacción

Registro Histórico

P0

P1

P2

P3

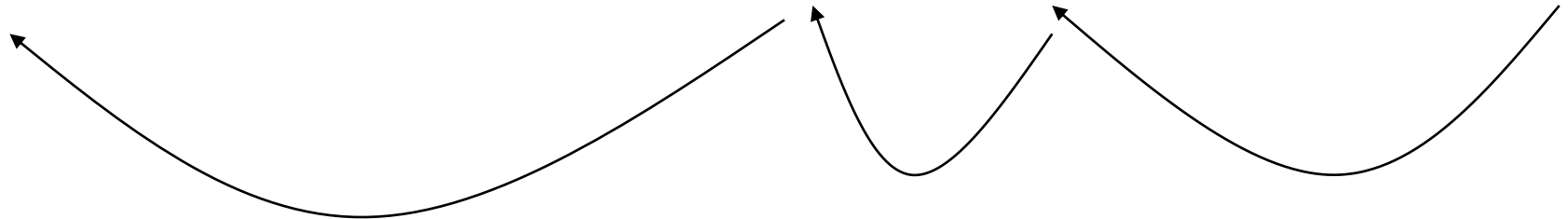
P4


P5

P6

P7


Objeto: A 100	Objeto: B 200	Objeto: C 300	Objeto: A 80	Objeto: B 220	Trans: T Preparad a <A,P1> <B,P2> P0	Trans:T Consumada	Objeto: C 278	Objeto: B 242	Trans: U Preparad a <C,P5> <B,P6> P4
------------------	------------------	------------------	-----------------	------------------	---	----------------------	------------------	------------------	---





Registro Histórico

- Tras una caída se aborta cualquier transacción que no tenga el estado de consumada.
- Cada entrada del estado de una transacción contiene un apuntador a la posición en el archivo de recuperación de la entrada que contiene el estado anterior a la transacción.



Registro Histórico

- Aproximaciones para restaurar:
 - Desde el comienzo del archivo. Las transacciones se repiten en el orden en el cual fueron ejecutadas.
 - Se recuperan los objetos leyendo el archivo de recuperación hacia atrás. Se utilizan las transacciones de estado consumadas para restaurar objetos que no han sido restaurados. Cada objeto sólo se restaura una vez.

Registro Histórico


P0	P1		P2	P3	P4	P5	P6	P7	
Objeto: A 100	Objeto: B 200	Objeto: C 300	Objeto: A 80	Objeto: B 220	Trans: T Preparad a <A,P1> <B,P2> P0	Trans:T Consumada	Objeto: C 278	Objeto: B 242	Trans: U Preparad a <C,P5> <B,P6> P4

The diagram illustrates a historical record of transactions across eight processes (P0 to P7). The record is organized into a table with columns for each process. The data in the table is as follows:

P0	P1	P2	P3	P4	P5	P6	P7		
Objeto: A 100	Objeto: B 200	Objeto: C 300	Objeto: A 80	Objeto: B 220	Trans: T Preparad a <A,P1> <B,P2> P0	Trans:T Consumada	Objeto: C 278	Objeto: B 242	Trans: U Preparad a <C,P5> <B,P6> P4

Arrows indicate dependencies between processes:

- A curved arrow from the bottom of the P0 column to the bottom of the P4 column.
- A curved arrow from the bottom of the P4 column to the bottom of the P5 column.
- A curved arrow from the bottom of the P5 column to the bottom of the P7 column.

- 
- En el ejemplo la recuperación se hará así:
 - Comienza en P7, concluye que U no se ha consumado y sus efectos se deben borrar.
 - Se mueve a P4 y concluye que T se había consumado. Se mueve a P3 y restaura A y B a partir de la lista de intenciones. Ya que no ha restaurado C se mueve hacia P0, que es el punto de control y restaura C.



Versiones Sombra

- Es una forma alternativa de organizar el archivo de recuperación.
- Utiliza un mapa para localizar las versiones de los objetos dentro de otro archivo.
- Cuando una transacción está preparada para hacer commit, se añaden los objetos cambiados al almacén de versiones dejando las versiones consumadas sin cambios.



Versiones Sombra

- Las versiones tentativas, se denominan versiones sombra. Cuando una transacción se consume se hace un nuevo mapa que reemplaza al anterior.
- Para restaurar los objetos el gestor de recuperación lee el mapa y utiliza su información para localizar los objetos en el archivo de versiones.
- El mapa debe estar siempre en un lugar bien conocido: un archivo separado, al comienzo del almacén de versiones.

Versiones Sombra

Mapa al Inicio	Mapa cuando T se consume
A -> P0 B -> P0' C -> P0''	A -> P1 B -> P2 C -> P0''

p0	P0'	P0''	p1	p2	p3	p4
100	200	300	80	220	278	242