UNIDAD III

CONCURRENCIA

3.1 DEFINICION

La concurrencia es un fenómeno que se presenta en varios contextos. Uno de ellos es la multiprogramación ya que el tiempo del procesador es compartido dinámicamente por varios procesos. Otro caso son las aplicaciones estructuradas, donde la programación estructurada se implementa como un conjunto de procesos concurrentes. Y por ultimo se tiene que la misma estructuración recién mencionada es utilizada en el diseño de los sistemas operativos, los cuales se implementan como un conjunto de procesos.

El termino concurrencia se refiere al hecho de que los DBMS(SISTEMAS DE ADMINISTRACION DE BD)permiten que muchas transacciones puedan accesar a una misma base de datos a la vez..

En un sistema de estos se necesitan algún tipo de mecanismos de control de concurrencia para asegurar que las transacciones concurrentes no interfieran entre si.

En sistemas multiusuario, es necesario un mecanismo para controlar la concurrencia. Se pueden producir inconsistencias importantes derivadas del acceso concurrente, como por ejemplo, el problema de la operación perdida.

En un sistema de biblioteca, existe un campo que almacena el numero de copias disponibles para préstamo. Este campo debe incrementarse en uno cada vez que se devuelve un ejemplar del libro y disminuirse en uno cada vez que se presta un ejemplar.

Si existen varias bibliotecarias, una de ellas inicia la transacción t1, leyendo la variable numero ejemplares (n), cuyo contenido se guarda en la variable n1. Tiempo después, otra bibliotecaria podría leer la misma variable incrementándola en una unidad , transacción t2. Después, la transacción t1 añade una unidad a esa variable y la actualiza, el resultado es erróneo, ya que la variable N debería haber aumentado en 2 unidades, y solo ha aumentado en una. La transacción t2 se ha perdido.

Consiste en controlar la interacción entre los usuarios concurrentes para no afectar la inconsistencia de los datos.

Cuando se trabajaba con manejadores de archivos y se estaba actualizando un determinado registro, ningún otro usuario podía actualizar el mismo archivo; aunque el registro a trabajar fuera otro. Con esta característica, ya más de un usuario puede accesar a un mismo registro y actualizarlo.

El objetivo fundamental del control de concurrencia de base de datos es garantizar que la ejecución concurrente de transacciones no resulta en una perdida de consistencia de la base de datos.

La concurrencia se da cuando dos transacciones están interconectadas, lo que es común en un entorno multiusuario. Así pues en un entorno de multiprogramación es posible ejecutar varias transacciones de manera concurrente.

Ejemplo

3.2 PROBLEMAS QUE SE PRESENTAN

1) Modificación perdida:

En T1 (Tiempo 1), arranca TA (Transacción A), leen dato X

En T2 (Tiempo 2), arranca TB (Transacción B), leen dato X datos = 100

En T3 (Tiempo 3), modifica TA (Transacción A), dato X (aumenta el 100%) datos = 200

En T4 (Tiempo 4), modifica TB, dato X (en base a lo que leyó en T2) (aumenta el 50%) 150 datos final.

2)Dependencia no COMMITADA

Permitir leer un dato sin esperar que una transacción que la estaba modificando haga su commit.

3) Análisis consistente:

TA (Transacción A): suma saldos

TB (Transacción B): transfiere $10 de cuenta 1 a cuenta3

CUENTA 1	**50**
CUENTA 2	**40**
CUENTA 3	**30**

TA CUENTA 1 = $50

T1 SALDO = $50

TA CUENTA 2 = $40

T2 SALDO 2 = $90

TB CUENTA 1 = $50

T3 CUENTA 1 = $50 − $10

CUENTA 1 = $40

TB CUENTA 3 = $30 + $10

T4 CUENTA 3 = $40

TA CUENTA 3 = $40

T5 SALDO = $130 *(EN REALIDAD ES $ 120)*

Para eliminar o disminuir estos 3 problemas se utilizan protocolos:

a) Bloqueos

- S compartido (para lecturas)
- X exclusivo, este puede ser por páginas/ tabla/ registros

TB/ TA	NO BLOQUEO	**S**	**X**
NO BLOQUEO	ACCEDE A OBJETO	ACCEDE A OBJETO	NO ACCEDE
S	ACCEDE A OBJETO	ACCEDE A OBJETO	NO ACCEDE
X	NO ACCEDE	NO ACCEDE	NO ACCEDE

Problemas con los bloqueos exclusivos: puede haber un bloqueo mortal o deadlock.

TA TB

I J

J I

Recursos

Políticas para terminar con el bloqueo mortal:

- Matar los dos
- Darles un tiempo de vida
- Matar a la más vieja
- Matar al azar

Existen tres formas en que una transacción, aunque sea correcta por si misma, puede producir una respuesta incorrecta si alguna otra transacción interfiere con ella en alguna forma.

Observe que la transacción que interfiere también puede ser correcta por si misma ;lo que produce el resultado incorrecto general ,es el intercalado sin control de las operaciones de las 2 transacciones correctas.

Actualización perdida:

Este problema puede presentarse si dos transacciones concurrentes actualizan una misma tupla, y la segunda que se realiza al final ,no toma en cuenta el efecto de la primera.

TRANSACCION A TIEMPO TRANSACCION B

Recuperar T T1

T2 recuperar T

Actualizar T T3

T4 Actualizar T

Dependencia no confirmada:

Ocurre si se permite que una transacción lea o modifique una tupla que ha sido modificada por otra transacción sin que se haya comandado el registro en firme de esta modificación,si ocurriera una operación de cancelación podrían generarse inconsistencias.

TRANSACCION A TIEMPO TRANSACCION B

T1 actualizar T

Recuperar T T2

T3 ROLLBACK

(La transacción A llega a ser dependiente de un cambio no confirmado en el tiempo T2)

TRANSACCION A TIEMPO TRANSACCION B

T1 actualizar T

Actualizar T T2

T3 ROLLBACK

(La transacción A actualiza un cambio no confirmado en el tiempo T2 y pierde esa actualización en el tiempo T3)

Análisis inconsistente:

Ocurre cuando una transacción hace un análisis contable o estadístico, sobre una tupla que esta siendo actualizada por otra transacción.

TRANSACCION A TIEMPO TRANSACCION B

Recuperar ACC1 T1

Suma = 40

Recuperar ACC2 T2

Suma = 90

T3 recuperar ACC3

T4 actualizar ACC3

30 20

T5 recuperar ACC1

T6 actualizar ACC1

- 50

T7 COMMIT

Recuperar ACC3 T8

Suma: 110 y no 120

TRANSACCION A

Suma saldos de cuenta ACC1: 40 50

ACC2: 50

TRANSACCION B

Transfiere una cantidad de 10

De la cuenta 3 a la 1 ACC3: 30 20

En los sistemas de tiempo compartido (aquellos con varios usuarios, procesos, tareas, trabajos que reparten el uso de CPU entre estos) se presentan muchos problemas debido a que los procesos compiten por los recursos del sistema. Los programas concurrentes a diferencia de los programas secuenciales tienen una serie de problemas muy particulares derivados de las características de la concurrencia:

- **Violación de la exclusión mutua:** En ocasiones ciertas acciones que se realizan en un programa concurrente no proporcionan los resultados deseados. Esto se debe a que existe una parte del programa donde se realizan dichas acciones que constituye una región critica, es decir, es una parte del programa en la que se debe garantizar que si un proceso accede a la misma, ningún otro podrá acceder. Se necesita pues garantizar la exclusión mutua.
- **Bloqueo mutuo o deadlock**: Un proceso se encuentra en estado de deadlock si esta esperando por un suceso que no ocurrirá nunca. Se puede producir en la comunicación de procesos y mas frecuentemente en la gestión de recursos. Existen cuatro condiciones necesarias para que se pueda producir deadlock:

    - Los procesos necesitan acceso exclusivo a los recursos.
    - Los procesos necesitan mantener ciertos recursos exclusivos mientras esperan por otros.
    - Los recursos no se pueden obtener de los procesos que están a la espera.
    - Existe una cadena circular de procesos en la cual cada proceso posee uno o mas de los recursos que necesita el siguiente proceso en la cadena.

- **Retraso indefinido o starvation:** Un proceso se encuentra en starvation si es retrasado indefinidamente esperando un suceso que no puede ocurrir. Esta situación se puede producir si la gestión de recursos emplea un algoritmo en el que no se tenga en cuenta el tiempo de espera del proceso.
- **Injusticia o unfairness**: Se pueden dar situaciones en las que exista cierta injusticia en relación a la evolución de un proceso. Se deben evitar situaciones de tal forma que se garantice que un proceso evoluciona y satisface sus necesidades sucesivas en algún momento.

- **Espera ocupada:** En ocasiones cuando un proceso se encuentra a la espera por un suceso, una forma de comprobar si el suceso se ha producido es verificando continuamente si el mismo se ha realizado ya. Esta solución de espera es muy poco efectiva, porque desperdicia tiempo de procesamiento, y se debe evitar. La solución ideal es suspender el proceso y continuar cuando se haya cumplido la condición de espera.
- **Condiciones de Carrera o Competencia:** La condición de carrera (race condition) ocurre cuando dos o mas procesos accesan un recurso compartido sin control, de manera que el resultado combinado de este acceso depende del orden de llegada.
- **Postergación o Aplazamiento Indefinido(a):** Consiste en el hecho de que uno o varios procesos nunca reciban el suficiente tiempo de ejecución para terminar a su tarea. Por ejemplo, que un proceso ocupa un recurso y lo marque como ocupado y que termine sin marcarlo como desocupado. Si algún otro proceso pide ese recurso, lo vera ocupado y esperara indefinidamente a que se desocupe.
- **Condición de Espera Circular**: Esto ocurre cuando dos o mas procesos forman una cadena de espera que los involucra a todos.
- **Condición de No apropiación**: Esta condición no resulta precisamente de la concurrencia, pero juega un papel muy importante en este ambiente. Esta condición especifica que si un proceso tiene asignado un recurso, dicho recurso no puede arrebatársele por ningún motivo, y estará disponible hasta que el proceso lo suelte por su voluntad.

Se debe evitar, pues, que dos procesos se encuentren en su sección critica al mismo tiempo.

3.3 SERIABILIDAD

La serialización es el criterio de lo correcto, para el control de la concurrencia. Un conjunto entrelazado de transacciones es correcto si es serializable. Es decir si produce el mismo resultado mediante la ejecución en serie de las mismas transacciones. Dado un conjunto de transacciones entrelazadas, cualquier ejecución de esas transacciones se dice que es una calendarización (scheduling)

Esta es la ejecución de esta aseveración:

1. − Las transacciones individuales son tomadas como correctas es decir, se da por hecho que transforman un estado correcto de la base de datos en otro estado correcto.

2. − Por lo tanto también es correcta la ejecución de una transacción a la vez en cualquier orden serial y se dice en cualquier orden serial debido a que las transacciones individuales son consideradas independientes entre sí.

3. − Por lo tanto una ejecución intercalada es correcta cuando equivale a una ejecución serial, es decir cuando es seriable.

Es la propiedad que garantiza que un plan de ejecución concurrente es equivalente al secuencial.

Formas de planificar la seriabilidad: **1) por conflicto 2) por visión**

Por simplicidad solo se consideran las operaciones de lectura y escritura. No se consideran las operaciones de cálculo sobre los datos obtenidos.

Seriabilidad por conflicto

- Eliminar conflictos entre dos o mas transacciones

- Operaciones sobre los mismos datos en mas de una transacción *
- Tipos de operaciones:

    - ♦ T1: lectura y T2: lectura
        - ◊ No hay conflicto
    - ♦ T1: lectura y T2: escritura ó T1: escritura y T2: lectura
        - ◊ Conflicto: hay que respetar el orden
    - ♦ T1: escritura y T2: escritura
        - ◊ Conflicto: el orden afecta al valor final de la BD

- Se dice que hay conflicto cuando se consideran operaciones sobre los mismos datos en dos transacciones diferentes

- Un plan de ejecución se puede transformar en otro cambiando de orden las instrucciones y manteniendo la seriabilidad

- Todos estos planes son equivalentes al plan secuencial.

Seriabilidad por visión

- Se basa en definir una regla de equivalencia menos estricta que la de conflicto.

- Pero basándose solo en las operaciones de lectura y escritura.

- Se puede considerar como un refinamiento de la equivalencia por conflicto.

Pruebas de seriabilidad

- Hacer la prueba de seriabilidad después de ejecutar el plan es un poco tarde.

- El objetivo es desarrollar un protocolo de control de concurrencia que asegure la seriabilidad.

- No suele analizar el grafo de precedencia.

- Lo habitual es imponer restricciones que aseguren la seriabilidad del plan.

Las pruebas sirven para ayudar a comprender los protocolos de control de concurrencia

VIOLACIONES A LA SERIALIZACIÓN

- Lectura sucia: la transacción ve información que no has sido consolidada (comandada)
- Lectura no repetible: la transacción ve dos valores diferentes del mismo objeto.
- Fantasmas: la transacción observa conjuntos diferentes de tuplas que satisface una condición.

La candelarización de las transacciones, una a la vez sin translape, se dice que es una calendarización serie. Dos calendarizaciones son equivalentes, si producen el mismo resultado en la base de datos. Una calendarización es correcta si es equivalente a una calendarización serie.

TEORIA DE LA SERIABILIDAD

Una calendarización (schedule), también llamado una historia, se define sobre un conjunto de transacciones T = { T1, T2, ..., Tn } y especifica un orden entrelazado de la ejecución de las operaciones de las transacciones.

La calendarización puede ser especificada como un orden parcial sobre T.

Ejemplo 6.1. Considere las siguientes tres transacciones:

*T*1: Read(*x*)	*T*2: Write(*x*)	*T*3: Read(*x*)
Write(*x*)	Write(*y*)	Read(*y*)
Commit	Read(*z*)	Read(*z*)
	Commit	Commit

Una calendarización de las acciones de las tres transacciones anteriores puede ser:

*H*1 = { *W*2(*x*), *R*1(*x*), *R*3(*x*), *W*1(*x*), *C*1, *W*2(*y*), *R*3(*y*), *R*2(*z*), *C*2, *R*3(*z*), *C*3 }

Dos operaciones *Oij*(*x*) y *Okl*(*x*) (*i* y *k* no necesariamente distintos) que accesan el mismo dato de la base de datos *x* se dice que están en *conflicto* si al menos una de ellas es una escritura. De esta manera, las operaciones de lectura no tienen conflictos consigo mismas. Por tanto, existen dos tipos de conflictos *read−write* (o *write−read*) y *write−write*. Las dos operaciones en conflicto pueden pertenecer a la misma transacción o a transacciones diferentes. En el último caso, se dice que las transacciones tienen *conflicto*. De manera intuitiva, la existencia de un conflicto entre dos operaciones indica que su orden de ejecución es importante. El orden de dos operaciones de lectura es insignificante.

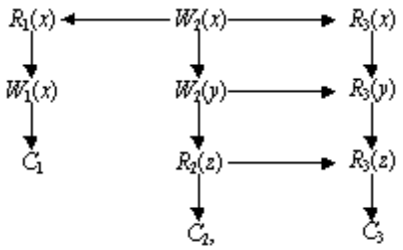
Una *calendarización completa* define el orden de ejecución de todas las operaciones en su dominio. Formalmente, una calendarización completa *STc* definido sobre un conjunto de transacciones *T* = { *T*1, *T*2, ..., *Tn* } es un orden parcial *STc* = { " *T*, <*T* } en donde

- " *T* = u i " *i* , para todos los *i* = 1, 2, ..., *n*
- <*T* " *i* <*i* , para todos los *i* = 1, 2, ..., *n*
- Para cualesquiera dos operaciones en conflicto *Oij* y *Okl* "" *T*, ó *Oij* <*T Okl* ó

Okl <*T Oij*

La primera condición establece simplemente que el dominio de la calendarización es la unión de los dominios de las transacciones individuales. La segunda condición define la relación de ordenamiento como un superconjunto de la relación de ordenamiento de transacciones individuales. Esto mantiene el ordenamiento de las operaciones dentro de cada transacción. La condición final define el orden de ejecución entre dos operaciones en conflicto.

Ejemplo 6.2. Considere las tres transacciones del Ejemplo 6.1, una posible calendarización completa está dada por la siguiente gráfica dirigida acíclica (DAG).



Una *calendarización* se define como un prefijo de una calendarización completa.

Un prefijo de un orden parcial se define como sigue. Dado un orden parcial *P* = { " , < }, *P*' = { " ', <' }, es un *prefijo* de *P* si

- " ' " " *i*
- ð "*ei* " " ', *e*1 <' *e*2, si y solamente si, *e*1 < *e*2, y
- ð "*ei* " " ', si "ð *ej* " " y *ej* < *ei*, entonces, *ej* ""

Las primeras dos condiciones definen a *P*' como una restricción de *P* en el dominio " ', en donde las relaciones de ordenamiento en *P* se mantienen por *P*'. La última condición indica que para cualquier elemento de " ', todos sus predecesores en " deben ser incluidos también en " '.

Ejemplo 6.3. La siguiente calendarización es un prefijo de la calendarización del Ejemplo 6.2.

Si en una calendarización *S*, las operaciones de varias transacciones no están entrelazadas, esto es, si las operaciones de una transacción ocurren de manera consecutiva, entonces se dice que la calendarización es *serial*. Si cada transacción es consistente (obedece las reglas de integridad), entonces la base de datos se garantiza ser consistente al final de la calendarización serial. La historia asociada a este tipo de calendarización se le conoce como *serial.*

Ejemplo 6.4. La siguiente es una historia serial para el Ejemplo 6.1.

HS = { *W*2(*x*), *W*2(*y*), *R*2(*z*), *C*2, *R*1(*x*), *W*1(*x*), *C*1, *R*3(*x*), *R*3(*y*), *R*3(*z*), *C*3 }

Las transacciones se ejecutan de manera concurrente, pero el efecto neto de la historia resultante sobre la base de datos es *equivalente* a alguna *historia serial*. Bassada en la relación de precedencia introducida por el orden parcial, es posible discutir la equivalencia de diferentes calendarizaciones con respecto a sus efectos sobre la base de datos.

Dos calendarizaciones, *S*1 y *S*2, definidas sobre el mismo conjunto de transacciones *T*, se dice que son *equivalentes* si para cada par de operaciones en conflicto *Oij* y *Okl* (*i " k*), cada vez que *Oij* <1 *Okl*, entonces, *Oij* <2 *Okl*. A esta relación se le conoce como *equivalencia de conflictos* puesto que define la equivalencia de dos calendarizaciones en término del orden de ejecución relativo de las operaciones en conflicto en ellas.

Una calendarización *S* se dice que es *serializable*, si y solamente si, es equivalente por conflictos a una calendarización serial.

Ejemplo 6.5. Las siguientes calendarizaciones no son equivalentes por conflicto:

HS = { *W*2(*x*), *W*2(*y*), *R*2(*z*), *C*2, *R*1(*x*), *W*1(*x*), *C*1, *R*3(*x*), *R*3(*y*), *R*3(*z*), *C*3 }

*H*1 = { *W*2(*x*), *R*1(*x*), *R*3(*x*), *W*1(*x*), *C*1, *W*2(*y*), *R*3(*y*), *R*2(*z*), *C*2, *R*3(*z*), *C*3 }

Las siguientes calendarizaciones son equivalentes por conflictos; por lo tanto *H*2 es serializable:

HS = { *W*2(*x*), *W*2(*y*), *R*2(*z*), *C*2, *R*1(*x*), *W*1(*x*), *C*1, *R*3(*x*), *R*3(*y*), *R*3(*z*), *C*3 }

*H*2 = { *W*2(*x*), *R*1(*x*), *W*1(*x*), *C*1, *R*3(*x*), *W*2(*y*), *R*3(*y*), *R*2(*z*), *C*2, *R*3(*z*), *C*3 }

La función primaria de un controlador de concurrencia es generar una calendarización serializable para la ejecución de todas las transacciones. El problema es, entonces, desarrollar algoritmos que garanticen que únicamente se generan calendarizaciones serializables.

SERIABILIDAD EN SMBD DISTRIBUIDOS

En bases de datos distribuidas es necesario considerar dos tipos de historia para poder generar calendarizaciones serializables: la calendarización de la ejecución de transacciones en un nodo conocido como *calendarización local* y la *calendarización global* de las transacciones en el sistema. Para que las transacciones globales sean serializables se deben satisfacer las siguientes condiciones:

- cada historia local debe ser serializable, y
- dos operaciones en conflicto deben estar en el mismo orden relativo en todas las historias locales donde las operaciones aparecen juntas.

La segunda condición simplemente asegura que el orden de serialización sea el mismo en todos los nodos en donde las transacciones en conflicto se ejecutan juntas.

Ejemplo 6.6. Considere las siguientes tres transacciones:

*T*1: Read(*x*)	*T*2: Read(*x*)
x ! x + 5	x ! x * 5
Write(*x*)	Write(*x*)
Commit	Commit

Las siguientes historias locales son individualmente serializables (de hecho son seriales), pero las dos transacciones no son globalmente serializables.

*LH*1 = { *R*1(*x*), *W*1(*x*), *C*1, *R*2(*x*), *W*2(*x*), *C*2 }

*LH*2 = { *R*2(*x*), *W*2(*x*), *C*2, *R*1(*x*), *W*1(*x*), *C*1 }

3.4.1 TIPO DE SEGUROS

Intervalo de Seguro.− El empleo de seguros tiene una dimensión temporal. Un enfoque de protección consiste en asegurar cada objeto leído, o necesario para actualización posterior tanto como sea posible; cuando la transacción se concluye, todos los seguros se liberarán.

El intervalo de seguros puede acortarse liberando los seguros de un objeto en cuanto se realice la actualización. Esto significa que cada objeto que vaya a leerse tendrá que liberarse si se considera una actualización. Cuando se utiliza el sistema es común asegurar durante el intervalo READ_TO_UPDATE_REWRITE.

Seguro Diferido.− La ampliación del concepto de transacción de dos fases al empleo de seguros puede evitar asegurar recursos hasta que la transacción esté totalmente definida. No asegurar la transacción de inmediato significa que existe una posibilidad de que otra transacción haya codificado los datos de manera que sea necesario volver a leer todos los datos que contribuyan al resultado. Cuando la transacción este lista para compromiso todos los datos se vuelve a leer los datos con seguros. El tiempo que los otros usuarios se ven excluidos se reduce, pero con un costo: es necesario leer dos veces cada registro.

Asegurar los dispositivos para reducir el intervalo de seguros.− El costo de una READ_TO_UPDATE extra, inmediatamente antes de REWRITE, es pequeño ya que se espera poca interferencia de la localización. Aunque se realice un READ el intervalo de seguro puede ser mucho menor si no hay otras transacciones que realicen acceso a los dispositivos que estén empleando y si se evitan las localizaciones.

La posibilidad de una localización entre READ_TO_UPDATE y REWRITE puede evitarse permitiendo que el seguro que se posiciona en el momento de la operación READ_TO_UPDATE, no solo excluye otros accesos al objeto si no que excluya también todos los otros accesos al dispositivo en su totalidad. En la tecnología de programación se dice que la secuencia de instrucciones entre READ_TO_UPDATE y la conclusión del REWRITE se vuelve una *sección critica.*

Especificación de seguro.− Es necesario que exista una forma de avisar al sistema de apoyo a la base de datos que se ha solicitado el seguro de un objeto. Seria de esperarse que el usuario final, El vendedor o el oficinista, solicitara la acción de aseguramiento. Comúnmente la solicitud de seguro la realiza un programador de aplicaciones que define a la transacción, el sistema de manejo de transacciones o el sistema de archivo, en respuesta a las operaciones solicitadas

Cuando se reclama un objeto el sistema nota la intención de usarlo. La reclamación. La reclamación es aceptable si no puede provocar problemas. La observación se anota(de nuevo utilizando un semáforo para evitar reclamaciones en conflicto) y la transacción puede proceder. Si se niega la reclamación la transacción tiene la posibilidad de continuar con otra tarea, tal vez para volver después o quedar en una línea de espera.

Cuando se logra el acceso a los objetos reclamado se colocan seguros. Puede haber retrasos, pero eventualmente una transacción que espere en la línea correspondiente a un objeto obtendrá el acceso al objeto reclamado y podrá proceder. La transacción coloca un seguro para lograr así el acceso exclusivo al objeto, y este seguro se conservara hasta que la transacción ejecute un comando RELEASE. Las reclamaciones conservadas por una transacción también se liberan cuando una transacción se concluye o aborta

Interacciones entre seguros.− Se mostró que es necesario proporcionar aseguramiento entre transacciones de actualización y entre consultas sólo de lectura que requieran resultados que puedan soportar una auditoria y transacciones de actualización. Las consultas sólo de lectura no interfieren unas con otros. El empleo de aseguramiento vuelve inaccesibles a una parte de la base de datos.

El manejo por bloques se logra al tener un semáforo de aseguramiento posicionado en el objeto de las operaciones.

Taxonomía de los mecanismos de control de concurrencia

El criterio de clasificación más común de los algoritmos de control de concurrencia es el tipo de primitiva de sincronización. Esto resulta en dos clases: aquellos algoritmos que están basados en acceso mutuamente exclusivo a datos compartidos (candados) y aquellos que intentar ordenar la ejecución de las transacciones de acuerdo a un conjunto de reglas (protocolos). Sin embargo, esas primitivas se pueden usar en algoritmos con dos puntos de vista diferentes: el punto de vista pesimista que considera que muchas transacciones tienen conflictos con otras, o el punto de vista optimista que supone que no se presentan muchos conflictos entre transacciones.

Los algoritmos *pesimistas* sincronizan la ejecución concurrente de las transacciones en su etapa inicial de su ciclo de ejecución. Los algoritmos *optimistas* retrasan la sincronización de las transacciones hasta su terminación. El grupo de algoritmos pesimistas consiste de algoritmos *basados en candados*, algoritmos *basados en ordenamiento por estampas de tiempo* y algoritmos *híbridos*. El grupo de los algoritmos optimistas se clasifican por basados en candados y basados en estampas de tiempo (Ver. Figura 6.1).

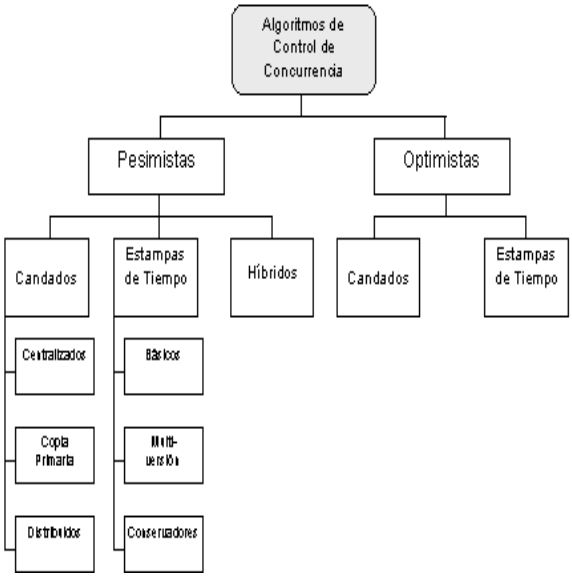


Figura 6.1. Clasificación de los algoritmos de control de concurrencia.

Algoritmos basados en candados

En los algoritmos basados en candados, las transacciones indican sus intenciones solicitando candados al despachador (llamado el *administrador de candados*). Los candados son de lectura (*rl*), también llamados *compartidos*, o de escritura (*wl*), también llamados *exclusivos*. Como se aprecia en la tabla siguiente, los candados de lectura presentan conflictos con los candados de escritura, dado que las operaciones de lectura y escritura son incompatibles.

	rl	*wl*
rl	si	no
wl	no	no

En sistemas basados en candados, el despachador es un *administrador de candados* (LM). El administrador de transacciones le pasa al administrador de candados la operación sobre la base de datos (lectura o escritura) e información asociada, como por ejemplo el elemento de datos que es accesado y el identificador de la transacción que está enviando la operación a la base de datos. El administrador de candados verifica si el elemento de datos que se quiere accesar ya ha sido bloqueado por un candado. Si candado solicitado es incompatible con el candado con que el dato está bloqueado, entonces, la transacción solicitante es retrasada. De otra forma, el candado se define sobre el dato en el modo deseado y la operación a la base de datos es transferida al procesador de datos. El administrador de transacciones es informado luego sobre el resultado de la operación. La terminación de una transacción libera todos los candados y se puede iniciar otra transacción que estaba esperando el acceso al mismo dato.

Bloqueo o seguro.

Se puede definir *bloqueo* (también llamado seguro o candado) como "una variable asociada a cada elemento de datos, que describe el estado de dicho elemento respecto a las posibles operaciones (recuperación o actualización) que se pueden realizar sobre ellos en cada momento".

Las transacciones pueden llevar a cabo bloqueos, por ejemplo, sobre los registros que vayan a utilizar, impidiendo a otros usuarios la recuperación o actualización de los elementos bloqueados, pudiéndose así evitar inconsistencias en el acceso concurrente.

Los bloqueos pueden ser de varios tipos:

− Bloqueos exclusivos (o de escritura): Cuando una transacción mantiene un bloqueo de este tipo sobre un objeto, ninguna otra transacción puede acceder a él, ni adquirir ningún tipo de bloqueo sobre ese objeto, hasta que sea liberado por la transacción que lo había retenido. Este tipo de bloqueos se utiliza cuando una transacción quiere actualizar algún objeto.

− Bloqueos compartidos (o de lectura): Cuando una transacción tiene sobre un objeto un bloqueo de tipo compartido, permite que otras transacciones retengan también ese mismo objeto en bloqueos compartidos, pero no exclusivos. Este tipo de bloqueo se utiliza cuando las transacciones no necesitan actualizar datos, pero quieren impedir cualquier modificación de éstos mientras son consultados.

Los bloqueos pueden colocarse, ya sea de manera automática por el DBMS o por medio de un comando emitido al DBMS partiendo del programa de aplicación del usuario de la consulta. Los bloqueos colocados por el DBMS se llaman bloqueos implícitos**,** los que son colocados por comando se llaman explícitos.

Los bloqueos también varían en tipo. Un bloqueo exclusivo cierra el elemento a un acceso de cualquier tipo. Ninguna otra transacción puede leer o modificar los datos. Un bloqueo compartido cierra elementos a modificaciones, no a la lectura. Otras transacciones pueden leer el elemento, siempre y cuando no intenten modificarlo.

De cualquier forma que el DBMS y el programa de aplicación lleven a cabo el bloqueo, deberán asegurarse que la base de datos no sufra ninguna anomalía debida al procesamiento concurrente.

Cuando dos o más operaciones se procesan en forma concurrente, los resultados en la base de datos deberán ser consistentes con los resultados que se habrían conseguido si las transacciones se hubieran procesado de una forma arbitraria y serial.

La capacidad de serializar puede conseguirse por varios medios. Una forma es procesar la transaccion utilizando un bloqueo de dos fases. Con tal estrategia, las transacciones tienen permitido obtener bloqueos según sea necesario, una vez que el primer bloqueo es liberado, ya no puede tener otro. Por lo tanto, las transacciones tienen una fase de crecimiento, en el cual se obtiene los bloqueos, y una fase de encogimiento, en la cual se liberan los bloqueos.

Los bloqueos se obtiene a todo lo largo de la transaccion, pero hasta que se emita el comando COMMIT o ROLLBACK no se libera ninguno.

- BLOQUEO O SEGURO DE DOS FASES: en este tipo de seguros, la transaccion le pone un seguro a un objeto antes de usarlo. Cuando un objeto es bloqueado con un seguro por otra transaccion, la transaccion solicitante debe esperar. Cuando una transaccion libera un candado, ya no puede solicitar mas candados. En la primera fase solicita y adquiere todos los bloqueos sobre los elementos que va a utilizar y en la segunda fase libera los bloqueos obtenidos uno por uno.

Puede sucede que si una transacción aborta despues de liberar un bloqueo, otras transacciones que hayan accesado el mismo elemento de datos aborten tambien provocandolo que se conoce como ***abortos en cascada.*** Para evitar lo anterior, los despachadores para seguros de dos fases implementan lo que se conoce como bloqueos estrictos de dos fases en los cuales se liberan todos los bloqueos juntos cuando la transacción termina (con COMMIT o aborta).

- BLOQUEO O SEGURO DE DOS FASES CENTRALIZADO: En sistemas distribuidos puede que la administración de los bloqueos se dedique a un solo nodo del sistema, por lo tanto, se tiene un despachador central el cual recibe todas las solicitudes de bloqueos del sistema. La comunicación se

presenta entre el administrador de transacciones del nodo en donde se origina la transacción, el administrador de bloqueos en el nodo central y los procesadores de datos de todos los nodos participantes. Los nodos participantes son todos aquellos en donde la operación se va a llevar a cabo.

- BLOQUEO O SEGURO DE DOS FASES DISTRIBUIDO: en los seguros de dos fases distribuidos se presentan despachadores en cada nodo del sistema. Cada despachador maneja las solicitudes de bloqueos para los datos en ese nodo.

- Una transacción puede leer cualquiera de las copias replicada del elemento x, obteniendo un bloqueo o seguro de lectura en cualquiera de las copias de x. La escritura sobre x requiere que se obtengan seguros para todas las copias de x.

- Los mensajes de solicitud de seguros se envían a todos los administradores de bloqueos que participen en el sistema. Las operaciones son pasadas a los procesadores de datos por los administradores de bloqueos. Los procesadores de datos envian su mensaje de fin de operación al administrador de transacciones coordinador.

3.4.2 PROTOCOLOS

PROTOCOLO DE BLOQUEO DE DOS FASES

Protocolo que asegura la secuencialidad es el protocolo de bloqueo de dos fases. Este protocolo exige que cada transacción realice las peticiones de bloqueo y desbloqueo de dos fases.

1.− Fase de crecimiento.− Una transacción puede obtener bloqueos pero no puede liberarlos.

2.− Fase de decrecimiento.− Una transacción puede liberar bloqueos pero no puede obtener ninguno nuevo.

Inicialmente una transacción está en la fase de crecimiento. La transacción adquiere los bloqueos que necesite. Una vez que la transacción entra un bloqueo, entra en la fase de crecimiento como puede alcanzar peticiones de bloqueo.

Se puede mostrar que el protocolo de bloqueo de dos fases asegura secuencialidad en cuanto a conflictos. Considérese cualquier transacción. El punto de la planificación en el cual la transacción obtiene el bloqueo final (el final de la fase de crecimiento) se denomina *punto de bloqueo* de la transacción.

El protocolo de bloqueo de dos fase no asegura la ausencia de interbloqueos.

Los retrocesos en cascada se pueden evitar por medio de una modificación del protocolo de dos fases que se denomina *protocolo de bloque estricto de dos fases.*

El *protocolo de bloque estricto de dos fases* exige que además de que el bloqueo sea de dos fases, una transacción debe poseer todos los bloqueos en modo exclusivo que tome hasta que dicha transacción no comprometida está bloqueado en modo exclusivo hasta que la transacción lea el dato..

Otra variante del bloqueo de dos fases es *el protocolo de bloqueo riguroso de dos fases*, el cual exige que se posean todos los bloqueos hasta que se comprometan la transacción. Se puede comprobar fácilmente que con el bloqueo riguroso de dos fases se pueden secuenciar las transacciones en el orden en que se comprometen.

Muchos sistemas de base de datos implementan tanto el bloqueo estricto como el bloqueo estricto de dos fases.

PROTOCOLOS BASADOS EN GRAFOS.

En ausencia de información acerca de la forma en que se accede a los elementos de datos, el protocolo de bloqueo de dos fases es necesario y suficiente para asegurar la secuencialidad. Así, si se desea desarrollar protocolos que no sean de dos fases, es necesario tener información adicional acerca de la forma en que cada transacción accede a la base de datos.

Existen varios modelos que diferensea en la cantidad de información que se proporciona. El modelo más simple que tenga conocimiento previo acerca con el cual accede a los elementos de la base de datos.

Dada la información es posible construir protocolos de bloqueos que no sean de dos fases pero que un obstante persigue la secuencialidad para adquirir al conocimiento previo que impone orden para el conjunto D={d1,.....,dn}.

El orden parcial implica que el conjunto **D** se puede ver como un grafo aciclico dirigido *grafo de la base de datos*. En este apartado, para simplificar, se centra la atención solo en aquellos grafos que sean árboles con raíz. Se puede presentar un protocolo simple llamado protocolo de árbol, el cual esta restringido a utilizar solo bloqueos de archivos.

Para ilustrar este protocolo, considérese el grafo de la base de datos de la Figura siguiente.

Las 4 transacciones siguientes siguen el protocolo de árbol sobre dicho grafo. Solo de muestran las instrucciones de bloqueo y desbloqueo:

T10: bloquear−X(B) ;bloquear−X(E); bloquear−X(D); desbloquear(B) ; desbloquear (E); bloquear−X(G); desbloquear (D); desbloquear (G).

T11: bloquear−X(D) ;bloquear−X(H); desbloquear(D) ; desbloquear (H).

T12: bloquear−X(B) ;bloquear−X(E); desbloquear (E); desbloquear (B).

T13: bloquear−X(D); bloquear−X(H); desbloquear (D); desbloquear (H).

- ***Si se quiere conseguir mayor grado de paralelismo hay que utilizar otro tipo de protocolos.***
- ***Los protocolos que no son del tipo anterior necesitan información adicional***
- ***Un modelo sencillo se basa en conocer el orden en que se accede a la información.***
- ***Si se puede establecer un orden parcial en el acceso a los datos, se puede aplicar un esquema de cerrojos mucho mas eficiente.***
- ***El orden parcial se puede representar en un grafo.***
- ***Los protocolos en árbol garantizan la seriabilidad y la ausencia de bloqueos***
- ***La liberación del cerrojo se puede producir antes en este protocolo que en el de bloqueo en dos fases.***
    - ♦ Tiempos de espera mas reducidos
    - ♦ aumento de la concurrencia
    - ♦ ausencia de bloqueos

- ***El inconveniente es que puede tener que bloquear datos que no necesita manejar***
    - ♦ aumenta el bloqueo, mayor tiempo de espera
    - ♦ potencial reducción de concurrencia

Funcionamiento: Se distinguen dos fases: votación y decisión.

En la primera el coordinador pregunta si están preparados para realizar (commit) la transacción. Si un participante vota abortar, o no responde dentro del timeout, entonces se aborta la transacción. Si todos votan para realizarla (commit), entonces el coordinador da instrucciones para realizar la transacción.

El periodo de timeout permite evitar el bloqueo de los nodos ante el fallo de uno de los participantes. Si un nodo falla en el proceso, todos siguen un protocolo de terminación, el nodo fallido sigue un *protocolo de recuperació*n.

Protocolo en árbol

- ***Los nodos representan los datos.***
- ***Los ejes el orden en que se acceden.***
- ***Es el protocolo de grafo mas sencillo***
- ***Utiliza solo cerrojos exclusivos***

1. El primer cerrojo de una transacción puede ser sobre cualquier dato.

2. Los siguientes datos se pueden bloquear si el padre del dato está bloqueado por la misma transacción.

3. Los datos se pueden desbloquear en cualquier momento.

4. Un dato que ha sido bloqueado y desbloqueado, no puede ser bloqueado de nuevo

PROTOCOLOS BASADOS EN MARCAS TEMPORALES

Otro método para determinar el orden de secuencialidad es seleccionar previamente un orden entre las transacciones en el método más común para hacer esto es utilizar un esquema de ordenación por marcas temporales

PROTOCOLO DE ORDENACIÓN POR MARCAS TEMPORALES

El protocolo de ordenación por marcas temporales asegura que todas las operaciones ***leer*** y ***escribir*** conflictivas se ejecutan en el orden de las marcas temporales.

El protocolo de ordenación por marcas temporales asegura la secuencialidad en cuanto conflictos. Esta afirmación se deduce del hecho de que las operaciones conflictivas se procesan durante la ordenación de las marcas temporales. El protocolo asegura la ausencia de interbloqueos, ya que ninguna transacción tiene que esperar. El protocolo puede generar planificaciones no recuperables; sin embargo se puede extender para producir planificaciones sin cascada.

Marcas de tiempo multiversión

El mecanismo de marcas de tiempo supone que existe una única versión de los datos; por lo que sólo una

transacción puede acceder a los mismos. Se puede relajar esta restricción permitiendo que varias transacciones lean y escriban diferentes versiones del mismo dato siempre que cada transacción *vea* un conjunto consistente de versiones de todos los datos a los que accede.

El problema con las técnicas de bloqueo es que puede producirse un *interbloqueo* (llamado *deadlock*), que es una situación que se da cuando dos o más transacciones están esperando cada una de ellas que otra libere algún objeto antes de seguir. Este problema, que ha sido estudiado en profundidad en los sistemas operativos, puede tener dos soluciones:

− Prevenir el interbloqueo, obligando a que las transacciones bloqueen todos los elementos que necesitan por adelantado.

− Detectar el interbloqueo, controlando de forma periódica si se ha producido, para lo que suele construirse un *grafo de espera*. Si existe un ciclo en el grafo se tiene un interbloqueo. Cada SGBD tiene su propia *política* para escoger las víctimas, aunque suelen ser las transacciones más recientes.

PROTOCOLOS BASADOS EN VALIDACION.

Para aquellos casos en los que la mayoría de las transacciones son de solo lectura. La tasa de conflictos entre las transacciones puede ser baja.

Así muchas de esas transacciones, si se ejecutasen sin la supervisión en un sistema de control de concurrencia, llevarían no obstante al sistema a un estado consistente.

Un esquema de control de concurrencia supone una sobrecarga en la ejecución del código y un posible retardo en las transacciones. La dificultad de reducir la sobrecarga está en que no se conoce de antema las transacciones que estarán involucradas en unos conflictos. Para obtener dicho conocimiento se necesita un esquema para observar el sistema.

Se asume que cada transacción Ti se ejecuta en dos o tres fases diferentes durante su tiempo de vida, dependiendo de sí es una transacción de solo lectura o una actualización. Las fases son en orden que sigue:

1.−***Fase de lectura.***− Durante esta fase tiene lugar la ejecución dela transacción Ti. Todas las operaciones escribir se realizan sobre las variables locales temporales sin actualizar la base de datos actual.

2.***Fase de validación.***−La transacción Ti realiza una prueba de validación para determinar si puede copiar a la base de datos las variables locales temporales que tienen como resultado de las operaciones escribir sin causar una violación de la secuencialidad.

3.***−Fase de escritura.***− Si la transacción Ti tienen éxito en la validación entonces las actualizaciones reales se aplican a la base de datos. En otro caso se retrocede.

Protocolos basados en cerrojos

- El aislamiento en las transacciones viene impuesto por el acceso a los datos.
- Una forma de garantizar la serialización es garantizar el acceso exclusivo a los datos.
- Definiendo regiones críticas específicas de un dato.
- El método mas utilizado es manejar el acceso a los datos a través de cerrojos.

Protocolo de cerrojo en dos fases

- ***Asegura planes de ejecución seriables***

- ***Protocolo en dos fases***

Fase 1: Agrupamiento

◊ la transacción puede obtener cerrojos
◊ la transacción no puede liberar cerrojos

Fase 2: Reducción

◊ la transacción puede liberar recursos
◊ la transacción no puede obtener cerrojos

- ***Se puede probar que las transacciones son seriables en el orden que apuntan los cerrojos (donde se adquiere el cerrojo final)***

- ***Este protocolo no evita los bloqueos***

PROTOCOLOS BASADOS EN BLOQUEO.

Una forma de asegurar la secuencialidad es exigir que el acceso a los elementos de datos se haga en exclusión mutua, es decir, mientras una transacción accede a un elemento de datos, ninguna otra transacción puede modificar dicho elemento. El método mas habitual que se usa para implementar este requisito es permitir que una transacción acceda a un elemento de datos sólo si posee actualmente un bloqueo sobre dicho elemento.

Existen varios protocolos basados en marcas de tiempo, entre los que destacan:

WAIT−DIE que fuerza a una transacción a esperar en caso de que entre en conflicto con otra transacción cuya marca de tiempo sea más reciente, o a morir (abortar y reiniciar) si la transacción que se está ejecutando es más antigua.

WOUND−WAIT, que permite a una transacción matar a otra que posea una marca de tiempo más reciente, o que fuerza a la transacción peticionaria a

Esperar

Estos son todos de dos fases:

- 2PL centralizado
- 2PL con copia primaria
- 2PL distribuido
- bloque mayoritario.

2PL centralizado.

Se caracteriza por:

Hay un único planificador, o gestor de bloqueos (lock manager), para la totalidad del SGBD Distribuido que pueden garantizar (grant) y liberar (release) bloqueos.

2PL centralizado.

o Funcionamiento:

- El coordinador de transacciones local divide la transacción en subtransacciones, usando el catálogo global del sistema. Si la transacción implica actualizar un dato que está replicado, el coordinador solicita un bloqueo de escritura de todas las copias antes de actualizar cada copia y liberar los bloqueos. El coordinador puede elegir cualquiera de las copias de un dato replicado para lectura.
- El gestor de transacciones local, implicado en la transacción global, solicita y libera los bloqueos que mantiene el gestor centralizado de bloqueos usando las reglas usuales para el bloqueo en dos fases.
- El gestor centralizado de bloqueos comprueba que las peticiones de bloqueo sobre un dato sean compatibles, de manera que el gestor de bloqueos envía un mensaje de vuelta al nodo que originó la petición reconociendo que el bloqueo ha sido concedido. En caso contrario, coloca la petición en una cola hasta que el bloqueo pueda ser realizado.

2PL con copia primaria.

Se caracteriza por:

Cada gestor de bloqueos local es responsable de un conjunto de datos, de manera que se elige una copia como **copia primari**a; el resto de copia se llaman **copias esclava**s. La elección del nodo primario es flexible, y el nodo elegido no contiene necesariamente la copia primaria de ese dato.

2PL distribuido

Se caracteriza por:

Se distribuye un gestor de bloqueo en cada nodo. Cada uno es responsable de la gestión de bloqueos de los datos que contiene en ese nodo. El 2PL distribuido implementa una protocolo de control de replicas Read−One−Write−All.

Cualquier copia de un dato replicado puede ser usada para operaciones de lectura, pero todas las copias deben ser bloqueadas para escritura antes que se puedan modificar.

Bloqueo mayoritario.

o Se caracteriza por:

Se mantiene un gestor de bloqueos en cada nodo para gestionar los bloqueos de ese nodo. Cuando se ejecuta una transacción que trabaja con un dato que esta replicado, debe enviar una petición de bloqueo a más de la mitad de los nodos donde esta el dato.

Protocolo de recuperación. Fallo en el coordinador:

- **Estado INICIAL:** La recuperación en este caso consiste en iniciar elprocedimiento de ejecución (commit).
- **Estado de ESPERA:** El coordinador no ha recibido todas las respuestas(ninguna de abortar). La recuperación consiste en reiniciar el procedimiento de ejecución (commit).
- **Estado DECIDIDO:** El coordinador ha dado instrucciones para abortar o ejecutar globalmente la transacción. Al reiniciar, si el coordinador ha recibido todos los reconocimientos, completa la transacción con éxito, en caso contrario, tiene que iniciar el protocolo de terminación.

Protocolo de recuperación. Fallo en un participante:

- **Estado INICIAL:** El participante no ha votado todavía sobre la transacción,de manera que cuando se recupera puede abortar.
- **Estado PREPARADO:** El participante ha enviado su voto al coordinador. En este caso, la recuperación se

hace mediante el protocolo de terminación discutido anteriormente.

- **Fallo en los estados ABORTADO/EJECUTADO:** El participante ha completado la transacción. Por consiguiente, al reiniciar, ninguna otraacción será necesaria.

Este protocolo debe intentar que todos los participantes realicen las mismas

acciones (estado consistente).

Protocolo de elección:

El protocolo es ejecutado cuando los participantes detectan que el coordinador ha fallado. Una asunción que hace este protocolo es que los nodos tienen preestablecido un orden.

Funcionamiento: Cada nodo envía su numero de orden al resto de nodos mayores que el. Si este recibe un mensaje de un nodo menor que el deja de enviar mensajes.

3.4.3 DEADLOCK

El deadlock es una condición que ningún sistema o conjunto de procesos quisiera exhibir, ya que consiste en que se presentan al mismo tiempo cuatro condiciones necesarias: La condición de no apropiación, la condición de espera circular, la condición de exclusión mutua y la condición de ocupar y esperar un recurso. Ante esto, si el deadlock involucra a todos los procesos del sistema, el sistema ya no podrá hacer algo productivo. Si el deadlock involucra algunos

procesos, éstos quedarán congelados para siempre.

Causas del Dead Lock.

Los puntos muertos ocurren cuando dos o mas transacciones solicitan recursos en forma incremental y se bloquean mutuamente impidiéndose una a otra la conclusión. Las transacciones y recursos forman un ciclo.

Hasta es posible que un punto muerto se cree cuando se realiza al acceso a un solo objeto mediante dos transacciones y se aceptan reclamaciones increméntales. Este tipo de punto muerto tiene una mayor probabilidad de ocurrir si los objetos que se están asegurando son grandes cuando la unidad de aseguramiento es un archivo la primera reclamación puede emitirse para lograr acceso a un registro y la reclamación incremental emitirse a fin de actualizar otro registro de archivo.

Puntos muertos debidos a recursos compartidos del sistema, los puntos muertos también pueden provocarse debido a la competencia por objetos que no estén identificados en forma especifica, pero que sean miembros de una clase compartida de recursos. Un bloqueo temporal ocurre cuando los limites de un recurso se alcanza.

Condiciones de punto muerto, la posibilidad de puntos muertos existen 4 condiciones:

- **Seguros:** La interferencia de acceso se resuelve posesionando y reputando los seguros.
- **Bloqueo:** Un propietario de un objeto esta bloqueado cuando solicita un objeto asegurado.
- **Garantía de Conclusión:** Los objetos no pueden quitarse de sus propietarios.
- **Circulación:** Existe una secuencia circular de solicitud como se muestra en el.

En el área de la informática, el problema del deadlock ha provocado y

producido una serie de estudios y técnicas muy útiles, ya que éste puede surgir en una sola máquina o como consecuencia de compartir recursos en una red.

En el área de las bases de datos y sistemas distribuidos han surgido técnicas como el 'two phase locking' y el 'two phase commit' que van más allá de este trabajo. Sin embargo, el interés principal sobre este problema se centra en generar técnicas para detectar, prevenir o corregir el deadlock.

Las técnicas para prevenir el deadlock consisten en proveer mecanismos para evitar que se presente una o varias de las cuatro condiciones necesarias del deadlock. Algunas de ellas son:

- Asignar recursos en orden lineal: Esto significa que todos los recursos están etiquetados con un valor diferente y los procesos solo pueden hacer peticiones de recursos 'hacia adelante'. Esto es, que si un proceso tiene el recurso con etiqueta '5' no puede pedir recursos cuya etiqueta sea menor que '5'. Con esto se evita la condición de ocupar y esperar un recurso.
- Asignar todo o nada: Este mecanismo consiste en que el proceso pida todos los recursos que va a necesitar de una vez y el sistema se los da solamente si puede dárselos todos, si no, no le da nada y lo bloquea.
- Algoritmo del banquero: Este algoritmo usa una tabla de recursos para saber cuántos recursos tiene de todo tipo. También requiere que los procesos informen del máximo de recursos que va a usar de cada tipo. Cuando un proceso pide un recurso, el algoritmo verifica si asignándole ese recurso todavía le quedan otros del mismo tipo para que alguno de los procesos en el sistema todavía se le pueda dar hasta su máximo. Si la respuesta es afirmativa, el sistema se dice que está en 'estado seguro' y se otorga el recurso. Si la respuesta es negativa, se dice que el sistema está en estado inseguro y se hace esperar a ese proceso.

Para detectar un deadlock, se puede usar el mismo algoritmo del banquero, que aunque no dice que hay un deadlock, sí dice cuándo se está en estado inseguro que es la antesala del deadlock. Sin embargo, para detectar el deadlock se pueden usar las 'gráficas de recursos'. En ellas se pueden usar cuadrados para indicar procesos y círculos para los recursos, y flechas para indicar si un recurso ya está asignado a un proceso o si un proceso está esperando un recurso. El deadlock es detectado cuando se puede hacer un viaje de ida y vuelta desde un proceso o recurso. Por ejemplo, suponga los siguientes eventos:

evento 1: Proceso A pide recurso 1 y se le asigna.

evento 2: Proceso A termina su time slice.

evento 3: Proceso B pide recurso 2 y se le asigna.

evento 4: Proceso B termina su time slice.

evento 5: Proceso C pide recurso 3 y se le asigna.

evento 6: Proceso C pide recurso 1 y como lo está ocupando el proceso A, espera.

evento 7: Proceso B pide recurso 3 y se bloquea porque lo ocupa el proceso C.

evento 8: Proceso A pide recurso 2 y se bloquea porque lo ocupa el proceso B.

En la figura 5.1 se observa como el 'resource graph' fue evolucionando hasta que se presentó el deadlock, el cual significa que se puede viajar por las flechas desde un proceso o recurso hasta regresar al punto de partida. En el deadlock están involucrados los procesos A,B y C.

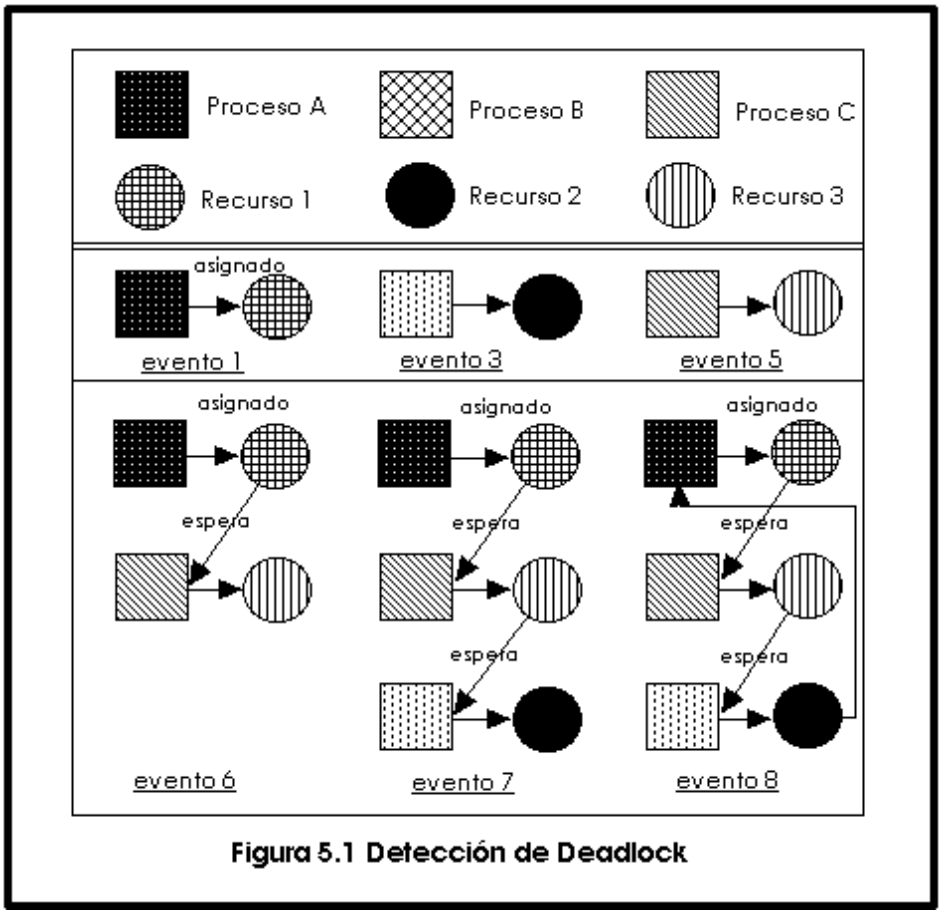


Figura 5.1 Detección de Deadlock

Una vez que un deadlock se detecta, es obvio que el sistema está en problemas y lo único que resta por hacer es una de dos cosas: tener algún mecanismo de suspensión o reanudación [Deitel93] que permita copiar todo el contexto de un proceso incluyendo valores de memoria y aspecto de los periféricos que esté usando para reanudarlo otro día, o simplemente eliminar un proceso o arrebatarle el recurso, causando para ese proceso la pérdida de datos y tiempo.

Causas de Punto Muerto.

Los puntos muertos ocurren cuando dos o más transacciones solicitan recursos en forma incremental y se bloquean mutuamente impidiéndose una a otra la conclusión. Las transacciones y recursos forman un ciclo.

Hasta es posible que un punto muerto se cree cuando se realiza al acceso a un solo objeto mediante dos transacciones y se aceptan reclamaciones increméntales.

Este tipo de punto muerto tiene una mayor probabilidad de ocurrir si los objetos que se están asegurando son grandes cuando la unidad de aseguramiento es un archivo la primera reclamación puede emitirse para lograr acceso a un registro y la reclamación incremental emitirse a fin de actualizar otro registro de archivo.

Puntos muertos debidos a recursos compartidos del sistema, los puntos muertos también pueden provocarse debido a la competencia por objetos que no estén identificados en forma especifica, pero que sean miembros de una clase compartida de recursos. Un bloqueo temporal ocurre cuando los limites de un recurso se alcanzan.

Las transacciones que se comunican a fin de procesar conjuntamente datos de proceso también están sujetas a puntos muertos, un ejemplo de transacción en cooperación de este tipo ocurre cuando los datos son trasladados entre archivos y buffers por una transacción y procesados simultáneamente por otras. Una

transacción puede generar datos que se escriban en los archivos con una gran velocidad y apoderarse de todos los buffers. Ahora una transacción de análisis de datos no puede proceder debido a que la transacción que los trasladando puede entregar datos adicionales debido a escasez de buffers ya que la transacción de análisis de los datos no libera sus buffers hasta que concluya, de nuevo, se presenta una situación clásica de punto muerto, los mecanismos para manejar los puntos muertos deben, incluir, todos los objetos asegurables, esto podría incluir unidades de cinta, impresora y otros dispositivos, buffers y áreas de memoria asignados por el sistema operativo, archivos de bloques y registros controlados por el sistema de archivos y de la base de datos.

Condiciones de punto muerto, la posibilidad de puntos muertos existen cuatro condiciones [coffman]:

1.− Seguros.− La interferencia de acceso se resuelve posesionando y reputando los seguros.

2.− Bloqueo.− Un propietario de un objeto esta bloqueado cuando solicita un objeto asegurado.

3.− Garantía de conclusión.− Los objetos no pueden quitarse de sus propietarios.

4. − Circularidad.− Existe una secuencia circular de solicitud como se muestra en él.

Deadlocks distribuidos

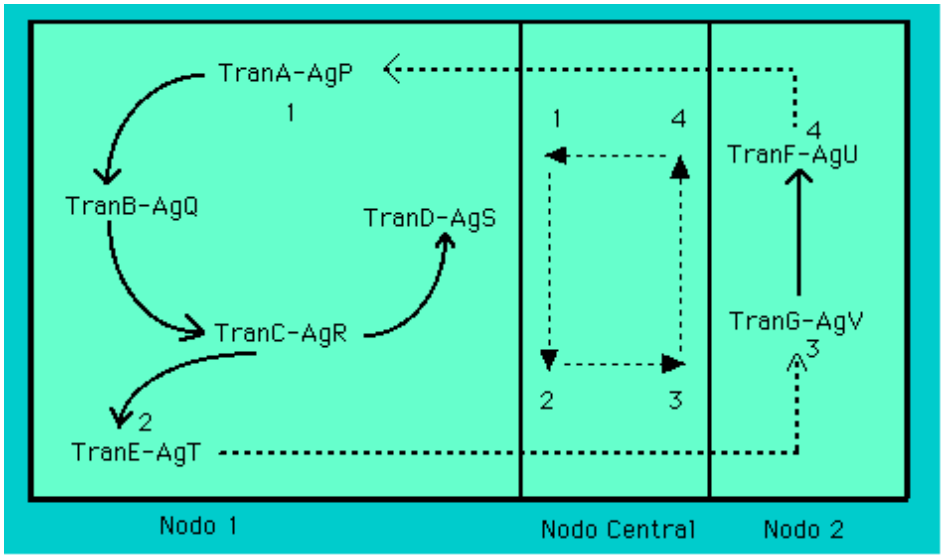
En la sección de detección de deadlocks vimos cómo detectar un deadlock en un solo nodo cuyos procesos compiten por recursos. En un ambiente distribuido la detección y recuperación es más compleja porque varios nodos están involucrados y se requiere el intercambio de información entre los manejadores de la transacción distribuida. En general se postulan tres alternativas para manejar los deadlocks

- **Detección de deadlocks usando un control jeráquico centralizado.**
- **Detección distribuida del deadlock.**
- **Prevención del deadlock.**

Como se puede apreciar, los métodos 1 y 2 son correctivos que necesariamente conllevan la pérdida de tiempo porque se tiene que abortar alguna transacción, el método 3 evita la aparición del deadlock y teóricamente sera preferible implantarlo.

Detección de deadlocks usando un control jerárquico centralizado

Con este método se requiere que cada nodo participante cuente con un detector de deadlocks que le enviarán al detector central la información pertinente de posibles deadlocks globales. Para entender esto, consideremos la siguiente figura de un deadlock global.



Cada detector local lo que tiene que hacer es reportar al nodo central qué transacciones están haciendo esperar a qué agentes distribuidos y detectar deadlocks locales. De esta manera, tenemos que el el detector local del nodo 1 le envía al nodo central mensajes diciendo que la transacción A está haciendo esperar al agente P (marcado con un número 1) y que la transacción E está haciendo esperar al agente T y para fines de detección el nodo central piensa que TranA−AgP está esperando un recurso bloqueado por TranE−AgT (flecha 1 a 2), por otro lado recibe del nodo 2 que TranF−AgU espera a TranA−AgP (flecha 4 a 1), y que TranG−AgV espera por TranF−AgU (flecha 3 a 4) del nodo 1 recibe que TranE−AgT espera por un recurso bloqueado por el nodo 2 por TranG−AgV (flecha 2 a 3) formando ya un ciclo que determina el deadlock.

Una vez detectado el deadlock, lo que se debe hacer es abortar alguna transacción en base a algún parámetro deseado, por ejemplo, evitar la pérdida del tiempo (abortar transacciones jóvenes), liberar el mayor número de recursos, etc.

Detección distribuida del deadlock

En este método se necesita que cada detector en cada nodo realice 4 pasos iterativos que se describen adelante. Ahora será necesario que exista un nodo llamado EXTERNO el cual recibe información de los nodos participantes. Los cuatro pasos son:

- Construir una gráfica de recursos con información de transacciones locales
- Para cada mensaje recibido de transacciones globales modificar la gráfica local de recursos añadiendo las transacciones externas. Si el mensaje viene del nodo EXTERNO, crear una flecha hacia la siguiente transacción contenida en el mensaje.
- Encontrar los ciclos que no involucren al nodo EXTERNO, si tales existen, entonces se le puede enviar un mensaje al nodo EXTERNO para que decida que transacciones abortar.
- Encontrar ciclos que involucren al nodo EXTERNO, estos ciclos denotan posibles deadlocks y pueden ser transmitidos al nodo EXTERNO para que a su vez los haga llegar a los otros nodos.

Ejemplo: Supóngase que ya existe un deadlock donde en el nodo 1 (N1) la transacción1(T1) espera a T2, T2 espera en N2 a T3, finalmente T3 espera a T1.

T1 ----------> T2	Nodo 1
T2 ----------> T3	Nodo1 a Nodo 2
T3 ----------> T1	Nodo 2 a Nodo 1

En el paso 1 el Nodo 1 construye su gráfica T1−−−> T2 y como T2 está esperando algo del sistema distribuido le envía esta información al nodo externo. El nodo externo le envía esta información al Nodo 2 el cual añade la transacción 2 a su control local y crea su gráfica T2−−>T3 y le informa al nodo externo que T3−−−>T1 en otro ciclo. El nodo externo le pasa este mensaje al Nodo 1 el cual entonces crea su gráfica T1−−>T2−−>T3−−>T1 lo cual es un deadlock.

3.4.3 TÉCNICAS PARA PREVENIRLO

Los métodos de corregir un deadlock una vez que ya se presentó éste conllevan la pérdida de tiempo y no de información, ya que las transacciones puedes abortarse de manera segura debido a las facilidades de las bitácoras, el protocolo de bloqueo de dos fases y el protocolo commit de dos fases.

El método de prevención del deadlock también hace necesario abortar transacciones y siempre serán las transacciones más nuevas o jóvenes las abortadas para evitar los deadlocks.

Existen dos métodos: el método apropiativo y el no apropiativo. Ambos se basan en la hora en que las transacciones son creadas, podemos hablar entonces de transacciones viejas y jóvenes.

En el método no apropiativo, si una transacción Ta pide un recurso bloqueado por Tb, se permite que Ta espere solamente si Ta es más vieja que Tb. Ta es reinicializada si es más joven que Tb, conservando siempre su antigüedad. Lo que se consigue con el método no apropiativo es que ninguna transacción joven esperará por las transacciones viejas, eliminando la condición de espera circular distribuida.

En el método apropiativo la regla es la inversa: Si Ta pide un bloqueo sobre un recurso cuyo dueño es Tb, se le permite esperar si Ta es más joven que Tb. Si Ta es más vieja, no espera sino que Tb es abortada por ser más joven y luego se reinicializa conservando su antigüedad.

Tambien podemos anular alguna de las 4 condiciones necesarias para que se produzca un deadlock:

- No puede ser anulada porque existen recursos que deben ser usados en modalidad exclusiva.
- La alternativa sería hacer que todos los procesos solicitaran todos los recursos que habrán de utilizar antes de utilizarlos al momento de su ejecución lo cual sería muy ineficiente.
- Para anular esta condición cuando un proceso solicita un recurso y este es negado el proceso deberá liberar sus recursos y solicitarlos nuevamente con los recursos adicionales. El problema es que hay recursos que no pueden ser interrumpidos.
- Espera Circular: esta estrategia consiste en que el sistema operativo numere en forma exclusiva los recursos y obligue a los procesos a solicitar recursos en forma ascendente. El problema de esto es que quita posibilidades a la aplicación.

3.4.3 b TÉCNICAS PARA PREVENIRLO

Si se tiene cuidado en la forma de asignar los recursos se pueden evitar situaciones de Deadlock. Supongamos un ambiente en el que todos los procesos declaren a priori la cantidad máxima de recursos que habá de usar.Estado Seguro: un estado es seguro si se pueden asignar recursos a cada proceso (hasta su máximo) en algún orden sin que se genere Deadlock. El estado es seguro si existe un ordenamiento de un conjunto de procesos {P1...Pn} tal que para cada Pi los recursos que Pi podrá utilizar pueden ser otorgados por los recursos disponibles mas los recursos utilizados por los procesos Pj,j<i. Si los recursos solicitados por Pi no pueden ser otorgados, Pi espera a que todos los procesos Pj hayan terminado.

Como evitar los puntos muertos.

Para evitar puntos muertos puede simplificar muchas de elecciones alternativas. Los esquemas para evitar

estos casos imponen a los usuarios restricciones que pueden resultar difíciles de aceptar. Existen cuatro enfoques para puntos muertos:

1.− **Reparación posterior:** No utilizar seguros y arreglar después las fallas por inconsistencia.

2.− **No bloquear:** Solamente afectar a quienes efectuaron solicitudes que provocaron reclamaciones en conflicto.

3.**− Asignación Previa:** Si existe algún conflicto, quitar los objetos a sus propietarios.

4.− **Aseguramiento de dos fases:** Se realizan primero todas las reclamaciones y si ninguna esta bloqueada se inician todas las modificaciones.

Reparación Posterior.

El primer enfoque, es reparar posteriormente los problemas debido a no asegurar, puede responder a un valido en sistemas experimentales y educativos, pero resulta inaceptable en la mayoría de las aplicaciones comerciales.

No Bloquear.

El segundo enfoque asigna la responsabilidad a la transacción, el sistema proporcionara un aviso de interferencia potencial al negar la solicitud de acceso explosivo.

Asignación Previa.

La asignación previa de reclamaciones otorgados a las transacciones reúne de una capacidad de volver a enrollar . La transacción, cuando se le notifica que no puede continuar, tiene que restaurar la base de datos y colocarse así misma en la línea de espera para un nuevo turno o el sistema tiene que eliminar la transacción, restaurar la base de datos y volver a iniciar de nuevo la transacción.

Secuencia Previa.

Consiste en evitar la circularidad en la secuencia de solicitud, en este caso existen 3 enfoques, vigilancia de la existencia para evitar la circularidad y aseguramiento de 2 fases, a fin de evitar puntos muertos puede vigilarse el patrón de solicitud para todas las transacciones.

Aseguramiento de Dos Fases.

Un enfoque simple para evitar la circularidad consiste en hacer que se reclamen previo todos los objetos antes de otorgar ningún seguro, el reclamar los recursos antes de prometer otorgar el acceso a ellos significa que posiblemente una transacción no sea capaz de concluir la fase de reclamación previa.

El problema de 2 fases es que la reclamación previa puede llegar a tener que reclamar mas y mayores objetos de los que en realidad se necesita, si un calculo sobre los datos determina que objetos se necesitan después, es posible que se reclame en forma previa todo un archivo en vez de un registro.

Recursos Reservados.

Los puntos muertos provocados por la competencia de recursos clasificados, algunas veces se disminuye al no permitir que ninguna nueva transacción se inicie cuando la utilización llega al nivel. Esta técnica no asegura evitar los puntos muertos a menos que la reserva se conserve de un tamaño tan grande que resulte poco

practico, lo suficiente para permitir que todas las transacciones activas se concluyan.

Métodos de recuperación

Se usan dos métodos de recuperación de base de datos: recuperación en avance o recuperación en retroceso (roll−forward o roll−back).El método a utilizar depende del tipo y la extensión de los errores.

1) Recuperación en avance

Con este método se consigue la restauración mediante la copia de respaldo de la base para recuperar la porción principal de los datos. Después se reaplican los registros post−imagen del registro a la copia de respaldo para incorporar las actualizaciones efectuadas desde que se hizo la copia de respaldo. Los registros post−imagen del registro log, más que las transacciones, se reaplican ala copia de respaldo, ya que estas imágenes son datos procesados ,listos para rescribirse en la base.

- **Recuperación en retroceso**

Esta recuperación puede nulificar el efecto de una sola transacción que hizo cambios en la base pero abortó antes del término. (Recuerde que para mantener la integridad de los datos, una transacción debe ejecutarse en su totalidad o no ejecutarse). La recuperación en retroceso se consigue llamando al registro ante−imagen del archivo log y reinstalándolo en la base para nulificar el efecto de transacción errónea.

Operación de registro

La mayoría de los DBMS proporcionan un elemento de rastreo para registrar lo sucedido en cada transacción actualizada por la base de datos .

El registro, es un prerrequisito para la recuperación de la base de datos; restaura un archivo a su estado anterior cuando ocurre alguna falla en una transacción. El manejador del registro (log manager), es una componente de la DBMS que efectúa el registro escribiendo cualquiera de los dos tipos de registro siguiente en un archivo de búsqueda:

- ante−imagen: se refiere a los bloques antiguos de datos originales en la base que se guardaron antes de la actualización de una transacción. Si ocurre un error, esta copia se puede usar para cancelar el efecto de la transacción.

2) Post−imagen: es el nombre que recibe un bloque procesado por una

transacción listo para ser reescrito en la base.

3.5 ETIQUETAS DE TIEMPO

Para evitar interferencia debida a la concurrencia se usa el aseguramiento o el tiempo de estampado para seriar la ejecución de transacciones concurrentes, de cualquier modo, estas estrategias requieren de complicados protocolos de control los que pueden deteriorar el desempeño del sistema al generar trafico extra entre las distintas localidades.

Para establecer este ordenamiento, el administrador de transacciones le asigna a cada transacción T1 una estampa de tiempo única Ts (Ti) cuando esta inicia.

A diferencia de los algoritmos basados en candados, los algoritmos basados en estampas de tiempo no pretenden mantener la seriabilidad por exclusión mutua. En lugar de eso, ellos seleccionan un orden de

serialización a priori y ejecutan las transacciones de acuerdo a ellas. Para establecer este ordenamiento, el administrador de transacciones le asigna a cada transacción Ti una estampa de tiempo única ts(Ti) cuando ésta inicia.

Cuando esta inicia una estampa de tiempo es un identificador simple que sirve para identificar cada transacción de manera única. Otra propiedad de las estampas de tiempo es la monoticidad, esto es, dos estampas de tiempo generadas por el mismo administrador de transacciones deben ser monotonicamente crecientes. Así, las estampas de tiempo son valores derivados de un dominio totalmente ordenado.

La acción de rechazar una operación , significa que la transacción que la enviò necesita reiniciarse para obtener la estampa de tiempo mas reciente del dato, e intentar nuevamente la operación sobre el dato.

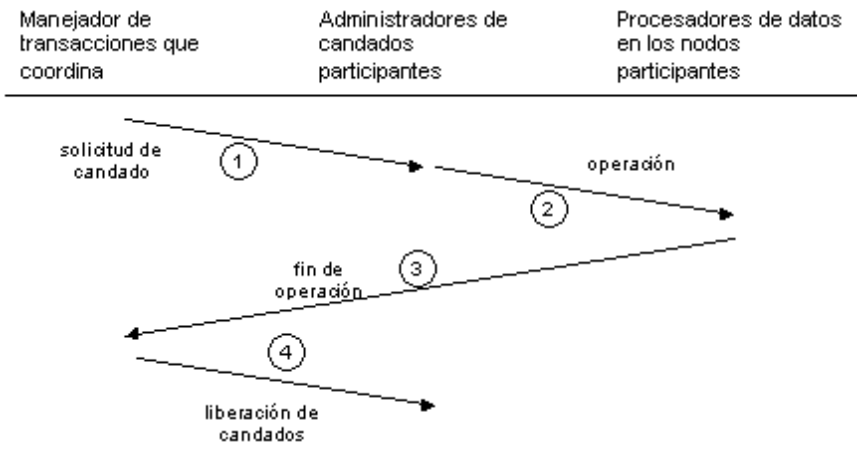


Figura. Comunicación en candados de dos fases distribuidos.

Por lo tanto, es preferible que cada nodo asigne de manera autónoma las estampas de tiempos basándose en un contador local. Para obtener la unicidad, cada nodo le agrega al contador su propio identificador. Así, la estampa de tiempo es un par de la forma

<contador local, identificador de nodo>

Note que el identificador de nodo se agrega en la posición menos significativa, de manera que, éste sirve solo en el caso en que dos nodos diferentes le asignen el mismo contador local a dos transacciones diferentes.

El administrador de transacciones asigna también una estampa de tiempo a todas las operaciones solicitadas por una transacción. Más aún, a cada elemento de datos *x* se le asigna una *estampa de tiempo de escritura*, *wts*(*x*), y una *estampa de tiempo de lectura*, *rts*(*x*); sus valores indican la estampa de tiempo más grande para cualquier lectura y escritura de *x*, respectivamente.

MARCAS DE TIEMPO (TIMESTAMPING)

Es una técnica que podría encuadrarse dentro de las optimistas.Se asigna a cada transacción un identificador único, su marca de tiempo (tiempo de inicio).No hay bloqueos. Se retardan las actualizaciones hasta el final de la transacción.

− Si una transacción quiere actualizar o consultar un dato que ha sido actualizado por una transacción de fecha posterior (más reciente), entonces se deshace y se vuelve a lanzar.

Es decir si se intenta usar un dato que se ha modificado después de que la transacción se inicie, no se puede continuar porque el valor inicial que tenía el dato para la transacción ha cambiado.

Este mecanismo asegura la seriabilidad al ordenar las transacciones evitando solapamientos, basandose en el momento de inicio de cada transacción.

Además, no se pueden producir interbloqueos al no usar mecanismos que impliquen la espera de las transacciones.

El administrador de transacciones asigna también una estampa de tiempo a todas las operaciones solicitadas por una transacción. Más aún, a cada elemento de datos x se le asigna una estampa de tiempo de escritura, wts(x), y una estampa de tiempo de lectura, rts(x); sus valores indican la estampa de tiempo más grande para cualquier lectura y escritura de x, respectivamente.

El ordenamiento de estampas de tiempo (TO) se realiza mediante la siguiente regla:

Regla TO: dadas dos operaciones en conflicto, *Oij* y *Okl*, perteneciendo a las transacciones *Ti* y *Tk*, respectivamente, *Oij* es ejecutada antes de *Okl*, si y solamente si, *ts*(*Ti*) < *ts*(*Tk*). En este caso *Ti* se dice ser un transacción *más vieja* y *Tk* se dice ser una transacción *más joven*.

Dado este orden, un conflicto entre operaciones se puede resolver de la siguiente forma:

	for *Wi*(*x*) ***do begin***
for *Ri*(*x*) ***do begin***	***if*** *ts*(*Ti*) < *rts*(*x*) ***and***
if *ts*(*Ti*) < *wts*(*x*) ***then***	*ts*(*Ti*) < *wts*(*x*) ***then***
reject Ri(*x*)	*reject Wi*(*x*)
else	***else***
accept Ri(*x*)	*accept Wi*(*x*)
rts(*x*) ð *ts*(*Ti*)	*wts*(*x*) ð *ts*(*Ti*)
end	***end***

La acción de rechazar una operación, significa que la transacción que la envió necesita reiniciarse para obtener la estampa de tiempo más reciente del dato e intentar hacer nuevamente la operación sobre el dato.

Ordenamiento conservador por estampas de tiempo

El ordenamiento básico por estampas de tiempo trata de ejecutar una operación tan pronto como se recibe una operación. Así, la ejecución de las operaciones es progresiva pero pueden presentar muchos reinicios de transacciones.

El ordenamiento conservador de estampas de tiempo retrasa cada operación hasta que exista la seguridad de que no será reiniciada.

La forma de asegurar lo anterior es sabiendo que ninguna otra operación con una estampa de tiempo menor puede llegar al despachador. Un problema que se puede presentar al retrasar las operaciones es que esto puede inducir la creación de interbloqueos (deadlocks).

Ordenamiento por estampas de tiempo múltiples

Para prevenir la formación de interbloqueos se puede seguir la estrategia siguiente. Al hacer una operación de escritura, no se modifican los valores actuales sino se crean nuevos valores. Así, puede haber copias múltiples de un dato. Para crear copias únicas se siguen las siguientes estrategias de acuerdo al tipo de operación de que se trate:

- Una operación de lectura *Ri*(*x*) se traduce a una operación de lectura de *x* de una sola versión encontrando la versión de *x*, digamos *xv*, tal que, *ts*(*xv*) es la estampa de tiempo más grande que tiene un valor menor a *ts*(*Ti*).
- Una operación de escritura *Wi*(*x*) se traduce en una sola version, *Wi*(*xw*), y es aceptada si el despachador no ha procesado cualquier lectura *Rj*(*xr*), tal que,

ts(*Ti*) < *ts*(*xr*) < *ts*(*Tj*)

Control de concurrencia optimista

Los algoritmos de control de concurrencia discutidos antes son por naturaleza pesimistas. En otras palabras, ellos asumen que los conflictos entre transacciones son muy frecuentes y no permiten el acceso a un dato si existe una transacción conflictiva que accesa el mismo dato.

Así, la ejecución de cualquier operación de una transacción sigue la secuencia de fases: validación (V), lectura (R), cómputo (C) y escritura (W) (ver Figura 6.6a). Los algoritmos optimistas, por otra parte, retrasan la fase de validación justo antes de la fase de escritura (ver Figura 6.6b). De esta manera, una operación sometida a un despachador optimista nunca es retrasada.

Las operaciones de lectura, cómputo y escrita de cada transacción se procesan libremente sin actualizar la base de datos corriente. Cada transacción inicialmente hace sus cambios en copias locales de los datos.

La fase de validación consiste en verificar si esas actualizaciones conservan la consistencia de la base de datos. Si la respuesta es positiva, los cambios se hacen globales (escritos en la base de datos corriente). De otra manera, la transacción es abortada y tiene que reiniciar.

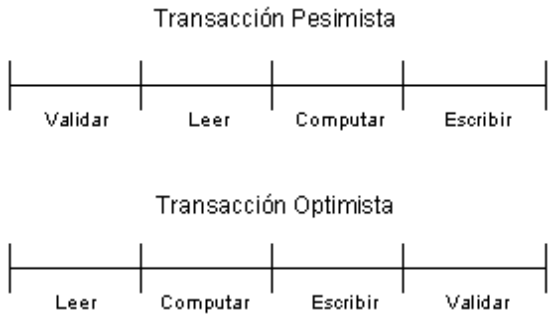


Figura 6.6. Fases de la ejecución de una transacción a) pesimista, b) optimista.

Los mecanismos optimistas para control de concurrencia fueron propuestos originalmente con el uso de estampas de tiempo. Sin embargo, en este tipo de mecanismos las estampas de tiempo se asocian únicamente con las transacciones, no con los datos.

Más aún, las estampas de tiempo no se asignan al inicio de una transacción sino justamente al inicio de su fase de validación. Esto se debe a que las estampas se requieren únicamente durante la fase de validación.

Una de las alternativas mas comunes para minimizar el ínter bloqueo consiste en asignar una secuencia especifica de acceso a los recursos para aquellos procesos que lo requieran en forma común.

Actualmente las políticas de los sistemas cuentan con técnicas para detectar el interbloqueo y abortar a uno o varios procesos involucrados; para darle agilidad al sistema, las formas de detección son:

−Control de longitud de la cola de proceso.

−Verificación de ciclos en las conexiones.

También se deben tomar precauciones para la seguridad de redes de usuario, como las siguientes:

−Validar no contraseñas repetidas.

−Eliminar claves de acceso a usuarios deshabilitados.

−Establecer sanción por desatención a estaciones desconectadas.

−Restringir procesos de alto riesgo a terminales de con mayor nivel de seguridad o vigilancia

−Implementar sistemas electrónicos de autentificación terminal.

−Políticas para denegar el acceso después de una cantidad determinada de intentos fallidos en un tiempo transcurrido.

Debe considerarse la posibilidad de controles alternos cuando el sistema maneje información o recursos altamente importantes para la organización.

Formas para autentificar la identidad del usuario:

−Algo que el usuario conoce. Password,contraseña,algoritmos de acceso.

−Algo que el usuario tiene.Tarjetas de acceso,bandas magneticas.

−Identificación de aspectos fisicos.Huellas digitales,examen de la retina,voz.

Control de concurrencia basado en timestamps:

En este método, cada transacción recibe una especie de boleto de antigüedad(timestamps) que les da cierta prioridad. La regla básica ahora es que las transacciones son ordenadas y ejecutadas de acuerdo con las timestamps ,de tal manera que si una transacción va a leer un dato x, solo es valido dicho dato si fue escrito por una transacción con un timestamp mas viejo.

El mecanismo básico de timestamps es como sigue:

- Cada transacción recibe su timestamp en su lugar de origen
- Cada operación de lectura o escritura está asociada al timestamp de la transacción que la solicita
- Para cada dato X, el timestamp más viejo de lectura TMVL y el timestamp de escritura más viejo TMVE son registrados.
- Para toda operación de lectura, si existe una operación OL(X) que actúa sobre el dato X se compara su timestamp con el TMVE. Si la operación de lectura actual tiene un timestamp más viejo que TMVE, la operación es abortada y la lectura recibe otro timestamp. Si el timestamp es más nuevo, lo cual indica que

el dato X fue escrito por una transacción más vieja, la operación es realizada y se modifica TMVL = más_viejo(TMVL,timestamp(OL(X)).

- Para toda operación de escritura OE(X) que actúa sobre el dato X, si su timestamp es más viejo TMVL o TMVE se aborta dicha escritura y la transacción es reinicializada con un nuevo timestamp. Si su timestamp es más nuevo, la operación se realiza y TMVE = timestamp(OE(X)).

Lo interesante del control de concurrencia por timestamps es que ya evita los deadlocks porque no permite la condición de espera circular, ya que una transacción, para cerrar el ciclo, necesita pedirle en algún momento un recurso a otra transacción más nueva. Otros aspectos interesantes es que proveen la serialización y únicamente necesitamos de la atomicidad, la cual puede conseguirse modificando los puntos 4 y 5 por los siguientes puntos 4,5 y 6:

4. Definamos como TS el timestamp de una operación de pre−escritura OPE(X) que actúa sobre el dato X. Si TS < TMVL o si TS < TMVE entonces OPE(X) no se hace y su transacción es reinicializada. Si no, OPE(X) se registra pero no se ejecuta.

5. Sea TS el timestamp de una operación de lectura OPL(X) sobre el dato X. Si TS < TMVE la operación es abortada. Si TS > TMVE entonces OPL(X) se ejecuta solamente si no existe ninguna OPE(X) registrada cuyo timestamp sea más viejo que TS. Si ocurre que TS(OPE) es ms vieja que TS, entonces la operación es registrada pero no ejecutada. Cuando las operaciones de escritura OPE(X) sean ejecutadas, inmediatamente se liberan las operaciones de lectura pendientes, logrando así su serialización.

6. Sea TS el timestamp de una operación de escritura OE(X) sobre el dato X. Si existe una pre−operación de escritura OPE(X), entonces OE(X) se almacena. Cuando las pre−operaciones de escritura OPE(X) sean ejecutadas, después se ejecutan las OE(X).

1. Leer artículo 100

2. Cambiar artículo 100

3. Escribir artículo 100

Usuario A

1. Leer artículo 200

2. Cambiar artículo 200

3. Escribir artículo 200

Usuario B

1. Leer artículo 100 para A

2. Leer artículo 200 para B

3. Cambiar artículo 100 para A

4. Escribir artículo 100 para A

5. Cambiar artículo 200 para B

6. Escribir artículo 200 para B

Orden de procesamiento de la CPU

GRAFO DE LA BASE DE

DATOS CON ESTRUCTURA DE ARBOL

A

C

B

F

D

E

I

H

G

J