Un prototipo de manejador de transacciones anidadas con restricciones de integridad *

A. Doucet[1], S. Gançarski[1], V. Leguizamo[2], C. León[2], M. Rukoz[2]

[1]: LIP6, Universite Pierre et Marie Curie. Case 169, 4,place Jussieu, 75252, Paris cedex 05. France

[2]: Universidad Central de Venezuela. Apdo. 47002, Los Chaguaramos,1041A, Caracas. Venezuela.

e-mail: {*Anne.Doucet,Stephane.Gancarski*}*@lip6.fr* {vleguiza,cleon,mrukoz}@strix.ciens.ucv.ve

Abstract

Este artículo presenta un prototipo de manejador de transacciones anidadas que integra la verificación de las restricciones de integridad al control de ejecución de una transacción. Este prototipo se desarrolló para evaluar un algoritmo de verificación de restricciones de integridad en sistemas que soportan transacciones anidadas. El algoritmo se basa en designar al ancestro común más joven de todas las sub-transacciones propensas a violar una restricción, como responsable de controlar su verificación. La evaluación muestra que realizar la verificación de restricciones lo más pronto posible durante la ejecución de una transacción, evita la ejecucón de un porcentaje importante de trabajo inútil al abandonar anticipadamente transacciones que en cualquier circunstancia serían abandonadas. Además, se muestra que un porcentaje significativo de transacciones son *salvadas* al lograr validar a pesar de haber violado restricciones de integridad durante su ejecución.

Palabras claves: Bases de Datos. Manejador de Transacciones. Transacciones anidadas. Restricciones de integridad.

1 Introducción.

Los sistemas para verificar restricciones de integridad son concebidos, en la mayoría de los casos, para el modelo simple y clásico de transacciones planas, existiendo muy pocas soluciones que consideren el contexto de transacciones anidadas[11, 7, 2]. Aún cuando parezca natural que las restricciones de integridad sean verificadas por cada sub-transacción de una transacción anidada, esta no es una solución eficaz, ya que el sistema podría enfrentarse a un problema de redundancia de verificación si muchas sub-transacciones *tocan* una misma restricción. Más aún, el sistema puede detectar al final de una sub-transacción una incoherencia que podriamos calificar de temporal en la medida que otra sub-transacción que toque la misma restricción y no haya todavía finalizado pueda corregir dicha incoherencia, ya que en las transacciones anidadas la propiedad de coherencia debe ser respetada unicamente por la transacción raíz [9].
En [3, 8] proponemos una solución para la verificación de restricciones de integridad en presencia de transacciones anidadas, en la cual se integra la capacidad de verificar restricciones de integridad al control de ejecución, sin interferencia. La transparencia es provista ya que los usuarios no tienen que agregar ningún código de control para la definición de restricciones o de transacciones. En este artículo se presenta un prototipo de manejador de transacciones anidadas desarrollado con el objeto de evaluar el algoritmo propuesto en [3, 8] para la verificación de restricciones de integridad. Este artículo está organizado de la siguiente manera. En la sección 2 se define el modelo de transacciones y se establecen las características del algoritmo para verificar restricciones de integridad en el contexto de transacciones anidadas. La sección 3 describe el prototipo del manejador de transacciones desarrollado. La sección 4 presenta los resultados obtenidos en la evaluación del algoritmo. Finalmente, en la sección 5 presentamos las conclusiones.

*Este trabajo fué parcialmente financiado por el CDCH-UCV y CONICIT, Venezuela y CNRS, Francia

2 Un modelo de transacciones anidadas

El modelo considerado en este trabajo es una extensión del modelo de transacciones anidadas propuesto por Moss [9]. Las transacciones anidadas (TA) permiten descomponer una transacción en una jerarquía de tareas llamadas sub-transacciones. Cada sub-transacción es ejecutada de forma aislada y constituye una unidad de recuperación después de una falla. Una TA puede contener una cantidad arbitraria de sub-transacciones, que a su vez pueden contener cualquier número de sub-transacciones, lo cual da origen a un *árbol de transacciones* de profundidad arbitraria, como el mostrado en la figura 1.

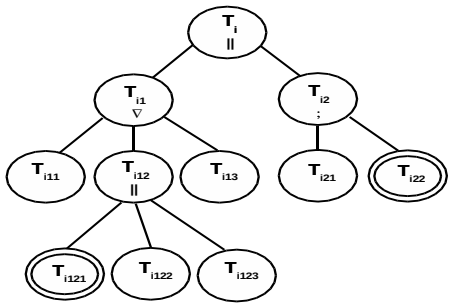

Figure 1: Ejemplo de una transacción anidada

2.1 Atomicidad y durabilidad

Debido al anidamiento entre las transacciones, las propiedades de atomicidad y durabilidad son reformuladas para las transacciones anidadas y se rigen por las siguientes reglas: (1) Una sub-transacción T_{ij} hija de T_i, comienza después y termina antes que T_i. (2) El abandono de una transacción implica el abandono de sus *descendientes*, pero no necesariamente el de sus *superiores*. (3) Cuando una transacción abandona, las actualizaciones que haya realizado son abandonadas. (4) Cuando una sub-transacción valida, sus actualizaciones son *heredadas* por su *madre*. (5) Cuando la *transacción raíz* valida todas las actualizaciones que ella haya *heredado* son integradas a la base de datos.
En este trabajo, consideramos la definición explícita de una transacción como *opcional* u *obligatoria*. El abandono de una sub-transacción *obligatoria* implica el abandono de su *madre*, mientras que el abandono de una sub-transacción *opcional* no obliga al abandono de su *madre*. En el árbol de transacciones de la figura 1 los nodos correspondientes a transacciones opcionales están representados con un doble óvalo. Así, T_{i121} y T_{i22} son transacciones opcionales, mientras que las restantes, son obligatorias.

2.2 Validación y abandono

Para el manejo de validaciones y abandonos de transacciones seguimos la propuesta de Moss [9]. Una transacción está inicialmente en estado de *ejecución* lo cual indica que ella existe pero aún no está lista para validar. Cuando la transacción termina todo el trabajo que deseaba hacer, entra en estado de *terminada*. Cuando todas las hijas de la transacción han terminado de forma aceptable (lo cual significa que todas las hijas que la madre requiere para validar ya han validado, mientras que sus restantes hijas pueden haber validado ó abandonado, pero ya no están en estado de *ejecución*) el manejador de transacciones mueve la transacción al estado de *validada*. Es de hacer notar que en este trabajo estamos obviando el protocolo de validación a dos fases [9] y en realidad el estado final de *validada* no hace permanentes los efectos de una transacción[1]. El estado *validada* indica que la transacción está lista o preparada para participar en un protocolo de validación y es ese el significado que damos al término en este documento.
Tan pronto como una sub-transacción *valide* la información concerniente a todos sus *inferiores* que hayan *validado* es heredada por su *madre*. Esto permite que cuando la *transacción raíz valide*, todas las actualizaciones

[1] De acuerdo al protocolo de validación a dos fases, la transacción debe pasar por los estados de *preparada* y luego *completada* para que sus actualizaciones sean permanentes.

realizadas por sub-transacciones que *validaron* y cuyos *ancestros* hayan *validado* sean hechas permanentes en la base de datos.

2.3 Modos de ejecución

Al momento de crear una TA, el usuario debe indicar explícitamente el modo de ejecución de las *hijas* de cada sub-transacción *madre*. Consideramos tres posibles modos de ejecución: **Ejecución Secuencial**, denotada por el operador ";" indica que las *hijas* deben ser ejecutadas una a la vez y de izquierda a derecha; **Ejecución Paralela**, denotada por el operador "||" indica que las *hijas* pueden ser ejecutadas simultáneamente y **Ejecución Alternativa**, denotada por el operador "$\bigtriangledown$" indica que las *hijas* deben ser ejecutadas una a la vez y sólo una de ellas puede validar. Este último modo de ejecución permite seleccionar uno entre varios procesos similares y descartar los otros. En la figura 1 se incluye el modo de ejecución de las *hijas* dentro del nodo que corresponda en el árbol a su *madre*. Así, el operador "||" en T_i indica que T_{i1} y T_{i2} pueden ser ejecutadas en paralelo, el operador "$\bigtriangledown$" en T_{i1} indica que sólo una alternativa entre T_{i11}, T_{i12} y T_{i13} debe ser validada y el operador ";" en T_{i2} indica que T_{i21} debe ser iniciada primero y una vez que ella finalice, es que puede ser iniciada T_{i22}.

2.4 Control de la concurrencia

En este trabajo seguimos el método de control de concurrencia de *bloqueo a dos fases* propuesto por Moss para las transacciones anidadas [9], el cual considera la herencia de bloqueos de las transacciones *hijas* hacia su *madre*. Al igual que en el trabajo de Moss, suponemos que sólo las *hojas* pueden acceder a los objetos de la base de datos. Esta consideración no le resta expresividad al modelo y permite simplificar los procesos de control de ejecución y de concurrencia de las transacciones anidadas.
El mecanismo de control de concurrencia asegura que si una transacción abandona sólo sus descendientes son afectados por dicho abandono, ya que los objetos usados por dicha transacción sólo han sido utilizados por las sub-transacciones dentro del sub-árbol del cual ella es raíz.

2.5 Manejo de restricciones de integridad

Para el manejo de retricciones de integridad seguimos el algoritmo propuesto en [3, 8], el cual está ubicado dentro del enfoque clásico de detección[2]. La verificación de restricciones de integridad se realiza bajo las siguientes hipótesis: (1) Las restricciones de integridad son declaradas de forma global a nivel del esquema de la BD. (2) Se realiza un análisis sintáctico de las restricciones y de las transacciones que permita reducir, en tiempo de compilación, el conjunto de restricciones a verificar, ya que se determina a priori el conjunto de restricciones susceptibles a ser violadas (*tocadas*) por una transacción[3]. En nuestro modelo de transacciones anidadas, es posible aplicar el principio de análisis sintáctico definido en [10] a cada transacción hoja para determinar el conjunto de restricciones que ella *toca*. El conjunto de restricciones *tocadas* por una transacción madre es la unión de las restricciones *tocadas* por sus hijas. (3) El chequeo de cada restricción se realiza a través de algoritmos generados automáticamente. Para optimizar el costo del proceso de verificación, los algoritmos son adaptados y optimizados de acuerdo a los diferentes tipos de restricciones. Estos algoritmos operan sobre un conjunto minimal de objetos, que es determinado en tiempo de ejecución y consiste únicamente de los objetos modificados por las transacciones que tocan la restricción y aquellos en la BD que sean relevantes para la restricción.
En este artículo, nuestra propuesta se centra en el proceso de verificación de restricciones durante el tiempo de ejecución de las transacciones anidadas. Suponemos que la especificación y análisis de la restricciones fué realizada durante la definición del esquema, que el análisis sintáctico fué realizado durante el tiempo de definición de las transacciones y que se realizó la generación automática de algoritmos de chequeo.

[2] El enfoque de detección consiste en evaluar las restricciones de integridad después que las actualizaciones sobre la BD son realizadas. En caso de violación, la transacción es abandonada y los cambios realizados a la BD, deben ser deshechos [6]. Los métodos de detección recurren al análisis de las transacciones y de las restricciones a fin de determinar el conjunto mínimo de restricciones que deben ser evaluadas luego de la ejecución de una transacción.

[3] El objetivo principal del análisis sintáctico es determinar el conjunto de objetos *involucrados* en una restricción C y el conjunto de objetos *tocados* por una transacción T. Cuando la intersección del análisis de T y del análisis de C es no vacía, decimos que la transacción T puede violar la restricción C, o lo que es lo mismo, que T *toca* C.

2.5.1 Principios del algoritmo para verificar restricciones

La idea central del algoritmo propuesto en [3, 8] es seleccionar para cada restricción, una transacción que será la responsable de su control: la transacción que corresponda en el árbol al ancestro común más joven de las hojas que tocan la restricción. Denotamos $SCA(C_i)$ a la transacción responsable de la restricción C_i. Dado que el $SCA(C_i)$ mantiene el control de ejecución de su sub-árbol, es posible asegurar que en caso de violación de C_i, no sólo serán abandonadas las transacciones que tocan C_i. También las transacciones que hayan visto sus resultados serán abandonadas en cascada. Por otra parte, y contrariamente a los sistemas que manipulan transacciones planas, las transacciones que no tengan relación con C_i continuan su ejecución. El mecanismo de control de concurrencia nos garantiza que la restricción C_i no es *tocada* por sub-transacciones fuera del sub-árbol del cual $SCA(C_i)$ es la raíz. En la figura 2 se representa el árbol correspondiente a una

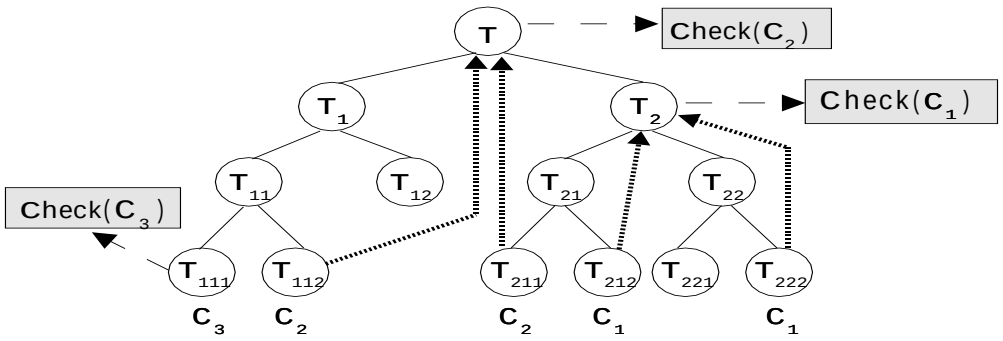

Figure 2: Restricciones *tocadas* por cada hoja de una TA

transacción anidada T. Debajo de cada transacción hoja T_j, se muestra el conjunto de restricciones que ella toca, es decir $C(T_j)$. Por ejemplo, $C(T_{211}) = \{C_2\}$. También podemos observar que $T(C_1) = \{T_{212}, T_{222}\}$, lo cual significa que la restricción C_1 es *tocada* por las transacciones T_{212} y T_{222}. T_2 es el ancestro común más joven de T_{212} y T_{222}, por lo tanto T_2 es el responsable del control de C_1. El procedimieno de verificación de cada restricción C_i se representa por un rectángulo asociado con la sub-transacción responsable de iniciarlo. Por ejemplo, el procedimiento de verificación de C_1 está asociado con T_2.La ejecución de $Check(C_i)$ es una transacción hija de $SCA(C_i)$. Cuando una transacción hoja T_j termina su ejecución (esto es, finaliza las acciones que debía llevar a cabo) envía el mensaje correspondiente (A^* ó V^*) a las transacciones responsables de mantener cada una de las restricciones que *toca*.
Cuando la transacción $SCA(C_i)$ ha recibido los avisos de terminación de todas las hojas que tocan C_i, inicia el proceso $Check(C_i)$. Si la restricción es satisfecha $SCA(C_i)$ no realiza acciones adicionales. Por el contrario, si la restricción es violada, el $SCA(C_i)$ enviará un mensaje $(R^*, SCA(C_i), C_i)$ por cada hoja T_j en $T(C_i)$ cuyos efectos hayan sido considerados en la verificación. Este mensaje R^* es enviado a través de la hija de $SCA(C_i)$ que sea ancestro de T_j. El significado de este mensaje es: *deshacer T_j y cualquier otra transacción que pueda haber utilizado los resultados de T_j*. El mensaje R^* es propagado de generación en generación hasta alcanzar el más joven ancestro activo de T_j, el cual denotamos $SAA(T_j)$. El abandono de $SAA(T_j)$, corresponde a nuestro objetivo inicial: siempre que una restricción sea violada todas las hojas que la *tocan* son abandonadas, conjuntamente con las transacciones que pudieran haber visto sus resultados. Es obvio que la identidad de $SAA(T_j)$ no puede ser determinado estáticamente, dado que depende del progreso alcanzado por la ejecución de la transacción anidada. El abandono de una transacción cualquiera sea la razón (por no verificación de una restricción de integridad, por abrazo mortal, por orden de su madre, etc.) puede provocar un efecto colateral sobre restricciones previamente verificadas, obligándonos a verificarlas nuevamente. Para ello, es necesario modificar el procedimiento de abandono de una transacción a fin de agregar el envío de un mensaje A^* a todas las transacciones responsables de mantener restricciones *tocadas* por el sub-árbol del cual la transacción sea raíz. Es de hacer notar que éste es el único caso en que modificamos el comportamiento de las transacciones anidadas.

3 Prototipo de un manejador de transacciones anidadas

El desarrollo de un prototipo de un manejador de transacciones anidadas que integra al control de ejecución la verificación de restricciones de integridad, nos permitió validar la solución descrita en la sección anterior. El prototipo implementado contempla el funcionamiento del manejador de transacciones sólo bajo una situación de "procesamiento normal" [5], es decir no se consideran posibles fallas en el sistema que obliguen a un reinicio, por tanto el prototipo no está preparado para una recuperación a fallas.

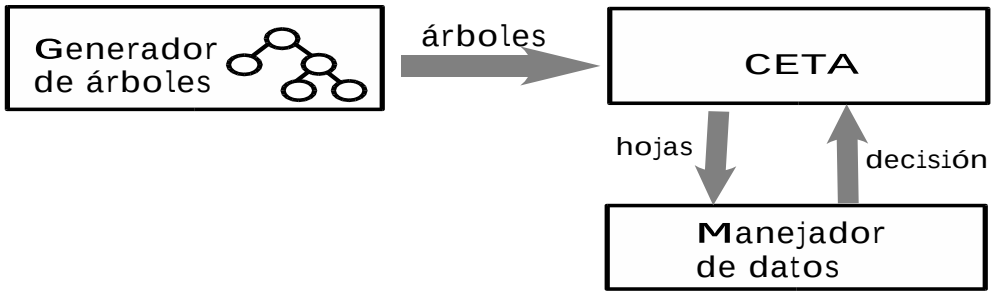

Figure 3: Arquitectura del Manejador de Transacciones Anidadas

La implementación se realizó sobre un computador PC Pentium III 800 Mhz, con 256 MB de RAM, bajo Windows NT. Para el desarrollo se utilizó el lenguaje Java, JDK 1.2.2, dado las facilidades que ofrece para la generación de hilos de ejecución (threads) y su portabilidad para realizar pruebas sobre otras plataformas. La figura 3 muestra la arquitectura del prototipo de manejador de transacciones anidadas, conformado por tres componentes: el control de ejecución de transacciones anidadas (CETA), el generador de árboles y el manejador de datos. El CETA implementa las funcionalidades de control de ejecución y verificación de restricciones del manejador de transacciones. El objetivo que se persigue con la implementación del prototipo, es realizar la evaluación del algoritmo propuesto en [3, 8] para la verificación de restricciones de integridad en las transacciones anidadas. Por lo tanto, nuestro esfuerzo se centró en la implementación de este componente. El generador de árboles se encarga de la creación de árboles y debería constituir una interfaz con los usuarios. El manejador de datos constituye el nivel de manejo de datos y operaciones que se corresponde a un SMBD. Estos dos componentes son provistos en este prototipo a través de simulación. A continuación se describe cada uno de estos componentes.

3.1 Generador de árboles

En este módulo se generan los árboles de transacciones que servirán de datos de prueba para el estudio que realizamos. Para la generación de los árboles se consideraron los siguientes parámetros en el modelo de simulación.

- Una transacción es desarrollada con algún objetivo determinado. Por esta razón consideramos que la transacción raíz es obligatoria, durante la construcción de un árbol se garantiza que al menos una de sus ramas está constituida por sub-transacciones obligatorias y la probabilidad de que una sub-transacción sea obligatoria es 0,80.

- La profundidad de los árboles es una variable aleatoria distribuida uniformemente entre los valores 1 y 4. Esto se consideró razonable, en base a estudios que muestran que las aplicaciones que se modelan a través de transacciones jerárquicas no son muy profundas, por estar estructuradas en base a los nodos que visitan [1, 4, 12]. La cantidad de nodos del árbol está asociada a su profundidad, a mayor profundidad mayor cantidad de nodos.

- La cantidad de hijas por cada transaccion madre está uniformente distribuida entre 2 y 5 hijas por nodo. El modo de ejecución (en paralelo, secuencial ó alternativo) de las hijas es una variable uniformemente distribuida.

- Se consideró la existencia de un *conjunto finito de restricciones globales* en el sistema cuya cardinalidad es definida en base a una variable aleatoria uniformemente distribuida entre 10 y 20 restricciones. La cardinalidad del sub-conjunto $C(T_j)$ de las *restricciones tocadas* por cada transacción hoja T_j es determinada a través de una función de probabilidad, la cual da más peso a cardinalidades menores, de la siguiente manera:
  - Probabilidad de que la hoja toque 0 restricciones: 0,3.
  - Probabilidad de que la hoja toque 1 restricción: 0,35.
  - Probabilidad de que la hoja toque 2 restricciones: 0,25.
  - Probabilidad de que la hoja toque 3 restricciones: 0,1.
- El conjunto de restricciones que cada hoja toca denotado como $C(T_j)$, es modelado a través de una distribución uniforme por cada elemento, sin reposición.

3.2 Control de ejecución de una transacción anidada

El CETA recibe un árbol que representa una transacción anidada que debe ser ejecutada. Fué implementado como un servidor iterativo, es decir soporta la ejecución de un árbol a la vez. La ejecución de un árbol se basa en las estructuras de datos mostradas en la figura 4, las cuales corresponden a :

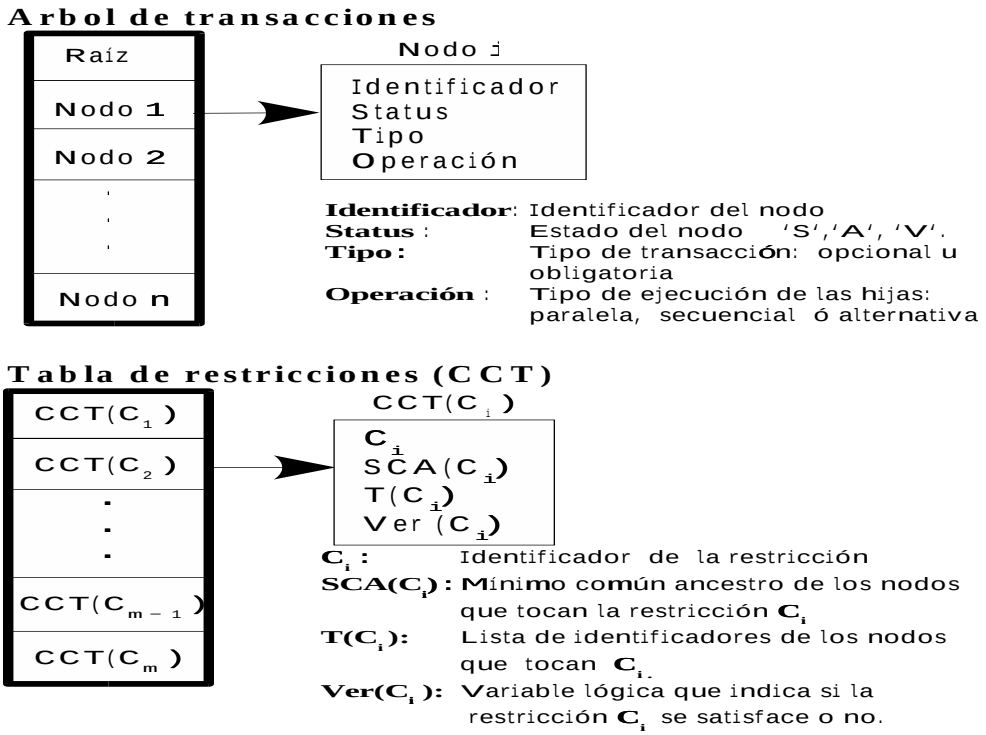

Figure 4: Estructuras de datos utilizadas en el prototipo del MT

- El árbol de transacciones, que mantiene la información de la estructura de la transacción y el estado de cada sub-transacción.
- La tabla de restricciones, que contiene información de las restricciones de integridad tocadas por la transacción anidada.

Para la ejecución de una transacción anidada, el árbol será recorrido por niveles, basándonos en la creación de un hilo de ejecución (thread) para cada sub-transacción. Cada hilo generado tendrá como función la ejecución de un sub-árbol del original y responde al siguiente comportamiento:

1. Si la raíz es un operador paralelo, se genera un hilo por cada sub-transacción hija. De esta manera se ejecutan concurrentemente las distintas ramas.

2. Si la raíz es un operador secuencial, se crearán en orden hilos de ejecución, uno por cada transacción hija, tomándolas de izquierda a derecha, y esperando la respuesta del que se esté ejecutando antes de iniciar el siguiente. De esta manera se ejecutan secuencialmente las distintas ramas del árbol.

3. Si la raíz es un operador alternativo, se crearán en orden hilos de ejecución, uno por cada transacción hija, tomándolas de izquierda a derecha y esperando la respuesta del que se esté ejecutando antes de iniciar el siguiente. Cuando se obtenga la primera respuesta de validación de alguna de las sub-transacciones hijas, las restantes no serán iniciadas. De esta manera se asegura que a lo sumo una de las hijas que se dan como alternativas valide.

4. Si la raíz del árbol es una transacción hoja, se envía un mensaje de requerimiento de ejecución a la capa subyacente, que llamamos Manejador de datos, y el hilo se queda esperando hasta recibir la respuesta. Esta respuesta corresponde al modo de terminación o decisión tomada por la transacción hoja: *validó ó abandonó*. La decisión tomada por la hoja es enviada tanto a su madre, como a aquellas transacciones encargadas de verificar las restricciones que toca.

5. Cuando la transacción $SCA(C_i)$ responsable de la restricción C_i recibe la respuesta de todas las hojas que tocan C_i, inicia un hilo de ejecución que corresponde a la verificación de C_i. La generación automática del código asociado a los procesos de chequeo de restricciones $Check(C_i)$ no fué considerado en este trabajo, por lo tanto estos procesos son simulados. El tiempo de ejecución de estos procesos es una variable aleatoria uniformemente distribuida entre los valores 1 a 5 segundos. El resultado del chequeo de una restricción, es decir si C_i se satisface ó no, es simulado a través de una distribución de probabilidad que de acuerdo a los valores iniciales dará lugar a diferentes escenarios de prueba como se verá en la siguiente sección. Los valores de esta función probabilística son asignados como parámetros al iniciar el prototipo del manejador.

6. En el caso que una restricción C_i no se satisfaga, el hilo correspondiente al $SCA(C_i)$ envía mensajes hacía abajo, hasta alcanzar al mínimo ancestro activo de cada transacción hoja que haya tocado C_i. Estas transacciones son terminadas abruptamente (abandonadas) realizando la reverificación de restricciones a que dieran lugar estos abandonos.

7. Cada $SCA(C_i)$ antes de tomar su propia decisión de abandonar o validar, revisa la tabla de restricciones para comprobar si se debe realizar la verificación de C_i.

8. La decisión de cada transacción madre (de validar o abandonar) es consecuencia de la estructura del árbol y es tomada por el mecanismo de control de ejecución el cual considera: el modo de ejecución de las hijas (secuencial, paralelo o alternativo), la decisión de cada una de las hijas, el tipo de dichas transacciones (opcional u obligatoria) y el resultado de los procesos de verificación de restricciones.

3.3 Manejador de datos

Esta capa subyacente al CETA es simulada, de manera que no hay una base de datos definida y las hojas de los árboles no realizan accesos a datos, dado que el manejo de datos no es lo esencial a probar en este trabajo. El comportamiento del manejador de datos es modelado a través de funciones de distribución probabilísticas. El manejador de datos recibe del CETA la solicitud de ejecución de una transacción hoja y retorna la decisión (abandono ó validación) tomada por dicha hoja. De esta manera sin tener definida una base de datos, se simula la ejecución de las hojas de los árboles. El manejador de datos soporta la ejecución de múltiples hojas a la vez, y para cada hoja se genera un hilo (thread) de ejecución. La duración ó tiempo de ejecución de cada hoja es una variable aleatoria uniformemente distribuida entre 1 y 10 seg. Con este margen amplio de posibles tiempos de ejecución tomamos en cuenta la posibilidad de ejecución de operaciones simples (lectura y/o escritura) o complejas, dadas las características del computador sobre el cual se realizó la implementación. La decisión tomada por cada transacción hoja, de validar o abandonar, es modelada a través de una función de probabilidad donde la decisión de validar tiene asignada una probabilidad de 0,90 mientras que una decisión de abandono la tiene de 0,10. Estas probabilidades fueron asignadas pensando

que el objetivo de una transacción es validar y sólo en situaciones excepcionales toma ella por si misma la decisón de abandonar.

4 Evaluación del prototipo

El objetivo de desarrollar un prototipo de manejador de transacciones anidadas, fué disponer de una herramienta que nos permitiera estudiar el comportamiento del algoritmo de verificación de restricciones de integridad propuesto en [3, 8]. Para ello, consideramos dos escenarios de prueba:

1. En un primer escenario consideramos un alto índice de violación de restricciones durante la ejecución de una transacción. Para este escenario **pesimista** asignamos una probabilidad de 0,5 a cada posible resultado del proceso $Check(C_i)$. Esto es, suponemos igual de probable que una restricción se satisfaga ó no durante el proceso de chequeo.

2. Un segundo escenario el cual llamamos **optimista**, es aquel en donde es poco probable que una transacción viole las restricciones de integridad. En este caso, estariamos modelando un sistema estable y depurado en el cual las transacciones han sido probadas exhaustivamente y una violación puede surgir sólo ante ciertos estados eventuales de la base de datos. Para este escenario **optimista** consideramos que con una probabilidad de 0,90 el proceso de $Check(C_i)$ arrojará como resultado que C_i se satisface y con una probabilidad de 0,10 el resultado será que C_i es violada.

Para estudiar el comportamiento del algoritmo en cada uno de estos dos (2) escenarios, se diseñaron experimentos que nos permitieran obtener la siguiente información estadística acerca de los resultados de la ejecución de un conjunto representativo de transacciones anidadas:

- La proporción de transacciones globales que validan y por ende, la proporción de transacciones que abandonan.

- La proporción de transacciones globales que logran validar a pesar de haber violado durante su ejecución al menos una restricción de integridad. Este experimento nos dice el porcentaje de transacciones "salvadas" por nuestro método, en comparación con la aplicación de otro que verifique las restricciones al final de la transacción global.

- La proporción de transacciones globales en las cuales se realizó al menos una reverificación de restricciones. Esto nos permite hacer inferencias acerca de cuan frecuente es necesario realizar reverificaciones de una restricción, lo cual si bien es necesario, constituye el aspecto computacional más costoso en el algoritmo propuesto.

- Discriminación de las transacciones globales que abandonaron de acuerdo a la proporción de la transacción que ya se había ejecutado, al momento del abandono. Este experimento nos permite mostrar el trabajo innecesario evitado por nuestro método al realizar la verificación lo más pronto posible durante la ejecución de una transacción anidada. El progreso alcanzado por una transacción al momento de su abandono (PTA) se modeló en base a la cantidad de sub-transacciones completamente ejecutadas (TE) sobre la cantidad total de sub-transacciones del árbol (TT), de acuerdo a la siguiente fórmula :

$$PTA = \frac{TE}{TT}$$

4.1 Resultados obtenidos en la evaluación

Se generaron dos muestras conformadas por 10.000 árboles cada una. Este tamaño de la muestra nos permite garantizar nuestros resultados con una probabilidad de 0,95 y un ϵ de 0,01[4]. Una muestra fué ejecutada en el prototipo bajo el escenario *pesimista* y la otra en el *optimista*. A continuación se presentan los resultados obtenidos en cada caso.

[4] $P(|\hat{p} - p|) \geq \epsilon) \leq 0,05$. Donde $\hat{p}$ es la proporción estimada y p la característica de la población.

	Escenario Pesimista	Escenario Optimista
Transacciones abandonadas	5.060 (50,6%)	2.658 (26,58%)
Transacciones validadas	4.940 (49,4%)	7.342 (73,42%)
Validadas con violación de RI	1.691 (16.91%)	807 (8,07%)
Media del PTA	57,03%	70,90%
Trabajo inútil evitado	42,97%	29,10%
Al menos una re-verificación	493	439

Las Transacciones abandonadas y validadas, son la cantidad de transacciones que logran validar y abandonar en cada escenario considerado.
Las transacciones validadas con violación de R.I. son aquellas que son *salvadas* gracias a la aplicación de este algoritmo, ya que logran validar a pesar de que han violado alguna restricción durante su ejecución, y por ende abandonado algunas sub-transacciones.
La gráfica de la figura 5 presenta los histogramas de frecuencia para los valores del PTA agrupados en intervalos de 10 en las transacciones que abandonaron, en cada uno de los escenarios considerados.

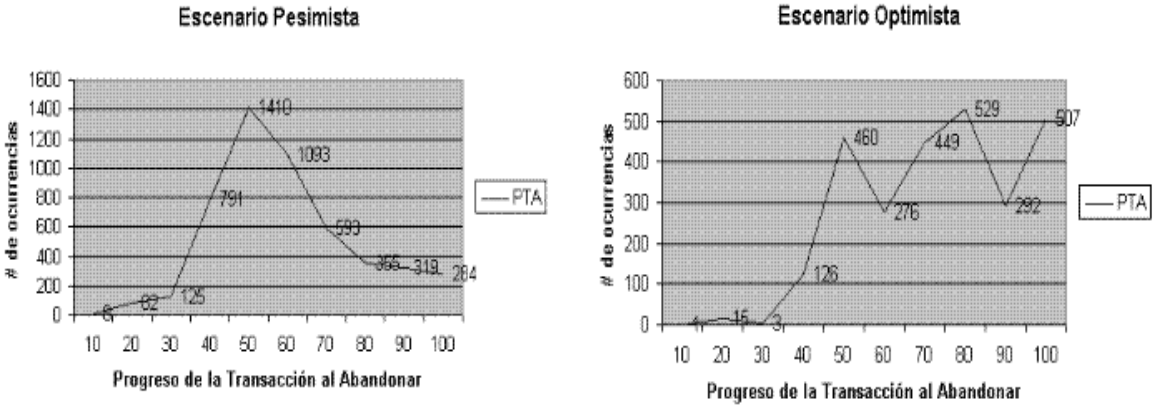

Figure 5: Histogramas de frecuencia del PTA en intervalos de 10

La proporción de trabajo inútil evitado es 100 menos la media del PTA. Representa la proporción de trabajo que la aplicación de este algoritmo logra evitar que se realice. Con cualquier otro mecanismo de chequeo de restricciones, inevitablemente la transacción global será igualmente abandonada por violación de la integridad. Sólo que, nuestro algoritmo provoca su abandono temprano y no espera al final de la transacción.
Con la finalidad de comprobar los resultados en cuanto a la reverificación de restricciones, se repitió la prueba 30 veces con 10.000 transacciones cada vez, en cada escenario. En estas nuevas pruebas, la cantidad de transacciones que realizan al menos una reverificacion para el escenario optimista oscilan entre un mínimo de 394 y un máximo de 468, con una media de 428 en las 30 pruebas. Para el escenario pesimista los resultados variaron entre 420 y 531, siendo la media 491 transacciones. Estos resultados, siempre cercanos al 4 % de la muestra, parecen indicar que la frecuencia de reverificación de restricciones es una variable independiente de los dos escenarios de prueba considerados aqui.

5 Conclusión.

En este artículo se presenta la implementación de un prototipo de un manejador de transacciones anidadas que integra un algoritmo de verificación de restricciones de integridasd al control de ejecución de las transacciones. La idea principal del algoritmo es asociar el control de una restricción a la transacción que corresponde en el árbol al ancestro común más joven de todas las sub-transacciones que *tocan* la restricción.
Los resultados obtenidos nos permiten afirmar que el algoritmo constituye una solución eficiente para la verificación de restricciones de integridad ante la presencia de transacciones anidadas. En un escenario

optimista, donde la probabilidad de que una restricción de integridad sea violada por una transacción la suponemos de 0.1 %, la contribución es notable, ya que el algoritmo logra *salvar* el 8.07 % del total de transacciones. Ante un escenario *pesimista*, donde la probabilidad de que una restricción sea violada por una transacción la suponemos 0.5, los resultados son aún mejores, ya que nuetsro algoritmo permite *salvar* el 16.91 % del total de transacciones.
Con este algoritmo, las restricciones se verifican lo más pronto posible durante la ejecución de una transacción anidada, abandonando la transacción completa cuando la violación de la integridad es insalvable. Esto constituye una ganancia con respecto a métodos donde la verificación de restricciones se realiza al final de la transacción raíz. En el escenario *optimista* se evita que se ejecuten en promedio el 29.1 % de las sub-transacciones de las transacciones anidadas que inevitablemente serán abandonadas, mientras que en el escenario *pesimista* el porcentaje de trabajo inútil evitado es de 42.97 % en promedio.

References

[1] Y. Breitbart, H. García-Molina, and A. Silberschatz. Overview of Multidatabase Transaction Management. *VLDB Journal*, 1(2):181–240, October 1992.

[2] B. Defude and H. Martin. Integrity checking for Nested Transactions. In R. Wagner and H. Thoma, editors, *Proc. 7th Int. Conf. Database and Expert Systems Applications, DEXA'96*, pages 147–152, Zurich (Switzerland), September 1996. IEEE-CS Press.

[3] A. Doucet, S. Gançarski, C. León, and M. Rukoz. Nested Transactions with Integrity Constraints. In G. Saake, K. Schwarz, and C. Türker, editors, *Transactions and Database Dynamics, 8th Int. Workshop on Foundations of Models and Languages for Data and Objects,TDD'99, Dagstuhl Castle, Germany, September 27-30, 1999, Selected Papers*, volume 1773 of *LNCS*, pages 130–149, Berlin, 2000. Springer-Verlag.

[4] H. Garcia-Molina and K. Salem. Sagas. In U. Dayal and I. L. Traiger, editors, *Proc. of the 1987 ACM SIGMOD Int. Conf. on Management of Data*, volume 16 of *ACM SIGMOD Record*, pages 249–259, San Franscisco, CA (USA), May 1987. ACM Press.

[5] J. Gray and A. Reuter. *Transaction Processing: Concepts and Techniques*. Morgan Kaufmann Publishers, San Mateo, CA, 1993.

[6] P. W. P. J. Grefen. *Integrity Control in Parallel Database Systems*. PhD thesis, Universiteit Twente, The Netherlands, 1992.

[7] H. F. Korth and G. D. Speegle. Formal Aspects of Concurrency Control in Long-Duration Transactions Systems Using the NT/PV Model. *ACM Transactions on Database Systems*, 19(3):492–535, September 1994.

[8] C. León. *Transacciones Anidadas y Restricciones de Integridad*. PhD thesis, Universidad Central de Venezuela and Université de Paris VI, Caracas, Venezuela, 2001.

[9] J. E. B. Moss. *Nested Transactions: An Approach to Reliable Distributed Computing*. MIT Press, Cambridge, MA, 1985.

[10] P.-Y. Policella. *Cohérence dans les Bases de Données Orientées Objet*. PhD thesis, Université de Paris XI, Paris, France, 1996.

[11] A.P. Sheth, M. Rusinkiewicz, and G. Karabatis. Using Polytransactions to Manage Interdependent Data. In A. K. Elmagarmid, editor, *Database Transaction Models for Advanced Applications*, pages 555–581. Morgan Kaufmann Publishers, San Mateo, CA, 1992.

[12] G. Weikum, A. Deacon, W. Schaad, and H.-J. Schek. Open Nested Transaction in Federated Database Systems. *IEEE Data Engineering Bulletin*, 16(2):4–7, June 1993.