Estrategias para verificar restricciones de integridad globales en multi base de datos con transacciones anidadas *

Anne Doucet[1], Stephane Gançarski[1], Claudia León[1,2], Marta Rukoz[2,3]

[1]: LIP6, Université 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.

[3]: School of CS, University of Central Florida, P.O. Box 162362, Orlando Fl 32816-2362. USA

e-mail: {*Anne.Doucet,Stephane.Gancarski,Claudia.Leon*}*@lip6.fr*, marta@cs.ucf.edu

Abstract

Este artículo propone estrategias para la verificación de diversos tipos de restricciones de integridad en los sistemas multi base de datos que soportan transacciones anidadas. La solución presentada en [7] para ambientes de base de datos centralizadas, se extiende a sistemas multi base de datos. Esta solución está basada en designar por cada restricción de integridad, una sub-transacción responsable de controlar su verificación durante la ejecución de una transacción anidada: el ancestro común más joven de todas las sub-transacciones propensas a violar la restricción.

En una multi base de datos, es necesario tomar en cuenta la estructura de la restricción y la localización de las transacciones y de los datos implicados, para determinar el sitio donde se debe ejecutar la verificación, a fin de minimizar la transferencia de datos a través de la red. Es por ello, que proponemos diferentes estrategias de verificación dependiendo del tipo de restricción a ser chequeada.

Palabras claves: Multi base de datos. Restricciones de integridad. Transacciones anidadas.

1 Introducción.

Uno de los aspectos más importantes en una base de datos (BD) es mantener su coherencia. Esta funcionalidad es generalmente asegurada a través de restricciones de integridad (RI) las cuales son aserciones lógicas que deben ser satisfechas al finalizar cada transacción. Muchos trabajos de investigación han sido consagrados a este problema [6, 13, 11] y la mayoría de los SMBD comerciales proveen alguna funcionalidad en este sentido [18, 9]. Sin embargo, pocos estudios han sido realizados para el manejo de restricciones de integridad en los sistemas multi base de datos. En tales sistemas, donde existen múltiples BD administradas en diferentes sitios [11, 5], se plantean nuevos problemas para el mantenimiento de la coherencia. Uno de ellos, es determinar en cual(es) sitio(s) debe ser mantenida y chequeada cada restricción, a fin de minimizar la transferencia de datos. Aunque algunos trabajos han sido realizados en el ámbito de BD federadas [5], distribuidas

*Este trabajo fué parcialmente financiado por el CDCH-UCV y por el programa europeo ALFA-B3 proyecto número 7-0010-9

[13] y multi base de datos [12, 17], ninguno de ellos considera el modelo de transacciones anidadas como base para las aplicaciones de los usuarios.
El modelo de transacciónes anidadas (TA), una transacción compuesta de una jerarquía de sub-transacciones, después de haber sido propuesto por Moss (ver [14]) ha sido objeto de vastos estudios. Se han desarrollado múltiples sistemas que las soportan [15, 4] y algunos SMBD las proveen entre sus funcionalidades [1, 9]. Sin embargo, pocos trabajos tratan el problema de verificación de restricciones en TA. Aunque parezca natural que para tales transacciones las restricciones sean verificadas al finalizar cada sub-transacción, ésta no es una buena solución. En efecto, podría llevar a que una restricción sea verificada repetidas veces si es tocada por muchas sub-transacciones, o lo que es peor, que se detecte una incoherencia en una sub-transacción que podría ser resuelta por otra aún no terminada.
En [6] y [7] abordan la verificación de RI en el contexto de ambientes centralizados que soportan TA. En [6] las RI son definidas a nivel de los métodos, a través de pre- y postcondiciones, asociadas a un mecanismo de manejo de excepciones. Bajo ese enfoque, el programador debe describir las acciones a seguir cuando una restricción es violada y a que nivel de la TA se ejecutan esas acciones. En [7] se presenta un mecanismo que le agrega al control de ejecución de las TA la capacidad de verificar RI tan pronto como sea posible, sin interferir con el control de ejecución subyacente. Las RI son definidas globalmente a nivel del esquema de la BD y el código para la verificación de la restricción es insertado automáticamente siendo transparente para el usuario el manejo de RI.
Para mantener la coherencia en los sistemas de multi base de datos es necesario determinar donde debe ser definida, almacenada y verificada cada restricción. La elección de un sitio adecuado debe tomar en cuenta el tipo de restricción, el tipo de actualización, y los sitios donde las actualizaciones fueron efectuadas. En todo caso, se debe minimizar la transferencia de datos y se debe evitar bloquear otras transacciones, locales o globales, por mantener los bloqueos sobre los objetos más tiempo de lo necesario. En este artículo estudiamos cómo verificar las RI globales (las cuales tocan múltiples sitios) durante la ejecución de una transacción anidada en un sistema de multi base de datos. Extendemos la solución presentada en [7] para considerar la distribución de los datos.
Este artículo está organizado de la siguiente manera. La sección 2 introduce las características del enfoque para la verificación de RI que será utilizado en este trabajo. La sección 3, presenta el mecanismo propuesto en [7] para la verificación de RI en el contexto de TA. La sección 4 describe los problemas presentes en la verificación de RI en los sistemas multi base de datos. En la sección 5 presentamos diferentes estrategias para realizar la verificación de RI, de acuerdo al tipo de restricción y a la localización de las sub-trasacciones que la toquen. Finalmente, en la sección 6 presentamos las conclusiones y algunas perpectivas.

2 Verificación de restricciones de integridad.

Todo usuario parte del principio de que la base de datos que utiliza es coherente. Esta funcionalidad es asegurada generalmente a través de restricciones de integridad (RI), las cuales son aserciones lógicas que deben ser siempre satisfechas por la BD. Una BD es coherente si y sólo si, todas las restricciones son satisfechas. Sin embargo, la verificación del conjunto completo de restricciones luego de cada actualización a una BD representa un costo insostenible.
Numerosos trabajos han estudiado este problema [5, 13, 11] y los diferentes enfoques para abordarlo son presentados de manera resumida en [10]. Las soluciones que consisten en verificar las restricciones después de la ejecución de una transacción (métodos de detección) generalmente uti-

lizan técnicas de compilación para reducir el proceso de verificación durante la ejecución [3, 2]. Nuestro trabajo está situado dentro de este enfoque y por lo tanto se puede adaptar a todo mecanismo de manejo de restricciones dentro de un SMBD que utilice los siguientes principios:

- Las RI son declaradas de forma global a nivel del esquema de la BD.
- Un análisis sintáctico de las restricciones y de las transacciones permite reducir el conjunto de restricciones a verificar, ya que permite determinar el conjunto de restricciones susceptibles a ser violadas por una transacción. En este artículo diremos que el análisis sintáctico determina los objetos *involucrados* en una restricción y los objetos *tocados* por una transacción. Una transacción *toca* una restricción, si manipula objetos involucrados en la restricción, esto es, si la intersección de los dos análisis sintácticos no es vacía. De allí es posible determinar el conjunto de restricciones a verificar.
- La verificación de las restricciones se realiza de forma automática (por ejemplo, por generación del código). Para optimizar el costo del proceso de verificación, los algoritmos son adaptados y optimizados de acuerdo a los diferentes tipos de restricciones. Por ejemplo sólo las modificaciones hechas sobre instancias de caminos que aparezcan en el análisis sintáctico son tomados en cuenta para la verificación. Más aún, para las restricciones universales no es necesario verificar sobre todos los objetos de la BD involucrados en la restricción, ya que si suponemos que inicialmente la BD es coherente, basta con considerar sólo los objetos modificados por la transacción. De esta manera, es posible limitar la verificación a 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.

3 Transacciones anidadas y verificación de restricciones de integridad.

Las transacciones anidadas [14] se ajustan a los sistemas de multi base de datos donde las transacciones largas pueden descomponerse en varias tareas lógicamente independientes [19, 1, 4, 15].
Una transacción anidada (TA) es un árbol de transacciones de profundidad arbitraria donde los componentes son sub-transacciones. La transacción en el tope del árbol es la *transacción raíz*. Las transacciones que tienen sub-transacciones son llamadas *padres*, y sus sub-transacciones son sus *hijas*. Las transacciones que no poseen sub-transacciones son llamadas *hojas*.
Cada transacción controla la ejecución de sus hijas, por tanto comienza antes y termina después que sus hijas. Cada sub-transacción se ejecuta independientemente y eventualmente en paralelo con las otras sub-transacciones, lo cual significa que ella puede decidir validar o abandonar de forma independiente. Esta decisión puede depender del resultado de la ejecución de sus hijas. Si una sub-transacción abandona sus hijas deben obligatoriamente abandonar, pero no necesariamente ocasiona el abandono de su padre. Si una sub-transacción decide validar, la validación no es definitiva ya que sus actualizaciones serán realizadas solamente si todas las transacciones que son sus ancestros deciden validar, lo cual significa que se realizarán sólo si la transacción raíz valida. Cuando la transacción raíz valida, es necesario garantizar que se realicen las actualizaciones de todas aquellas sub-transacciones que decidieron validar y ninguno de sus ancestros abandonó. De allí que una TA puede validar manteniendo la coherencia aún si algunas de sus sub-transacciones abandonaron. Esto se denomina '*abandono parcial*'.

El comportamiento de las TA con respecto a las propiedades ACID es el siguiente: todas las transacciones de una TA (incluyendo la raíz) deben respetar este nuevo comportamiento así como también la propiedad clásica de aislamiento (*isolation*). Sin embargo, sólo la raíz debe garantizar la coherencia de los datos y su durabilidad.
Muchas variantes del modelo de TA propuesto por Moss [14] han sido definidas y pueden ser consultadas en [8]. En este artículo consideramos el modelo conocido como '*closed nested transaction*' donde sólo las hojas de una TA pueden realizar actualizaciones a la BD, por tanto son las únicas propensas a violar las RI. Esta condición permite simplificar el proceso de verificación de las RI, sin restarle expresividad al modelo de transacciones [14]. Así mismo, consideramos que la estructura de una TA es conocida al momento de su compilación, por lo tanto es posible conocer el ancestro común más joven de un conjunto de hojas de una TA al momento de su compilación.
En este artículo extendemos a sistemas multi base de datos, la solución propuesta en [7] para verificar restricciones en sistemas de BD centralizadas que soportan TA. El principio fundamental de la solución propuesta en [7] radica en seleccionar para cada restricción una sub-trasacción responsable de su verificación : el ancestro común más joven de las hojas que tocan la restricción. Las técnicas de análisis de los métodos de detección descritos en la sección 2 son utilizados para determinar durante la compilación :

- Para cada hoja Tl_j de una TA el conjunto $C(Tl_j)$ de todas las restricciones tocadas por Tl_j.
- Para cada restricción C_i el conjunto $T(C_i)$ de hojas Tl que tocan C_i. Denotamos con $SCA(C_i)$ al ancestro común más joven de todas las sub-transacciones del conjunto $T(C_i)$.

$SCA(C_i)$ es responsable del mantenimiento de la restricción C_i. Dado que esta transacción controla la ejecución de su sub-árbol, es posible asegurar que en caso de violación de C_i, no sólo las hojas que tocaron C_i serán abandonadas, sino también todas las sub-transacciones que hayan podido utilizar sus resultados. Por otro lado, y a diferencia de los sistemas que utilizan transacciones planas, las sub-transacciones que no tocan C_i pueden continuar su ejecución.
La comunicación directa entre las hojas que tocan una restricción C_i y la sub-transacción $SCA(C_i)$, permite iniciar la ejecución del proceso de verificación, $Check(C_i)$, apenas terminen su ejecución todas las hojas que tocan C_i. En caso de violación, la propagación, en el sub-árbol cuya raíz es $SCA(C_i)$ de un mensaje de abandono, permite abandonar sólo las sub-transacciones necesarias para restaurar la satisfacción de la restricción.
En [7] se muestra que aún cuando una restricción C_i deba ser verificada varias veces, la verificación *efectiva* (la última) se realiza siempre lo más pronto posible, apenas el estado final de la BD con respecto a C_i se produzca. También se muestra que una situación donde C_i deba ser re-verificada por una sub-transacción ya terminada es imposible.

4 Restricciones de integridad globales en multi base de datos.

Un sistema multi base de datos soporta operaciones sobre múltiples BD componentes, cada una manejada por un SMBD. En estos sistemas, es posible tener diferentes arquitecturas y diferentes niveles de integración entre los componentes, correspondientes a diferentes niveles de servicios globales [16]. Sin embargo solamente la distinción entre transacción global y transacción local influye sobre nuestro enfoque. En efecto, nuestro objetivo es mantener las restricciones de integridad globales (aquellas que involucran datos almacenados en diferentes sitios) en un sistema multi

base de datos. En esos caso, es conocido que la presencia de transacciones locales produce un problema difícil de resolver: controlar una restricción que toca varios sitios cuando un único sitio controla la ejecución de la transacción local. Es por ello que en este trabajo no consideramos las transacciones locales.

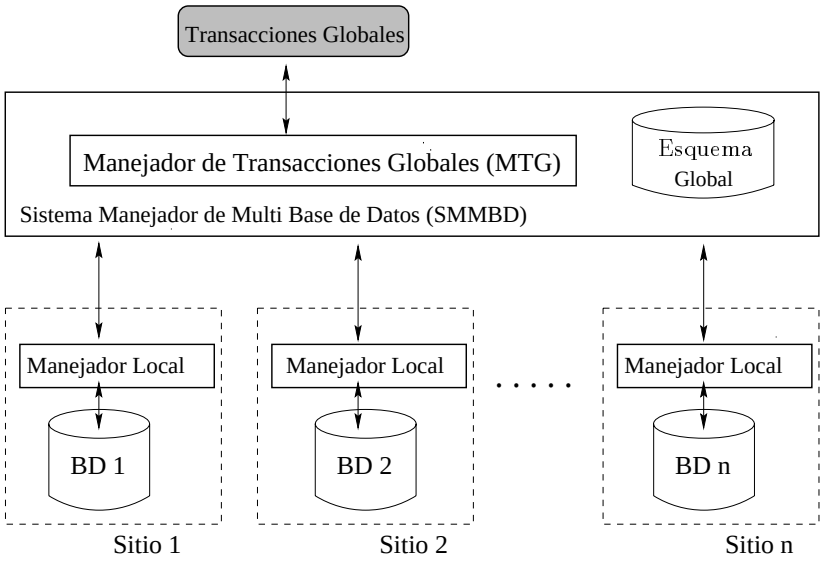

Figure 1: Arquitectura del sistema multi base de datos.

La figura 1 muestra la arquitectura (clásica) del sistema multi base de datos que tomamos en consideración, la cual posee las características siguientes :

1. Todas las transacciones son globales y manejadas por el manejador de transacciones globales (MTG) con la ayuda de un esquema global que describe cada uno de los datos y su localización. El MTG garantiza el control de concurrencia entre transacciones y entre sub-transacciones de una transacción (ver por ejemplo [19]).

2. Como mencionamos en la sección 3, utilizamos un modelo de TA donde sólo las hojas acceden a los objetos. Cada hoja es ejecutada en un sólo sitio y sólo accede a datos en dicho sitio.

3. No tomamos en cuenta eventuales duplicaciones de un mismo dato sobre diferentes sitios.

La asignación de sub-transacciones de una TA a los diferentes sitios es inducida por la distribución de los datos. Cuando una transacción global es iniciada sobre un sitio, sus sub-transacciones son iniciadas recursivamente, hasta encontrar una sub-transacción con la totalidad de sus hojas ejecutándose sobre un mismo sitio. Esta sub-transacción es enviada al sitio en cuestión, donde será ejecutado completamente su sub-árbol. La figura 2 ilustra esta política. La transacción global T es iniciada sobre el sitio S_i. Las sub-transacciones T_1 y T_2, las cuales tienen hojas sobre diferentes sitios, serán ejecutadas sobre el sitio iniciador S_i. Por el contrario, la sub-transacción T_{11} quien tiene todas sus hojas sobre el sitio S_1 es enviada al sitio S_1, donde se ejecutará todo su sub-árbol. Un sub-árbol asignado a un sitio puede reducirse a una sóla hoja, como es el caso de Tl_{12} y Tl_{21}. La solución presentada en la sección 3 debe ser extendida a fin de tomar en cuenta la distribución de los datos y de las sub-transacciones sobre diferentes sitios del sistema multi base de datos. El principio fundamental de esa solución continua siendo el mismo: el control de la verificación de

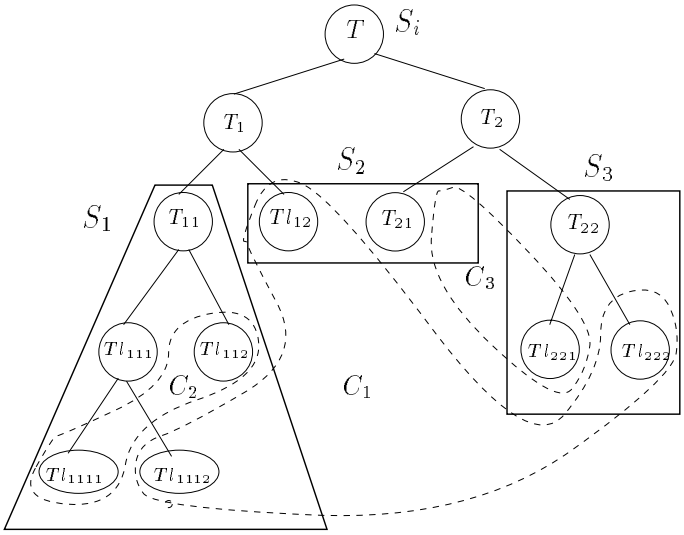

Figure 2: Restricciones globales tocadas por una transacción anidada distribuida.

una restricción C_i es realizado por el ancestro común más joven de las hojas $SCA(C_i)$ que tocan la restricción. Nótese que $SCA(C_i)$ está ubicado o sobre el mismo sitio donde están todas las hojas que tocan C_i o sobre el sitio donde se inició la transacción global. Sin embargo, la verificación efectiva de una restricción C_i, realizada a través del proceso $Check(C_i)$, debe tomar en cuenta los costos de comunicación entre los sitios para optimizar el tiempo de verificación. Se pueden distinguir dos casos :

1. La restricción C_i es *local*, es decir involucra datos almacenados sobre un único sitio S_j. Es evidente que la solución más eficaz es ejecutar $Check(C_i)$ sobre S_j.

2. La restricción C_i es *global*, es decir, involucra datos localizados en diferentes sitios. En ese caso, es necesario escoger el o los sitios adecuados para ejecutar el proceso de verificación a fin de minimizar la cantidad y el tamaño de los mensajes a transmitir sobre la red.

Para realizar la verificación de restricciones globales de forma eficaz es necesario considerar tanto la estructura de la restricción como los sitios sobre los cuales son ejecutadas las hojas que la tocan. Este problema es ilustrado en la figura 2 donde los datos involucrados por cada restricción C_1, C_2 o C_3 están delimitados por líneas en puntillas y las hojas que tocan parte del dominio de una restricción están incluidas en dicho dominio. Así, C_1 involucra datos sobre los sitios S_1, S_2 y S_3 y es tocada por Tl_{1112} sobre S_1, Tl_{12} sobre S_2 y Tl_{222} sobre S_3. C_2 es *local* a S_1 y es tocada por Tl_{112} y Tl_{1111} sobre ese sitio. C_3 involucra datos sobre los sitios S_2 y S_3 y es tocada por Tl_{221} sobre S_3. A través de este ejemplo podemos observar que definir una estrategia de verificación resulta más o menos complejo según el caso. Asi :

- Para la restricción C_2, es claro que el proceso de verificación puede realizarse completamente sobre S_1, ya que ningún dato de otro sitio es necesario para esa verificación.

- Para la restricción C_3, aunque la única hoja que la toca es ejecutada sobre S_3, es probable que el proceso de verificación sea ejecutado principalmente sobre S_2, si C_3 es una restricción universal. Si fuera éste el caso, la evaluación de C_3 necesita comparar los datos modificados

sobre S_3 por Tl_{221} con los datos almacenados en S_2 asociados a la restricción. Evaluar C_3 en S_2 requiere sólo transferir los datos modificados en S_3 hacia S_2, mientras que su evaluación sobre S_3 requiere transferir todos los datos de S_2 involucrados en la restricción, incluso aquellos que no son afectados directa o indirectamnente por la transacción.

- Para la restricción C_1 la situación es todavía más compleja. Su evaluación puede ser ejecutada en S_1, S_2, S_3 o en cualquier combinación de esos sitios, dependiendo tanto de la naturaleza de la restricción, como de las acciones realizadas por Tl_{1112}, Tl_{12} y Tl_{222}.

5 Estrategias para la verificación de restricciones globales.

Para las restricciones globales es necesario determinar el(los) mejor(es) sitio(s) donde realizar su verificación, lo cual depende principalmente del tipo de restricción. En esta sección presentamos una tipología de restricciones globales sobre una multi base de datos así como estrategias de verificación a seguir para cada tipo de restricción estudiada.

5.1 Restricciones globales verificables localmente.

Las restricciones globales que pueden ser expresadas a través de una conjunción o de una disyunción de restricciones locales pueden ser verificadas utilizando un conjunto de procesos locales.

5.1.1 Conjunción de restricciones locales.

Las restricciones globales que se pueden expresar como una conjunción de restricciones locales son aquellas de la forma: $C_1 \wedge C_2 \ldots \wedge C_n$, tales que, para todo i, C_i es una restricción local a la BD del sitio S_i. Si las C_i son completamente independientes (i.e si ellas tocan clases diferentes no relacionadas entre ellas), C puede ser reemplazada, en la fase de concepción de la restricción, por la conjunción de las C_i. Por ejemplo, si C es de la forma $\forall o_1 \in Cl_1, \forall o_2 \in Cl_2, F(o_1, o_2)$, con las clases Cl_1 (resp. Cl_2) sobre el sitio S_1 (resp. S_2) y $F(o_1, o_2)$ puede re-escribirse como: $F_1(o_1) \wedge F_2(o_2)$. En estos casos se pueden declarar dos restricciones independientes: $C_1 : (\forall o_1 \in Cl_1, F_1(o_1))$ en el sitio S_1 y $C_2 : (\forall o_2 \in Cl_2, F_2(o_2))$ en el sitio S_2.
En este grupo entran también aquellas restricciones definidas sobre una clase fragmentada. Por ejemplo, si la restricción C es de la forma $\forall o \in Cl, F(o)$ y la clase Cl está fragmentada sobre $S_1, \ldots, S_n$. Si llamamos Cl_{S_i} el fragmento de Cl localizado en S_i, entonces la restricción C se transforma en $(\forall o_1 \in Cl_{S_1}, F(o_1)) \wedge (\forall o_2 \in Cl_{S_2}, F(o_2)) \ldots \wedge (\forall o_n \in Cl_{S_n}, F(o_n))$.
Estrategia de verificación : La estrategia a seguir para este tipo de restricciones es simple. En el momento de la compilación de la restricción C, se re-escribe como $C_1, \ldots, C_n$. C se satisface si y sólo si se satisface el conjunto de C_i. Para mantener C es suficiente con mantener cada C_i, cuya verificación puede ser realizada localmente en el sitio correspondiente.

5.1.2 Disyunción de restricciones locales.

Las restricciones globales que se pueden expresar como una disyunción de restricciones locales son aquellas de la forma: $C_1 \vee C_2 \ldots \vee C_n$, tal que C_i es una restricción local a la base de datos del sitio S_i. Dentro de tales restricciones se encuentran las restricciones existenciales sobre una clase fragmentada. Así una restricción de la forma: $\exists o \in Cl, F(o)$ con Cl fragmentada en

$Cl_1, Cl_2, \ldots Cl_n$ respectivamente sobre los sitios $S_1, S_2, \ldots S_n$ puede re-escribirse como: $(\exists o_1 \in Cl_1, F(o_1)) \vee (\exists o_2 \in Cl_2, F(o_2)) \ldots \vee (\exists o_n \in Cl_n, F(o_n))$.

Estrategia de verificación : Diversas estrategias pueden ser utilizadas para verificar una disyunción de restricciones locales de la forma $C : C_1 \vee C_2 \ldots \vee C_n$. Para ello es necesario dintinguir cuales C_i son tocadas por las hojas de la TA y cuales no. Para simplificar, supongamos las restricciones locales $C_1, C_2 \cdots C_k$, con $k \leq n$ son tocadas por la TA y las restricciones $C_{k+1} \cdots C_n$ no lo son.

Un primer método para es efectuar la verificación de las restricciones locales $C_{k+1} \cdots C_n$ en paralelo con la ejecución de la transacción, es decir sin esperar el fin de la ejecución de las hojas que tocan la restricción. Como estas restricciones no son afectadas por la TA, la satisfacción de una de ellas es suficiente para garantizar la satisfacción de la restricción global, independiente de los efectos de la transacción. En ese caso, no será necesario verificar las restricciones locales $C_1 \cdots C_k$ después de la ejecución de las hojas. Por el contrario, si ninguna de las restricciones $C_{k+1} \cdots C_n$ es satisfecha, entonces $SCA(C)$ debe esperar el fin de las hojas que tocan C para ejecutar los procesos de verificación locales correspondientes a $C_1 \cdots C_k$ para determinar si alguna de ellas es satisfecha. Si ninguna de las $C_1 \cdots C_k$ se satisface $SCA(C)$ iniciará el proceso de abandono parcial descrito en la sección 3. Si bien, este método presenta el inconveniente de acceder a los sitios $S_{k+1}, \ldots, S_n$ no tocados por la transacción, su ventaja es que aumenta el paralelismo y evita tener que esperar que las hojas terminen para comenzar el proceso de verificación.

Un segundo método es proceder de manera inversa y esperar la culminación de las transacciones hojas que tocan C para iniciar la verificación sobre los sitios tocados $S_1, \ldots, S_k$ por la TA. Si ninguna restricción local es satisfecha $SCA(C)$ lanza la verificación en $S_{k+1}, \ldots, S_n$ para determinar si alguna C_j es satisfecha. Este método tiene la ventaja de acceder a los sitios no tocados por la transacción sólo si es necesario, pero reduce el paralelismo.

En el caso que sólo transacciones hojas sobre un mismo sitio toquen la restricción C, por ejemplo S_j, un tercer método es verificar C_j antes de la ejecución de las hojas sobre S_j. En efecto, si C_j no es satisfecha entonces existe un sitio S_i (diferente de S_j), no tocado por la transacción sobre el cual C_i es satisfecha, ya que suponemos que C, la cual es una disyunción, se satisface al inicio de la transacción. En consecuencia, se tiene la certeza que C será satisfecha despues de la ejecución de la TA, independientemente de los efectos de las hojas sobre S_j. Si C_j es satisfecha al inicio de la transacción, cualquiera de los métodos precedentes puede ser aplicado. Esta estrategia puede generalizarse a los casos donde C puede ser tocada por hojas sobre diferentes sitios, pero sólo resulta conveniente si el número de sitios donde existen hojas es muy pequeño con respecto al número total de sitios implicados por la restricción.

Note que esos tres métodos pueden ser optimizados si el sistema mantiene información sobre la satisfacción de las diferentes componentes de la disyunción.

5.2 Restricciones globales no verificables localmente.

Una restricción que no pueda ser verificada por procesos locales independientes (como los casos precedentes), contiene predicados y cuantificadores que necesitan una verificación simultánea sobre diferentes sitios. Tales restricciones pueden tener una complejidad arbitraria, ya que pueden ser definidas a partir de cualquier tipo de predicado, de cuantificador y de conector, lo cual hace imposible establecer una estrategia de verificación general. En este artículo, nos restringimos a una clase importante de restricciones, que llamamos *restricciones globales conjuntivas*. Una

restricción de esta clase puede ser expresada de la siguiente forma:

$$\forall(o_1 \in Cl_1, \cdots, o_n \in Cl_n) F_1(\vec{x}_1) \wedge F_2(\vec{x}_2) \ldots \wedge F_j(\vec{x}_j) \Rightarrow F_{j+1}(\vec{x}_{j+1}) \wedge F_{j+2}(\vec{x}_{j+2}) \ldots \wedge F_m(\vec{x}_m)$$

donde F_i es un predicado cualquiera y $\vec{x}_i$ un sub-conjunto cualquiera de variables $o_1, o_2, \cdots, o_n$. Este tipo de restricciones aparece generalmente cuando existen cadenas de relaciones entre objetos de diferentes BD. Este es el caso de la figura 3 donde existe una relación entre la clase *Vehículo* en el sitio S_1 y la clase *Persona* en el sitio S_2, a través de la referencia *propietario*. Sobre esa multi base de datos podemos definir una restricción global universal C tal que:

$$C : \forall(v \in Vehiculo, p \in Persona), (marca(v) = Ferrari \wedge propietario(v) = p) \Rightarrow edad(p) \geq 33$$

C establece que el *Propietario* de todo *Vehículo* que es un *Ferrari* debe tener al menos 33 años.

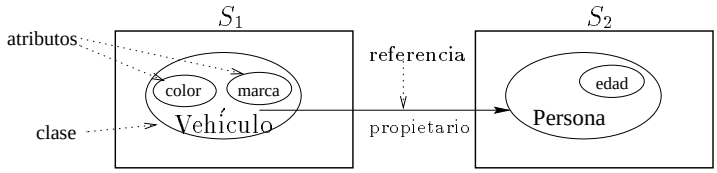

Figure 3: Relación de S_1 con S_2 a través de la referencia propietario

Como mencionamos en la sección 2, para las restricciones universales se agrupan, en el momento de la ejecución, los objetos modificados por una transacción y se verifica la restricción sólo con respecto a esos objetos (y aquellos relacionados a través de la restricción). En un contexto distribuido, es necesario minimizar la transferencia de objetos sobre los cuales la restricción será verificada. Para ello, se realiza un análisis previo de cada restricción con un doble propósito.
El primer propósito, inspirado en [13], es determinar para los sitios donde sea posible un predicado localmente evaluable, cuya satisfacción con respecto a los objetos modificados sobre ese sitio sea suficiente para garantizar la satisfacción de la restricción global. Ese predicado será designado en adelante por PL_i. En el ejemplo de la figure 3 PL_1 corresponde a $marca(v) \neq Ferrari$ y PL_2 corresponde a $edad(p) \geq 33$.
El segundo propósito es basado en el método propuesto en [12]. Consiste en descomponer una restricción global cuantificada universalmente sobre una conjunción de predicados a través de la definición de *predicados intersitios*. La idea es obtener una sub-restricción para cada BD que describa la responsabilidad de dicha BD con respecto a la satisfacción de la restricción global, más una restricción particular que relaciona todos los predicados intersitios con la restricción a verificar. Esta última restricción es almacenada en un sólo sitio, en donde la verificación final de la restricción tendrá lugar. Existen tantas descomposiciones posibles para una restricción global universal como sitios estén implicados por ella. Por lo tanto, para la restricción C existen dos descomposiciones:

1. Si C se verifica sobre S_1, es re-escrita de la manera siguiente:

   En S_1**:** $\forall(v \in Vehiculo, p \in Persona\_menor\_33), marca(v) = Ferrari \Rightarrow propietario(v) \neq p$

   En S_2**:** $\forall(p \in Persona), edad(p) < 33 \Leftrightarrow p \in Persona\_menor\_33$

   Persona_menor_33 es el conjunto asociado al predicado intersitio, que contiene los identificadores de los objetos de la clase *Persona* tocados por la transacción que no satisfacen el predicado local PL_2.

2. Si C se verifica sobre S_2, es re-escrita de la manera siguiente:

   En S_1: $\forall(v \in Vehiculo, p \in Persona\_con\_ferrari), (marca(v) = Ferrari \wedge propietario(v) = p) \Leftrightarrow p \in Persona\_con\_ferrari$

   En S_2: $\forall(p \in Persona), p \in Persona\_con\_ferrari \Rightarrow edad(p) \geq 33$

   Persona_con_ferrari es el conjunto asociado al predicado intersitio que contiene los identificadores de los objetos de la clase *Persona* tocados por la transacción que no satisfacen el predicado local PL_1.

Estrategia de verificación : Como estrategia de verificación de esta clase de restricciones proponemos guardar todas las descomposiciones posibles y utilizar en el momento de la verificación aquella que minimice el intercambio de datos. Esta selección se realiza dinámicamente en base a los sitios donde se ejecuten las transacciones que tocan la restricción.
$SCA(C)$ inicia, en el sitio donde se encuentre, el proceso $Check(C)$ encargado de la verificación de C. Este proceso será la raíz de una transacción anidada distribuida que tendrá un hijo en cada sitio S_i donde existan sub-transacciones hojas que toquen C. Cada hijo, llamado $Check(CMJ_i)$, estará encargado de la verificación de C con respecto a las actualizaciones hechas en S_i. Los procesos $Check(CMJ_i)$ tendrán el comportamiento siguiente:

1. Verificar PL_i si existe. Si PL_i es satisfecho, no será necesario realizar otras acciones, ya que C será satisfecha con respecto a las modificaciones hechas en S_i.

2. Si PL_i no es satisfecho o no existe, se utilizan los predicados intersitios para identificar los objetos sobre S_i necesarios para hacer la verificación de la restricción global. En este caso, se envían al sitio S_j donde se va a verificar C, las modificaciones realizadas en S_i. Luego, se inicia en S_j el proceso de verificación propiamente dicho.

Note que este método de verificación origina una transacción anidada $Check(C)$ distribuida sobre diferentes sitios: los sitios que contienen hojas que tocan C y los sitios donde se debe verificar C. Para ilustrar esto, retomemos la restricción C definida sobre los datos de los sitios S_1 y S_2 de la figura 3. Para la verificación de C se genera una de las tres transacciones siguientes, dependiendo de los sitios donde fueron ejecutadas las hojas que tocan C:
Primer Caso: Todas las hojas que tocan C se ejecutan en S_1. La estructura de la transacción $Check(C)$ corresponde al sub-árbol izquierdo de la TA representada en la figura 4. $Check(C)$ tendrá una sóla hija $Check(CMJ_1)$ encargada de la verificación de C con respecto a las modificaciones realizadas en S_1. $Check(CMJ_1)$ tendrá dos sub-transacciones secuenciales (indicado por el símbolo ;). La primera, $Check(PL_1)$, verifica el predicado local PL_1 suficiente para garantizar la satisfacción de C. Si PL_1 no es satisfecho, es decir si las transacciones hojas dan como nuevo valor *Ferrari* en la *marca* de un $Vehiculo$, o si ellas asignan un nuevo *propietario* a un *Ferrari*, entonces es necesario construir el conjunto $Intersitio_1$ con los identificadores p_j de los propietarios de esos vehículos. Este conjunto será enviado a S_2 donde $Check(CL_2)$ chequeará si todo propietario p_j de $Intersitio_1$ tiene una edad superior o igual a 33.
Segundo Caso: Todas las hojas que tocan C se ejecutan en S_2. Este caso es simétrico al precedente. Aquí las modificaciones conciernen al atributo *edad(p)* de los objetos de la clase *Persona*. Si existen actualizaciones que producen una *edad* inferior a 33 en una *Persona*, es necesario construir el conjunto $Intersitio_2$ conteniendo los identificadores p_j de dichas personas. Este conjunto será enviado al sitio S_1 donde se chequeará si toda persona de ese conjunto no es

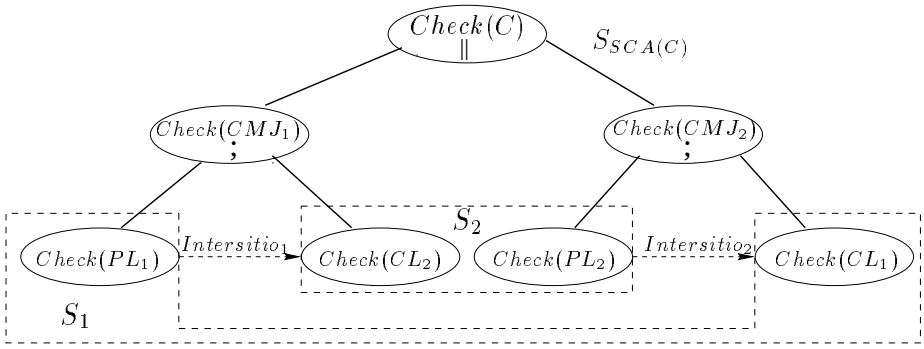

Figure 4: Verificación de una restricción global

propietaria de un *Ferrari*. La estructure de la transacción $Check(C)$ corresponde al sub-árbol derecho de la transacción de la figura 4.

Tercer Caso: Las hojas que tocan C se ejecutan en S_1 y en S_2. Dado que una hoja se ejecuta completamente en un sitio, entonces existen unas transacciones que tocan la clase *Vehículo* y otras transacciones que tocan la clase *Persona*. La figura 4 muestra la estructura de la TA $Check(C)$ que tendrá dos sub-transacciones, $Check(CMJ_1)$ y $Check(CMJ_2)$, que se ejecutarán en paralelo (indicado por el símbolo $\|$). $Check(CMJ_i)$ verifica la restricción C con respecto a todas las hojas que se ejecutan en S_i. Los procesos $Check(CMJ_1)$ y $Check(CMJ_2)$ utilizan dos conjuntos de datos diferentes $Intersitio_2$ y $Intersitio_1$ respectivamente, lo cual permite la paralelización.

6 Conclusión.

En este artículo hemos estudiado diferentes estrategias de verificación de restricciones globales definidas de manera declarativa en un ambiente multi base de datos que soporta transacciones anidadas. Hemos mostrado como minimizar el costo de verificación al minimizar la transferencia de datos entre los sitios. Mediante una tipología de restricciones, establecemos diferentes estrategias en función de la naturaleza de la restricción y de la estructura de la transacción. Las restricciones globales estudiadas son aquellas expresadas a través de una conjunción (resp. disjunción) de restricciones locales y una clase importante de restricciones globales cuantificadas universalmente. Para estas últimas utilizamos el método de descomposición descrito en [12] así como la definición de predicados localmente evaluables que garantizan la satisfacción de restricciones globales [13]. Al igual que en [7], el proceso de verificación de una restricción es controlado por el ancestro común más joven de todas las hojas de una transacción anidada que tocan la restricción. El proceso de verificación es estructurado como una transacción anidada distribuida, lo cual permite su paralelización.

Según nuestro conocimiento, nuestro enfoque es el único que trata la verificación de restricciones globales en presencia de transacciones anidadas. Otros trabajos sobre el mantenimiento de restricciones de integridad globales [13, 11, 12, 5] sólo se interesan en modelos de transacciones distribuidas con un sólo nivel de anidamiento.

Actualmente estudiamos la implantación del método propuesto en este artículo sobre un sistema que soporta transacciones anidadas distribuidas : SIMA un sistema para la manipulación de imágenes médicas [15]. Por otra parte, esperamos extender nuestro enfoque a fin de tomar en

cuenta otra clase de restricciones globales, principalmente las restricciones de agregación.

References

[1] Transarc Encina Product Information. http://www.transarc.com/Solutions. http://www.transarc.com/Product/Txseries/Encina/.

[2] V. Benzaken and A. Doucet. Thémis: A Database Programming Language Handling Integrity Constraints. *The VLDB Journal*, 4(3):493–517, July 1995.

[3] B. T. Blaustein. *Esforcing Database Assertions*. PhD thesis, Harvard University, Cambridge, MA, 1981.

[4] E. Boertjes, P. W. P. J. Grefen, J. Vonk, and P. M. G. Apers. An Architecture for Nested Transactions Support on Standard Database Systems. In G. Quirchmayr, E. Schweighofer, and T. J. M. Bench-Capon, editors, *Proc. 9th Int. Conf. Database and Expert Systems Applications, DEXA'98*, volume 1460 of *LNCS*, pages 448–459, Vienna (Austria), August 1998. Springer-Verlag.

[5] S. Conrad, M. Höding, S. Janssen, G. Saake, I. Schmitt, and C. Türker. Integrity Constraints in Federated Database Design. Technical Report 2, Fakultät fur Informatik, Universität Magdeburg, April 1996.

[6] 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.

[7] 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.

[8] A. K. Elmagarmid, editor. *Database Transaction Models For Advanced Applications*. Morgan Kaufmann Publishers, San Mateo, CA, 1992.

[9] D. Enser and I. Stevenson. *Oracle8 Design Tips*. O'Reilly & Associates, Cambridge, MA, 1997.

[10] P. W. P. J. Grefen and P. M. G. Apers. Integrity Control in Relational Database Systems — An Overview. *Data & Knowledge Engineering*, 10:187–223, 1993.

[11] P. W. P. J. Grefen and J. Widom. Protocols for Integrity Constraint Checking in Federated Databases. *Distributed and Parallel Databases*, 5(4):327–355, October 1997.

[12] S. Grufman, F. Samson, S.M. Embury, P.M.D. Gray, and T. Risch. Distributing Semantic Constraints Between Heterogeneous Databases. In Alex Gray and Per-Åke Larson, editors, *Proceedings of the Thirteenth International Conference on Data Engineering, ICDE 1997, April 7-11*, pages 33–42, Birmingham U.K., April 1997. IEEE Computer Society.

[13] A. Gupta and J. Widom. Local Verification of Global Integrity Constraints in Distributed Databases. In P. Buneman and S. Jajodia, editors, *Proc. of the 1993 ACM SIGMOD Int. Conf. on Management of Data*, volume 22 of *ACM SIGMOD Record*, pages 49–58, Washington, D.C.(USA), May 1993. ACM Press.

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

[15] M. Rukoz, C. León, and M. Rívas. SIMA: A Java Tool for Constructing Image Processing Applications on a Heterogeneous Network. *to appear in Parallel and Distributed Computing Practices. Special Issue on Distributed Object Systems*, 2000. Acepted 1999.

[16] A.P. Sheth and J.A. Larson. Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases. *ACM Computing Surveys*, 22(3):183–236, September 1990.

[17] 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.

[18] Sybase Inc. Press, Emeryville,CA (USA). *Sybase SQL Server*, 1989.

[19] 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.