ANOTACIONES PARA EL DISEÑO DE UNA BASE DE DATOS ORIENTADA A OBJETOS Y DISTRIBUIDA

*John Freddy Duitama M**
*Pascual Uribe M.***

RESUMEN

El presente trabajo toma como punto de referencia la metodología usada para el diseño de una base de datos distribuida relacional, evalúa que problemas nuevos aparecen cuando se habla de una B. de D. orientada a objetos y propone un tratamiento a tales aspectos.

Palabras clave: Diseño de bases de datos, bases de datos distribuidas, objetos distribuidos.

ABSTRACT

On the basis of the methodology used to design relational distributed data bases, this paper assesses which are the new issues of the object oriented data bases and proposes how to manage them.

1. ESCENARIO

Nuestro propósito es tomar la metodología DATAID-D propuesta por [Ceri87] y orientada a la fragmentación horizontal; tomar igualmente la metodología de fragmentación vertical propuesta en [Nava84] y la evaluación de costos para un diseño distribuido relacional presentada en [Özsu91] e introducir los elementos de análisis necesarios para adecuarlas a un ambiente orientado a objetos.

Las metodologías mencionadas permiten fragmentar las relaciones de modo horizontal o vertical; posteriormente permiten definir en que nodo se ubica la primera copia de cada fragmento y en que situaciones es viable definir réplicas, todo bajo el criterio de la máxima localidad, es decir, que la información se ubique en el nodo que más la usa. Posteriomente permiten evaluar las decisiones tomadas a la luz de otros criterios, como el de carga de trabajo asignada a cada nodo. Para más detalles véase [Duit99] o [Urib99].

En un ambiente orientado a objetos, partimos de la existencia de un esquema global representado como un modelo de clases en notación U.M.L. compuesto por clases que contienen atributos y métodos; además aparecen tres tipos de asociaciones: de generalización, de agregación y relaciones simples. Véase [Booc 99] para más detalles.

* Profesor Facultad de Ingeniería. Universidad de Antioquia.

** Analista de Sistemas. Empresas Públicas de Medellín.

Definimos una transacción como un programa de computador que, para su realización ,invoca una serie de métodos de los definidos para las clases que usa.

Asumimos como esquema de procesamiento básico el que aparece en la figura 1.

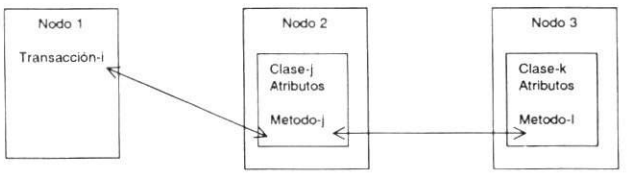

Figura 1. Esquema de procesamiento

Cada transacción es emitida desde un nodo donde se ejecuta (nodo 1); cuando esta transacción requiere de un método envía un mensaje al nodo donde se encuentra (nodo 2); allí es ejecutado el método invocado y al finalizar retorna su respuesta a la transacción que lo invoca. Es posible que al ejecutarse este primer método (método-j) él a su vez invoque otros métodos (método-l); de nuevo el procedimiento es el mismo. El método-j envía el mensaje al método-l, éste se ejecuta en el nodo 3, donde se halla , y finalmente retorna la respuesta al método llamante.

2. ASPECTOS RELEVANTES

Los elementos más relevantes que aparecen en este nuevo escenario son:

- Los métodos: significa que en el momento de fragmentar y localizar una clase se debe considerar , además de sus atributos, qué hacer con sus métodos.

- Las asociaciones de generalización y agregación: su semántica adicional debe tenerse presente en el momento de definir la fragmentación de las clases.

- El encapsulamiento de la información: asumimos que toda operación sobre una clase se realiza únicamente a través de los métodos.

- La clasificación de los métodos: requerimos clasificar los métodos en: los de solo lectura y los de actualización. Además nos interesa, de cada método, su costo en operaciones de CPU y costos de acceso a disco; pues es posible que en el momento de localizar un método se dé prioridad al criterio de distribución de carga, lo que implica que métodos con excesivo uso de CPU pueden ubicarse en sitios diferentes al nodo donde está la clase que los contiene.

3. REQUERIMIENTOS DE DISTRIBUCIÓN

- Es posible obtener una matriz por clase (MC): En las filas , sus atributos y en las columnas, cada uno de sus métodos: de tal manera que se pueda conocer qué atributos de la clase usa cada método y cómo los usa (para consulta o para modificación).

- Requerimos para cada transacción conocer qué métodos involucra y cuántas veces los invoca, incluyendo los que usa indirectamente (porque son invocados por otros métodos dentro de la transacción). Con esta información construimos por cada clase una matriz (TM) de transacciones contra métodos; en cada celda aparece:

  0: si la transacción no usa este método.

  n: si la transacción invoca *n* veces a este método

- A partir de la información anterior es posible construir una tabla de acceso lógico como lo requiere *dataid-d*: las transacciones contra los atributos de las clases.

- Para cada transacción requerimos conocer la frecuencia de ejecución desde cada sitio al igual que en *dataid-d*. $F_{i,j}$ = Frecuencia de la transacción i desde el sitio j.

- Una información adicional pero de utilidad posterior radica en conocer para cada método: $C_{CPU}(Mi)$ = costo en CPU del método-i y $C_{I/O}$ (Mi) = Costo en acceso a disco del método-i. Este costo incluye los recursos consumidos por los otros métodos que Mi invoque durante su ejecución.

3.1. Fragmentación Horizontal

Los criterios establecidos para la fragmentación horizontal no cambian cuando hay una asociación simple entre dos clases. Fundamentalmente se parte de los predicados más usados por los métodos invocados en la transacciones más relevantes. Igualmente se puede realizar fragmentación principal y fragmentación derivada.

Si se encuentra una asociación de agregación entre dos clases, de nuevo es válido el criterio de la fragmentación derivada en la clase componente a partir de la fragmentación ya realizada sobre la clase compuesta.

Si existe una generalización debe tratarse como un problema de fragmentación vertical.

3.2. Fragmentación vertical

Para las clases no fragmentadas horizontalmente se puede aplicar el algoritmo de fragmentación vertical reseñado en [Nava84] y que consta de tres pasos: hallar la matriz de afinidad entre atributos, construir la matriz agrupada a partir de la matriz de afinidad y finalmente, desde la matriz agrupada y usando el algoritmo de partición, definir los posibles fragmentos de la relación.

Para las generalizaciones se puede usar el mismo algoritmo, pero con una variante: Aunque la definición de una clase y varias subclases de hecho establece una fragmentación vertical entre los atributos de la superclase y los atributos de cada subclase, es probable que no convenga separar físicamente estos fragmentos, pues los métodos de la subclase pueden requerir frecuentemente de atributos heredados. Para evaluar tal situación se puede aplicar el algoritmo de partición, obviando los pasos anteriores (hallar la matriz de afinidad y la matriz agrupada). Se toma la subclase incluyendo los atributos y métodos heredados, se definen los conjuntos tope y base, con los atributos de la superclase en el primero y los atributos que solo aparecen en la subclase en el segundo; si el Z evaluado da un valor positivo entonces se reafirma la partición fisica de ambos grupos de atributos; de otra manera se descarta. Si se optó por fragmentar la generalización puede continuarse la evaluación por separado para cada clase que aparezca en el camino de generalización. En estos casos deben aplicarse completamente los tres pasos del procedimiento de fragmentación vertical.

3.3. Localización de los fragmentos

Se procede al igual que en *dataid-d*, pues se cuenta con toda la información de entrada relevante; se sigue en todos los casos el criterio de máxima localidad. Se procede de igual manera para la definición de réplicas en la información.

Para la ubicación de los métodos, también bajo el criterio de máxima localidad, hay dos situaciones diferentes:

- Todo fragmento horizontal va acompañado de los métodos de la clase.

- Cada fragmento vertical va acompañado únicamente de los métodos que requieran de atributos que aparezcan en el fragmento, información que se puede obtener desde la matriz TM.

3.4. Evaluación de costos

La fórmula base para los cálculos tiene dos términos:

STC_{jk} =Costo de almacenar el fragmento j en el sitio k, que no tiene variaciones respecto a *dataid-d* y

QPC_i = Costo de procesar la transacción i en cada sitio que la use; puede calcularse apoyados en el C_{CPU}(Mi) y el $C_{I/O}$ (Mi) de cada método, multiplicados cada vez por el número de veces que la transacción los invoque. Los otros elementos que inciden, como costo de transmisión entre nodos y costo del control de concurrencia, continúan sin variación.

Si esta evaluación de costos pone en evidencia sobrecarga de trabajo en algunos nodos, puede optarse por trasladar los métodos de mayor costo en CPU y más frecuentemente usados (matriz TM) a otros sitios con menos carga de trabajo y de poco costo en comunicación con el sitio donde continúa ubicada la clase propietaria. Esta estrategia de distribución de carga de trabajo se puede realizar hasta que se logre un

equilibrio razonable. Se seleccionan los métodos de mayor costo en CPU y de poco costo de I/O, pues de esta manera se disminuye el volumen de datos por transmitir entre nodos y se libera de trabajo adicional la CPU del nodo de donde se remueve el método.

4. CONCLUSIONES

En general puede concluirse que, a partir de la introducción de unas cuantas variaciones a la metodología distribuida para bases de datos relacionales, es posible obtener un a estrategia satisfactoria y eficaz en ambientes orientados a objetos. En este nuevo escenario es posible realizar cálculos más preciso en costos de CPU, lo que facilita la labor de distribución de la carga de trabajo; además puede aprovecharse la fragmentación natural de las generalizaciones para simplificar los cálculos realizados por el proceso de fragmentación vertical.

Un elemento interesante de analizar pero no cubierto en este escrito, tiene que ver con la posibilidad de cambiar dinámicamente la ubicación de los fragmentos y de los métodos, aprovechando para ello las bondades que ofrecen algunos manejadores de bases de datos[Ston98]. Significa lo anterior que el proceso de distribución puede contemplar varios escenarios y tomar decisiones de diseño para cada uno de ellos; a continuación, utilizando el sistema de reglas del manejador de bases de datos, es posible instruir a éste para que migre dinámicamente los objetos, en la medida que cambien las circunstancias de uso de los datos y los métodos.

5. BIBLIOGRAFÍA

[BOOC99] BOOCH, G., J. RUMBAUGH, IVAR JACOBSON. 1999. The Unified Modeling Language. User Guide. Addisson Wesley Massachusetts.

[CERI87] CERI, S., B.PERNICI AND G.WIEDERHOLD, "Distributed Database Design Methodologies", Proceedings of the IEEE, vol.75, No. 5, pp.533-547, May 1987.

[DUIT99] DUITAMA, JOHN FREDDY. "Integración de Metodologías para el diseño de una Base de Datos Distribuida". Proyecto financiado por el CODI bajo el título: Fragmentación vertical para Optimabdd. S.p. 1999. Universidad de Antioquia.

[NAVA84] NAVATHE, S.B., S.CERI, G.WIEDERHOLD AND J. DOU, "Vertical partitioning algorithms for database design", ACM Transactions on Database Systems, vol.9, número 4, pp. 680-710, Dec.1984.

[ÖZSU91] ÖZSU, M.TAMER AND PATRICK VALDURIEZ. Principles of Distributed Database Systems. New Jersey: Prentice-Hall, 1991.

[STON98] STONEBRAKER, MICHAEL, PAUL M. AOKI, ROBERT DEVINE, WITOLD LITWIN AND MICHAEL OLSON. Mariposa: A New Architecture For Distributed Data. Computer Science Div., Dept. Of EECS, University of California, Berkeley, California 94720. 1998, 17p.

[URIB99] URIBE MÚNERA, PASCUAL. "Una metodología para el diseño de aplicaciones en ambiente cliente servidor bajo la visión objetual". Junio 1999. Tesis de Maestría Universidad Nacional-Medellín Posgrado en Ingeniería de Sistemas.