Anexo B:

Tipos de fragmentación de Base de Datos Distribuidas[1].
Resumido por José Luis Cisneros de los autores originales.

Dado que una relación se corresponde esencialmente con una tabla y la cuestión consiste en dividirla en fragmentos menores, inmediatamente surgen dos alternativas lógicas para llevar a cabo el proceso: la división horizontal y la división vertical. La división o fragmentación horizontal trabaja sobre las tuplas, dividiendo la relación en subrelaciones que contienen un subconjunto de las tuplas que alberga la primera. La fragmentación vertical, en cambio, se basa en los atributos de la relación para efectuar la división. Estos dos tipos de partición podrían considerarse los fundamentales y básicos. Sin embargo, existen otras alternativas. Fundamentalmente, se habla de fragmentación mixta o híbrida cuando el proceso de partición hace uso de los dos tipos anteriores. La fragmentación mixta puede llevarse a cabo de tres formas diferentes: desarrollando primero la fragmentación vertical y, posteriormente, aplicando la partición horizontal sobre los fragmentos verticales (denominada partición VH), o aplicando primero una división horizontal para luego, sobre los fragmentos generados, desarrollar una fragmentación vertical (llamada partición HV), o bien, de forma directa considerando la semántica de las transacciones. Otro enfoque distinto y relativamente nuevo [2], consiste en aplicar sobre una relación, de forma simultánea y no secuencial, la fragmentación horizontal y la fragmentación vertical; en este caso, se generara una rejilla y los fragmentos formaran las celdas de esa rejilla, cada celda será exactamente un fragmento vertical y un fragmento horizontal (nótese que en este caso el grado de fragmentación alcanzado es máximo, y no por ello la descomposición resultará más eficiente).

Volviendo a la figura 1, puede observarse como los casos C y D se basan en la mencionada generación de la rejilla, con la diferencia que en el primero de ellos se produce una fusión, una desfragmentación de las celdas, agrupándolas de la manera más adecuada para obtener mayor rendimiento, ya que los fragmentos generados son muy pequeños. En el segundo caso se asignan las celdas a los sitios y luego se realiza una rigurosa optimización de cada sitio. El caso E sería aquel en el que se utiliza la fragmentación VH o la fragmentación HV.

[1] Tomado de www.monografias.com, con el tema "Diseño de Bases de Datos Distribuidas".
José Luis Cisneros Cervantes – Jorge Javier Jirón Rosero

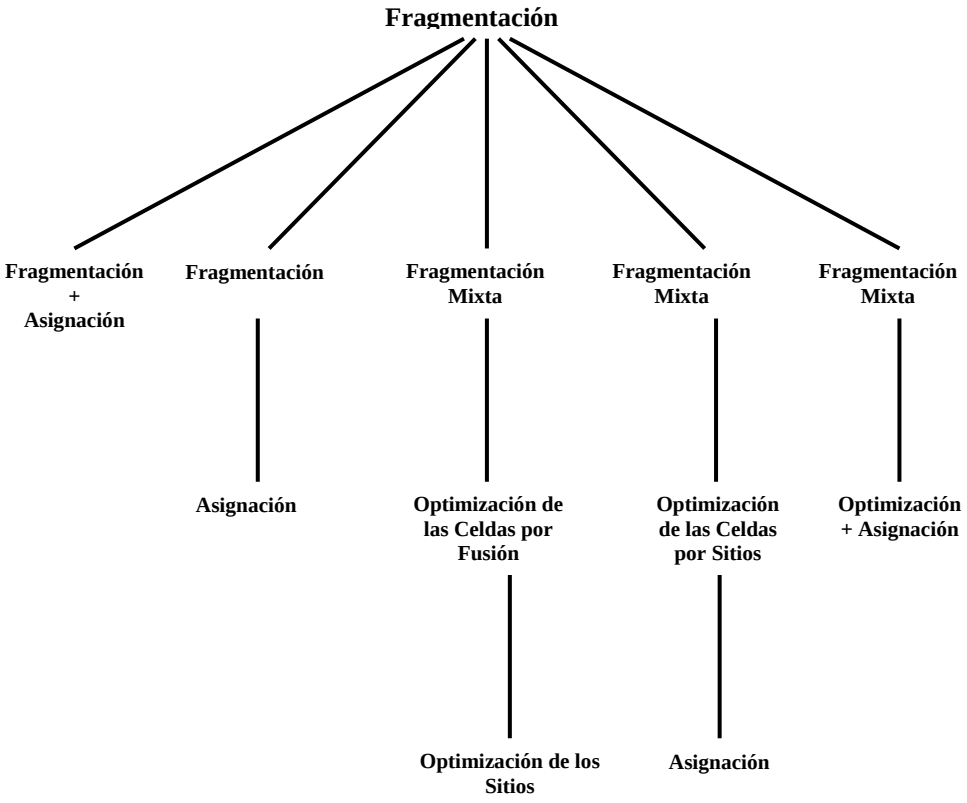

Figura 1. Enfoques para realizar el diseño distributivo.

Grado de fragmentación. Cuando se va a fragmentar una base de datos deberíamos sopesar qué grado de fragmentación va a alcanzar, ya que éste será un factor que influirá notablemente en el desarrollo de la ejecución de las consultas. El grado de fragmentación puede variar desde una ausencia de la división, considerando a las relaciones unidades de fragmentación; o bien, fragmentar a un grado en el cada tupla o atributo forme un fragmento. Ante estos dos casos extremos, evidentemente se ha de buscar un compromiso intermedio, el cual debería establecerse sobre las características de las aplicaciones que hacen uso de la base de datos. Dichas características se podrán formalizar en una serie de parámetros. De acuerdo con sus valores, se podrá establecer el grado de fragmentación del banco de datos.

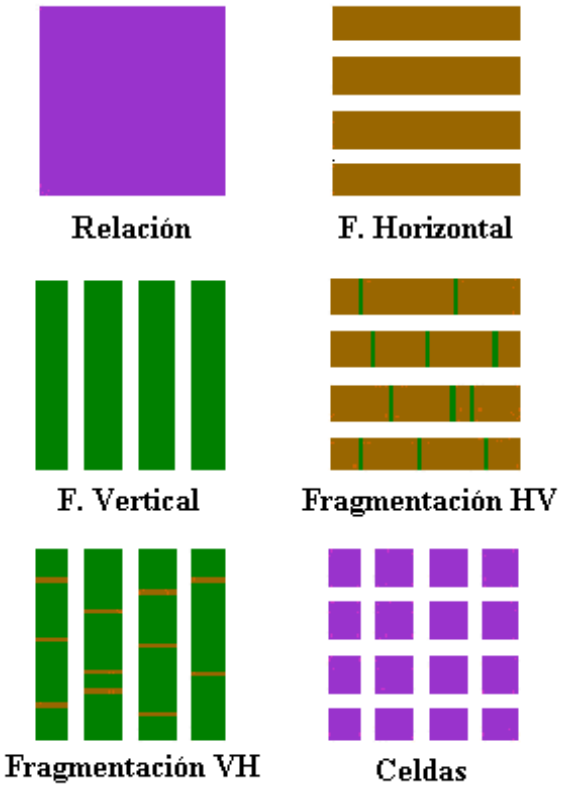

Figura 2. Distintos tipos de fragmentación.

Grado de fragmentación.

Cuando se va a fragmentar una base de datos deberíamos sopesar qué grado de fragmentación va a alcanzar, ya que éste será un factor que influirá notablemente en el desarrollo de la ejecución de las consultas. El grado de fragmentación puede variar desde una ausencia de la división, considerando a las relaciones unidades de fragmentación; o bien, fragmentar a un grado en el cada tupla o atributo forme un fragmento. Ante estos dos casos extremos, evidentemente se ha de buscar un compromiso intermedio, el cual debería establecerse sobre las características de las aplicaciones que hacen uso de la base de datos. Dichas características se podrán formalizar en una serie de parámetros. De acuerdo con sus valores, se podrá establecer el grado de fragmentación del banco de datos.

Reglas de corrección de la fragmentación.

A continuación se enuncian las tres reglas que se han de cumplir durante el proceso de fragmentación, las cuales asegurarán la ausencia de cambios semánticos en la base de datos durante el proceso.

1. Compleción. Si una relación *R* se descompone en una serie de fragmentos *R1, R2,..., Rn*, cada elemento de datos que pueda encontrarse en *R* deberá poder encontrarse en uno o varios fragmentos *Ri*. Esta propiedad extremadamente importante asegura que los datos de la relación global se proyectan sobre los fragmentos sin pérdida alguna. Tenga en cuenta que en el caso horizontal el elemento de datos, normalmente, es una tupla, mientras que en el caso vertical es un atributo.
2. Reconstrucción. Si una relación *R* se descompone en una serie de fragmentos *R1, R2, ..., Rn*, puede definirse una operador relacional ∇ tal que

$$R = \nabla R_i, \ \forall R_j \ \in \ F_R$$

   El operador ∇será diferente dependiendo de las diferentes formas de fragmentación. La reconstrucción de la relación a partir de sus fragmentos asegura la preservación de las restricciones definidas sobre los datos en forma de dependencias.

3. Disyunción. Si una relación *R* se descompone horizontalmente en una serie de fragmentos *R1, R2,..., Rn*, y un elemento de datos di se encuentra en algún fragmento *Rj*, entonces no se encuentra en otro fragmento *Rk* ($k \neq j$). Esta regla asegura que los fragmentos horizontales sean disjuntos. Si una relación *R* se descompone verticalmente, sus atributos primarios clave normalmente se repiten en todos sus fragmentos.

Alternativas de asignación.

Partiendo del supuesto que el banco de datos se haya fragmentado correctamente, habrá que decidir sobre la manera de asignar los fragmentos a los distintos sitios de la red. Cuando una serie de datos se asignan, éstos pueden replicarse para mantener una copia. Las razones para la réplica giran en torno a la seguridad y a la eficiencia de las consultas de lectura. Si existen muchas reproducciones de un elemento de datos, en caso de fallo en el sistema se podría acceder a esos datos ubicados en sitios distintos. Además, las consultas que acceden a los mismos datos pueden ejecutarse en paralelo, ya que habrá copias en diferentes sitios. Por otra parte, la ejecución de consultas de actualización, de escritura, implicaría la actualización de todas las copias que existan en la red, cuyo proceso puede resultar problemático y complicado. Por tanto, un buen parámetro para afrontar el grado de réplica consistiría en sopesar la cantidad de consultas de lectura que se efectuarán, así como el número de consultas de escritura que se llevarán a cabo. En una red donde las consultas que se procesen sean mayoritariamente de lectura, se podría alcanzar un alto grado de réplica, no así en el caso contrario. Una base de datos fragmentada es aquella donde no existe réplica alguna. Los fragmentos se alojan en sitios donde únicamente existe una copia de cada uno de ellos a lo largo de toda la red.

En caso de réplica, podemos considerar una base de datos totalmente replicada, donde existe una copia de todo el banco de datos en cada sitio, o considerar una base de datos parcialmente replicada donde existan copias de los fragmentos ubicados en diferentes sitios.

El número de copias de un fragmento será una de las posibles entradas a los algoritmos de asignación, o una variable de decisión cuyo valor lo determine el algoritmo. La figura 3 compara las tres alternativas de réplica con respecto a distintas funciones de un sistema de base de datos distribuido.

	Réplica total	**Réplica parcial**	**Partición**
Procesamiento de consultas	fácil	dificultad	Similar
Gestión del directorio	fácil o inexistente	dificultad	Similar
Control de concurrencia	moderado	Difícil	Fácil
Seguridad	muy alta	Alta	Baja
Realidad	posible aplicación	realista	Posible aplicación

Figura 3. Comparación de las alternativas de réplica

Información necesaria.

Un aspecto importante en el diseño de la distribución es la cantidad de factores que contribuyen a un diseño óptimo. La organización lógica de la base de datos, la localización de las aplicaciones, las características de acceso de las aplicaciones a la base de datos y las características del sistema en cada sitio, tienen una decisiva influencia sobre la distribución. La información necesaria para el diseño de la distribución puede dividirse en cuatro categorías: la información del banco de datos, la información de la aplicación, la información sobre la red de ordenadores y la información sobre los ordenadores en sí. Las dos últimas son de carácter cuantitativo y servirán, principalmente, para desarrollar el proceso de asignación. Se entrará en detalle sobre la información empleada cuando se aborden los distintos algoritmos de fragmentación y asignación

FRAGMENTACIÓN VERTICAL

Introducción

Recuérdese que la fragmentación vertical de una relación *R* produce una serie de fragmentos *R1*, *R2*,...,*Rr*, cada uno de los cuales contiene un subconjunto de los atributos de *R* así como la clave primaria de *R*. El objetivo de la fragmentación vertical consiste en dividir la relación en un conjunto de relaciones más pequeñas tal que algunas de las aplicaciones de usuario sólo hagan uso de un fragmento. Sobre este marco, una fragmentación óptima es aquella que produce un esquema de división que minimiza el tiempo de ejecución de las aplicaciones que emplean esos fragmentos.

La partición vertical resulta más complicada que la horizontal. Esto se debe al aumento del número total de alternativas que tenemos disponibles. Por ejemplo, en la partición horizontal, si el número total de predicados simples de *Pr* es *n*, existen 2*n* predicados mintérminos posibles que puedan definirse. Además, sabemos que algunos de estos predicados resultarán contradictorios con algunas de las aplicaciones existentes, por lo que podremos reducir el número inicial. En el caso vertical, si una relación tiene *m* atributos clave no primarios, el número de posibles fragmentos es igual a *B*(*m*), es decir el m-ésimo número de Bell [3]. Para valores grandes de *m*, $B(m) \approx m^m$; por ejemplo, para $m = 10$, $B(m) \approx 115.000$, para $m = 15$, $B(m) \approx 109$, para $m = 30$, $B(m) = 1023$.

Estos valores indican que la obtención de una solución óptima de la fragmentación vertical resultará una tarea inútil, sino nos apoyamos en el uso de heurísticos. Existen dos enfoques heurísticos para la fragmentación vertical de relaciones:

1. Agrupación. Comienza asignando cada atributo a un fragmento, y en cada paso, junta algunos de los fragmentos hasta que satisface un determinado criterio. La agrupación sugirió en principio para bases de datos centralizadas y se usó posteriormente para las bases de datos distribuidas.
2. Escisión. A partir de la relación se deciden que fragmentos resultan mejores, basándose en las características de acceso de las aplicaciones a los atributos. Esta técnica se presentó, también, para bases de datos centralizadas. Posteriormente, se extendió al entorno distribuido.

En este documento se tratará únicamente la técnica de escisión, ya que es más apropiada para la estrategia descendente y porque resulta más probable encontrar la solución para la relación entera que a partir de un conjunto de fragmentos con un único atributo. Además, la escisión genera fragmentos no solapados mientras que la agrupación normalmente produce fragmentos solapados. Dentro del contexto de los sistemas de bases de datos distribuidos, son preferibles los fragmentos no solapados por razones obvias. Evidentemente, los fragmentos no solapados se refieren únicamente a atributos clave no primarios.

Antes de comenzar, vamos a aclarar un problema: la réplica de las claves de la relación en los fragmentos. Esta es una característica de la fragmentación vertical que permite la reconstrucción de la relación global. Por tanto, la escisión considera únicamente aquellos atributos que no son parte de la clave primaria.

La réplica de los atributos clave supone una gran ventaja, a pesar de los problemas que pueda causar. La ventaja está relacionada con el esfuerzo para mantener la integridad semántica. Tenga en cuenta que cada dependencia (funcional, multivaluada ...) es, de hecho, una restricción que influye sobre el valor de los atributos de las respectivas relaciones en todo momento. También muchas de estas dependencias implican a los atributos clave de una relación.

Si queremos diseñar una base de datos tal que los atributos clave sean parte de una fragmento que está ubicado en un sitio, y los atributos relacionados sean parte de otro fragmento asignado a un segundo sitio, cada petición de actualización provocará la verificación de integridad que necesitará de una comunicación entre esos sitios. La réplica de los atributos clave de cada fragmento reduce esta problemática, pero no elimina toda su complejidad, ya que la comunicación puede ser necesaria para las restricciones de integridad que implican a las claves primarias, así como para el control de concurrencia.

Una posible alternativa a la réplica de los atributos clave es el empleo de identificadores de tuplas, que son valores únicos asignados por el sistema a las tuplas de una relación. Mientras el sistema mantenga los identificadores, los fragmentos permanecerán disjuntos.

Información necesaria para la fragmentación vertical.

La principal información que necesitaremos se referirá a las aplicaciones. Por tanto, este punto tratará de especificar la información que de una aplicación que funciona sobre la base de datos podamos extraer. Teniendo en cuenta que la fragmentación vertical coloca en un fragmento aquellos atributos a los que se accede de manera simultánea, necesitaremos alguna medida que defina con más precisión el concepto de simultaneidad. Esta medida es la afinidad de los atributos, que indica la relación estrecha existente entre los atributos. Desgraciadamente, no es muy realista esperar que el diseñador o los usuarios puedan especificar estos valores. Por ello, presentaremos una forma por la cual obtengamos esos valores partiendo de datos más básicos.

El principal dato necesario relativo a las aplicaciones es la frecuencia de acceso. Sea $Q = \{q1, q2, ..., qq\}$ el conjunto de consultas de usuario (aplicaciones) que funcionan sobre una relación $R(A1, A2, ..., An)$. Entonces, para cada consulta qi y cada atributo Aj, asociaremos un valor de uso de atributos, representado por $uso(qi, Aj)$ y definido como sigue:

uso(*qi, Aj*) = 1 si la consulta *qi* hace referencia a *Aj*
uso(*qi, Aj*) = 0 en cualquier otro caso

Los vectores uso(qi,□) pueden definirse muy fácilmente para cada aplicación siempre que el diseñador conozca las aplicaciones existentes en el sistema. La regla 80/20 expuesta páginas atrás podría resultar útil para el desarrollo de esta tarea.

Ejemplo 10. Sea la relación CLIENTES *de la base de datos. Asumamos que las siguientes aplicaciones van a hacer uso de ella. Se indica, también, la especificación SQL.*

q1 : *Encuéntrese el nombre y la dirección del cliente a partir de su código.*

SELECT CNOMCLI, CDIRCLI FROM CLIENTES WHERE CCODCLI = *Valor*

q2 : *Encuéntrese el nombre y DNI de todos los clientes residentes en una determinada provincia.*

SELECT CNOMCLI, CDNICIF FROM CLIENTES WHERE NCODPROV = *Valor*

q3 : *Encuéntrese el nombre de todos los clientes.*

SELECT CNOMCLI FROM CLIENTES

q4 : *Encuéntrese la población y el código postal de un cliente a partir de su número.*

SELECT CPOBCLI, CPTLCLI FROM CLIENTES WHERE CLIENTES = *Valor*

De acuerdo a estas cuatro aplicaciones, se pueden especificar los valores de uso de los atributos. Como convenio de notación, denominaremos A1 = CLIENTES, *A2* = CCODCLI, *A3* = CNOMCLI, *A4* = CDIRCLI, *A5* = CPOBCLI, *A6* = NCODPROV, *A7* = CPTLCLI *y A8* = CDNICIF. *Los valores de uso se definen en la matriz de uso de atributos siguiente. Cada entrada* (*i, j*) *corresponde a uso*(*qi, Aj*).

	A_1	**A_2**	**A_3**	**A_4**	**A_5**	**A_6**	**A_7**	**A_8**
q_1	0	1	1	1	0	0	0	0
q_2	0	0	1	0	0	1	0	1
q_3	0	0	1	0	0	0	0	0
q_4	1	0	0	0	1	0	1	0

Los valores del uso de los atributos en general no son suficientes para desarrollar la base de la escisión y la fragmentación de los atributos, ya que estos valores no representan el peso de las frecuencias de la aplicación. La dimensión de esta frecuencia puede incluirse en la definición de la medida de los atributos afines *afd*(*Ai, Aj*), la cual mide el límite entre dos atributos de una relación de acuerdo a cómo las aplicaciones acceden a ellos.

La medida de los atributos afines entre los dos atributos *Ai* y *Aj* de una relación *R*(*A1, A2, ..., An*) con respecto a un conjunto de aplicaciones *Q* = {*q1, q2, ..., qq*} se define como

$$adf(A_i, A_j) = \sum_{K|uso(q_k, A_i)=1 \wedge usi(q_k, A_j)=1} \sum_{\forall S_j} ref_j(q_k) acc_j(q_k)$$

donde *refl*(*qk*) es el número de accesos a los atributos (*Ai, Aj*) por cada ejecución de la aplicación *qk* en el sitio *Sl* y *acc*(*qk*) es la medida de la frecuencia de acceso de una aplicación definida anteriormente, pero modificada para incluir las frecuencias en los distintos sitios.

El resultado de este cálculo es una matriz de *n x n*, donde cada elemento es una de las medidas definidas antes. Denominaremos a esta matriz la matriz de atributos afines o *MAA*. Dicha matriz se empleará en el resto de este punto para mostrar el proceso de fragmentación. El proceso implica una agrupación de los atributos con alta afinidad, para luego realizar la escisión de los mismos.

Ejemplo 11. Continuemos con el ejemplo 10. Para no complicar demasiado las cosas, asumiremos que refl(*qk*) *= 1 para todo qk y Sl.. Así mismo, supondremos que en nuestra red existen tres sitios. Si las frecuencias de acceso de las aplicaciones a los atributos son las siguientes*

acc1(q1)=15	acc2(q1)=20	acc3(q1)=10
acc1(q2)=5	acc2(q2)=0	acc3(q2)=0
acc1(q3)=25	acc2(q3)=25	acc3(q3)=25
acc1(q4)=3	acc2(q4)=0	acc3(q4)=0

entonces la medida de afinidad entre los atributos A6 y A8 puede medirse como

$$adf(A_6, A_8) = \sum_{k=2}^{2} \sum_{j=1}^{3} acc_j(q_k) \Rightarrow$$

$$adf(A_6, A_8) = acc_1(q_2) + acc_2(q_2) + acc_3(q_2) = 5$$

ya que la única aplicación que accede a ambos atributos es q2. A continuación se muestran todos los resultados en la matriz de atributos afines. Advierta que los valores de la diagonal también se calculan, pese a que el par de atributos sea el mismo y, por tanto, su valor irrelevante.

	A_1	**A_2**	**A_3**	**A_4**	**A_5**	**A_6**	**A_7**	**A_8**
A_1	3	0	0	0	3	0	3	0
A_2	0	45	45	45	0	0	0	0
A_3	0	45	125	45	0	5	0	5
A_4	0	45	45	45	0	0	0	0
A_5	3	0	0	0	3	0	3	0
A_6	0	0	5	0	0	5	0	5
A_7	3	0	0	0	3	0	3	0
A_8	0	0	5	0	0	5	0	5

Algoritmo de agrupación.

No confunda la agrupación de atributos con la técnica heurística del mismo nombre presentada al principio, no tiene nada que ver. La tarea fundamental consiste en diseñar un algoritmo de fragmentación vertical que encuentre un término medio de agrupación de los atributos de una relación basándose en los valores de afinidad de atributos contenidos en *MAA*. Se ha sugerido por parte de muchos autores [4][5][6] que el algoritmo de energía límite o *BEA*[1] resulta muy apropiado para tal propósito. Se considera adecuado por las siguientes razones:

1. Se diseñó específicamente para determinar grupos de elementos similares frente a una ordenación lineal de los elementos (es decir, grupos de atributos con gran afinidad frente a grupos de atributos con valores pequeños de la misma).
2. Los grupos resultantes no eran sensibles al orden en el cual los elementos se dispusiesen por el algoritmo.
3. El tiempo de cálculo del algoritmo es razonable, $O(n^2)$, donde n es el número de atributos.
4. La interrelación secundaria entre grupos de atributos es identificable.

El algoritmo de energía límite toma como entrada la matriz de atributos afines, permuta filas y columnas y genera una matriz de grupos afines (*MGA*). La permutación se hace de tal manera, que se maximice la siguiente medida de afinidad global (*AG*):

$$AG = \sum_{i=1}^{n} \sum_{j=1}^{n} afd(A_i, A_j) CV$$

donde *CV* son los cuatro vecinos de un elemento de la matriz, es decir

$$CV = afd(A_i, A_{j-1}) + afd(A_i, A_{j+1}) + afd(A_{i-1}, A_j) + afd(A_{i+1}, A_j)$$

A su vez,

$$afd(A_0, A_j) = afd(A_i, A_0) = afd(A_{n+1}, A_j) = afd(A_{i+1}, A_{n+1}) = 0$$

El último conjunto de condiciones toma en consideración los casos en los que los atributos se sitúan en la *MGA* a la izquierda del atributo extremo izquierdo o a la derecha del atributo extremo derecho durante la permutación de columnas, e igualmente respecto a la fila que se sitúa en la parte superior durante la permutación de filas. En estos casos, tomaremos el cero como valor de afinidad entre los atributos considerados para su ubicación y sus vecinos izquierdos o derechos (superiores o inferiores), que no existen en la *MGA*.

La función de maximización considera sólo los vecinos más próximos, resultando en el grupo de valores grandes solo estos, y en el grupo de valores pequeños solo éstos. Además, la matriz de atributos afines es simétrica, lo cual reduce la función objetiva de la formulación a

El algoritmo se presenta a continuación. Para la generación de la matriz de grupos afines se siguen tres pasos:

1. Iniciación. Sitúa y fija una de las columnas de *MAA* arbitrariamente dentro de *MGA*. En el algoritmo se escoge la columna 1.
2. Iteración. Se toma cada una de las columnas restantes *n - i* (donde *i* es el número de columnas que ya se han situado en *MGA*) y se intenta situarlas en las *n - i* posiciones restantes de la matriz *MGA*. Se escoge como lugar de emplazamiento aquel que proporciones una mayor contribución a la medida de afinidad global descrita anteriormente. Se continúa con este paso hasta que se agote el número de columnas a situar.
3. Ordenación de las filas. Una vez que la ordenación de las columnas ha finalizado, se debe proceder a ordenar las filas de tal modo que su posición relativa coincida con la de las columnas. Por ejemplo, si la columna 3 se ha situado en la primera posición, la fila número 3 también debería pasar a ocupar la primera posición.

```
Algoritmo BEA Algoritmo BEA
Entrada:
        MAA: matriz de atributos afines
Salida:
        MGA: matriz de grupos afines
Inicio
{iniciación; recuerde que MAA es una matriz de n x n
        MGA(*, 1)←MAA(*, 1)
        MGA(*, 2)←MAA(*, 2)
        índice←3
        mientras índice ≤ n hacer
        {escoger la mejor ubicación para el atributo MAA_índice}
        inicio
                para i=1 hasta índice-1 por 1 hacer
                        calcular cont(Ai-1), Aíndice, Ai
                fin-para
                calcular cont(Aíndice-1,Aíndice, Aíndice-1)
                {condición de inmovilidad}
                pos←ubicación dada por el máximo valor de cont
                para j=índice hasta pos por –1 hacer
                        {cambiar posición de las columnas}
                        MGA(*, j) ←MGA(*, j-1)
                fin-para
                MGA(*, pos) ←MAA(*, índice)
                índice←índice+1
        fin-mientras
        ordenar las filas según la ordenación
fin.
```

Para la segunda parte del algoritmo, necesitaríamos definir que significa la contribución de un atributo a la medida de afinidad. Esta contribución puede derivarse como se expondrá ahora. Partiremos de la definición dada de la medida de afinidad global que se escribió como

$$AG = \sum_{i=1}^{n} \sum_{j=1}^{n} adf(A_i, A_j)[afd(A_i, A_{j-1}) + afd(A_i, A_{j+1})]$$

la cual puede rescribirse como

$$AG = \sum_{i=1}^{n} \sum_{j=1}^{n} [adf(A_i, A_j)afd(A_i, A_{j-1}) + afd(A_i, A_j)afd(A_i, A_{j+1})] \Rightarrow$$

$$AG = \sum_{i=1}^{n} \left[\sum_{j=1}^{n} adf(A_i, A_j)afd(A_i, A_{j-1}) + \sum_{i=1}^{n} afd(A_i, A_j)afd(A_i, A_{j+1}) \right]$$

Definamos el límite lím entre dos atributos *Ax* y *Ay* como

$$\lim(A_x, A_y) = \sum_{z=1}^{n} afd(A_z, A_x)afd(A_z, A_y)$$

Entonces podemos escribir *AG* como

$$AG = \sum_{j=1}^{n} [\lim(A_j, A_{j-1}) + \lim(A_j, A_{j+1})]$$

Consideremos ahora los siguientes *n* atributos

$$\underbrace{A_1 A_2 \ldots A_{i-1} A_i}_{AG'} \underbrace{A_j A_{j+1} \ldots A_n}_{AG''}$$

La medida de afinidad global para estos atributos puede escribirse como

$$\begin{aligned} AG_{ant} = AG' + AG'' + \lim(A_i, A_j) + \lim(A_i, A_j) = \\ \sum_{i=1}^{n} [\lim(A_i, A_{i-1}) + \lim(A_i, A_{i+1})] + \\ + \sum_{i=i+2}^{n} [\lim(A_i, A_{i-1}) + \lim(A_i, A_{i+1})] \\ + 2\lim(A_i, A_j) \end{aligned}$$

Consideremos ahora que entre los atributos *Ai* y *Aj* de la matriz de grupos afines se sitúa un nuevo atributo *Ak*. La nueva medida de afinidad global sería entonces

$$\begin{aligned} AG_{nueva} &= AG' + AG'' + lim(A_i, A_k) + lim(A_k, A_i) \\ &\quad + lim(A_k, A_j) + lim(A_j, A_k) \\ &= AG' + AG'' + 2lim(A_i, A_k) + 2lim(A_k, A_j) \end{aligned}$$

Por tanto, la contribución a la red de la medida de afinidad global al situar el atributo *Ak* entre *Ai* y *Aj* es

$$\begin{aligned} &cont(A_i, A_k, A_j) = AG_{nueva} - AG_{ant} \Rightarrow \\ &cont(A_i, A_k, A_j) = 2lim(A_i, A_k) + 2lim(A_k, A_j) - 2lim(A_i, A_j) \end{aligned}$$

Ejemplo 12. Consideremos la matriz MAA que se desarrolló anteriormente para la relación *CLIENTES*. Estudiemos la contribución que se realiza al colocar el atributo A4 entre los atributos A1 y A2.

$$cont(A1, A4, A2) = 2lím(A1, A4) + 2lím(A4, A2) - 2lím(A1, A2)$$

Haciendo los cálculos para cada término, se obtiene

lím(*A1, A4*) = 3·0 + 0·45 + 0·45 + 0·45 + 3·0 + 0·0 + 3·0 + 0·0 = 0 *lím*(*A4, A2*) = 2025 *lím*(*A1, A2*) = 0

Por lo tanto, cont*(*A1, A4, A2*) = 2·0 + 2·2025 – 2·0 = 4050*

El algoritmo se concentra en las columnas de la matriz de atributos afines. Podemos emplear los mismos argumentos y rediseñarlo de tal manera que opere sobre las filas también. Otro punto importante de este algoritmo es que mejora la eficiencia, la segunda columna también se fija y se sitúa tras la primera durante el proceso de iniciación. Esto es perfectamente válido, ya que *A2* puede situarse a la derecha o a la izquierda de *A1*. El límite entre las dos, sin embargo, es independiente de las posiciones relativas que tengan la una sobre la otra. Finalmente, deberíamos indicar el problema de calcular *cont* en los extremos. Si un atributo *Ai* se debe situar a la izquierda del atributo más a la izquierda, una de las ecuaciones del límite se calculará sobre un elemento inexistente, el de la izquierda, y sobre *Ak*. Entonces, necesitamos referirnos a las condiciones impuestas en la definición de la medida de afinidad global, donde *MGA*(*0, k*) = 0. El caso contrario se produce cuando *Aj* es el atributo situado más a la derecha, ubicado ya en la matriz *MGA*, y queremos saber cuál es la contribución de situar el atributo *Ak* a la derecha de *Aj*. Ante tal situación se debe calcular el *lím*(*k, k+1*). Ya que no existe un atributo situado todavía en la columna *k+1* de la *MGA*, la medida de afinidad no puede establecerse. Por tanto, de acuerdo a las condiciones de los extremos, el valor del límite es también 0.

Ejemplo 13. Volviendo a la matriz de grupos afines de la relación *CLIENTES*, vamos a desarrollar los pasos del algoritmo para decidir en qué posición se ha de colocar, por ejemplo, el atributo A3. Como ya se ha comentado el algoritmo deja fijas en un principio las dos primeras columnas A1 y A2. Por tanto, las posibles ubicaciones para el atributo A3 serían: a la izquierda de A1 (ordenación 0-3-1), entre A1 y A2 (ordenación 1-3-2) y a la derecha de A2 (ordenación 2-3-4). Tenga en cuenta dos cosas. Primero, ya se ha citado que el límite de cualquier atributo respecto al extremo, A0 en este caso, es siempre cero. Segundo, para la ordenación 2-3-4, el atributo A4 en la matriz MGA todavía no se ha colocado, por tanto no existe y su límite con otro atributo también es cero. Aclarados estos dos puntos, se realizan los siguientes cálculos:

Ordenación (0-3-1)

cont*(*A0, A3, A1*) = 2*lím*(*A0, A3*) + 2*lím*(*A3, A1*) – 2*lím*(*A0, A1*)lím*(*A0, A3*) = 0* lím(A0, A1*) = 0*lím(A3, A1*) = 0·3 + 45·0 + 125·0 + 45·0 + 0·3 + 5·0 + 0·3 + 5·0 = 0* cont(A0, A3, A1*) = 0*

Ordenación (1-3-2)

cont*(*A1, A3, A2*) = 2*lím*(*A1, A3*) + 2*lím*(*A3, A2*) – 2*lím*(*A1, A2*) lím*(*A1, A3*) =* lím*(*A3, A1*) = 0*lím*(*A3, A2*) = 9630*lím*(*A1, A2*) = 0*cont*(*A1, A3, A2*) = 19260*

Ordenación (2-3-4). Esta ordenación representa la condición de inmovilidad, ya que A3 ocuparía su posición natural.

cont*(*A2, A3, A4*)* = *2*lím*(*A2, A3*)* + *2*lím*(*A3, A4*)* – *2*lím*(*A2, A4*)* lím*(*A2, A3*)* = lím*(*A3, A2*)* = *9630* lím*(*A3, A4*)* = *0* lím*(*A2, A4*)* = *0* cont*(*A2, A3, A4*)* = *19260*

Por los resultados obtenidos habrá podido observar que la máxima contribución la aportan las ordenaciones (1- 3-2) y (2-3-4). Además, dicha contribución es idéntica. En este caso, se deja el criterio en manos del lector. En nuestro caso la condición de inmovilidad se prefiere hacer prevalecer sobre el resto, por tanto el atributo A3 se situaría en MGA en la posición 3, la misma posición que ocupaba en la matriz MAA.

Completando el proceso para el resto de los atributos se obtienen los resultados que se muestran a continuación:

a. Como se ha expuesto, A3 permanece en la posición número 3. Lo mismo sucede con el atributo A4.

	A_1	**A_2**	**A_3**	**A_4**
A_1	3	0	0	0
A_2	0	45	45	45
A_3	0	45	125	45
A_4	0	45	45	45
A_5	3	0	0	0
A_6	0	0	5	0
A_7	3	0	0	0
A_8	0	0	5	0

b. El atributo A5 pasa a la posición número 1.

	A_5	**A_1**	**A_2**	**A_3**	**A_4**
A_1	3	3	0	0	0
A_2	0	0	45	45	45
A_3	0	0	45	125	45
A_4	0	0	45	45	45
A_5	3	3	0	0	0
A_6	0	0	0	5	0
A_7	3	3	0	0	0
A_8	0	0	0	5	0

c. La mayor contribución para el atributo A6 se da en su posición natural.

	A_5	**A_1**	**A_2**	**A_3**	**A_4**	**A_6**
A_1	3	3	0	0	0	0
A_2	0	0	45	45	45	0
A_3	0	0	45	125	45	5
A_4	0	0	45	45	45	0
A_5	3	3	0	0	0	0
A_6	0	0	0	5	0	5
A_7	3	3	0	0	0	0
A_8	0	0	0	5	0	5

d. El atributo A7 se desplaza hasta la primera posición.

	A_7	**A_5**	**A_1**	**A_2**	**A_3**	**A_4**	**A_6**
A_1	3	3	3	0	0	0	0
A_2	0	0	0	45	45	45	0
A_3	0	0	0	45	125	45	5
A_4	0	0	0	45	45	45	0
A_5	3	3	3	0	0	0	0
A_6	0	0	0	0	5	0	5
A_7	3	3	3	0	0	0	0
A_8	0	0	0	0	5	0	5

e. *La mayor contribución de A8 se da entre los atributos A3 y A4.*

	A_7	**A_5**	**A_1**	**A_2**	**A_3**	**A_8**	**A_4**	**A_6**
A_1	3	3	3	0	0	0	0	0
A_2	0	0	0	45	45	0	45	0
A_3	0	0	0	45	125	5	45	5
A_4	0	0	0	45	45	0	45	0
A_5	3	3	3	0	0	0	0	0
A_6	0	0	0	0	5	5	0	5
A_7	3	3	3	0	0	0	0	0
A_8	0	0	0	0	5	5	0	5

f. *Por último, se ordenan las filas según la posición relativa de las columnas.*

	A_7	**A_5**	**A_1**	**A_2**	**A_3**	**A_8**	**A_4**	**A_6**
A_7	3	3	3	0	0	0	0	0
A_5	3	3	3	0	0	0	0	0
A_1	3	3	3	0	0	0	0	0
A_2	0	0	0	45	45	0	45	0
A_3	0	0	0	45	125	5	45	5
A_8	0	0	0	0	5	5	0	5
A_4	0	0	0	45	45	0	45	0
A_6	0	0	0	0	5	5	0	5

Algoritmo de partición.

El objetivo de la escisión consiste en encontrar conjuntos de atributos a los que se acceda conjuntamente, o a un gran número de ellos, por los distintos grupos de aplicaciones. Por ejemplo, si es posible identificar dos atributos, *A1* y *A2*, a los que accede únicamente la aplicación *q1*, y unos atributos *A3* y *A4*, a los que acceden dos aplicaciones *q2* y *q3*, sería muy sencillo decidir qué fragmentos establecer. Por ello, es necesario encontrar un algoritmo que identifique estos grupos.

Considere la matriz de grupos afines de la figura 4. Si se fija un punto a lo largo de la diagonal, se pueden establecer dos conjuntos de atributos. Un conjunto {*A1, A2, ..., Ai*} situado en la esquina superior izquierda y un segundo conjunto {*Ai+1, ..., An*} en la parte inferior derecha de este punto. Denominaremos al primero de ellos el conjunto superior y al segundo el conjunto inferior y notaremos sus conjuntos de atributos como *AS* y *AI*, respectivamente.

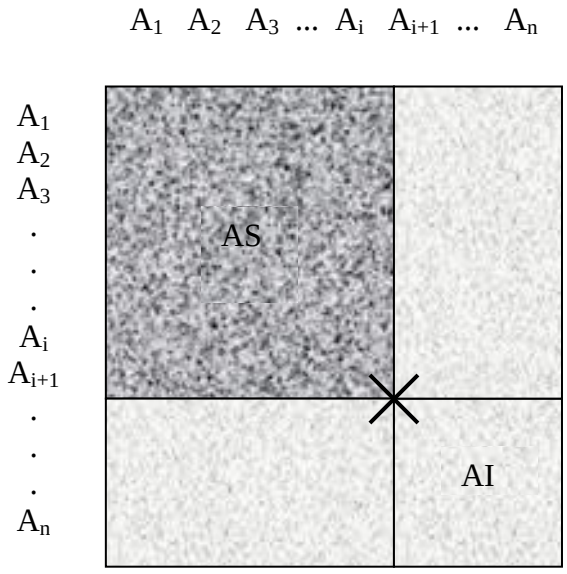

Figura 4. Escisión de la matriz.

Detengámonos ahora en el conjunto de aplicaciones *Q* = {*q1, q2, ..., qq*} y definamos los conjuntos de aplicaciones que acceden solamente a *AS* o a *AI*, o a ambas. Estos conjuntos se definen como sigue:

$$AQ(q_i) = \{A_j \mid uso(q_i, A_j) = 1$$
$$SQ = \{q_i \mid AQ(q_i) \subseteq AS\}$$
$$IQ = \{q_i \mid AQ(q_i) \subseteq AI\}$$
$$XQ = Q - \{SQ \cup IQ\}$$

La primera de estas ecuaciones define el conjunto de atributos a los que accede la aplicación *qi*; *SQ* e *IQ* son el conjunto de aplicaciones que sólo acceden a los conjuntos *AS* y *AI*, respectivamente, y *XQ* es el conjunto de aplicaciones que accede a ambos.

Nos encontramos ante un problema de optimización. Si hay n atributos en una relación, existen *n - 1* posibles posiciones en las que situar el punto de división a lo largo de la diagonal, para producir los conjuntos *SQ* e *IQ*, tal que el acceso total a un único fragmento se maximice mientras que el acceso por las aplicaciones a varios fragmentos se minimice. Definiremos, por ello, las siguientes ecuaciones de coste:

$$CQ = \sum_{q_i \in Q} \sum_{\forall S_j} ref_j(q_i) acc_j(q_i)$$
$$CSQ = \sum_{q_i \in SQ} \sum_{\forall S_j} ref_j(q_i) acc_j(q_i)$$
$$CIQ = \sum_{q_i \in IQ} \sum_{\forall S_j} ref_j(q_i) acc_j(q_i)$$
$$CXQ = \sum_{q_i \in XQ} \sum_{\forall S_j} ref_j(q_i) acc_j(q_i)$$

Cada una de las ecuaciones cuenta el número total de accesos a los atributos por parte de cada aplicación para sus respectivas clases. Basándonos en esta medida, el problema de optimización se define encontrando el punto *x* ($1 \le x \le n$) tal que la expresión siguiente sea máxima:

$$Z = CSQ * CIQ - CXQ$$

La característica fundamental de esta expresión es que define dos fragmentos tales que los valores de *CSQ* y *CIQ* son tan próximos como sea posible. Esto permite el equilibrio de la carga de procesamiento cuando los fragmentos estén distribuidos entre varios sitios. Está claro que el algoritmo de partición tiene complejidad lineal con respecto al número de atributos de la relación, que es $O(n)$.

Existen dos complicaciones que necesitan apuntarse. La primera es relativa a la escisión. Como el lector habrá podido percatarse, el procedimiento de escisión únicamente divide el conjunto de atributos de dos formas: generando el conjunto *AS* y el conjunto *AI*. En el caso que el conjunto de atributos sea grande, seguramente será necesario un método de partición que soporte n-formas de división. El diseño de este algoritmo es posible, pero computacionalmente costoso. A lo largo de la diagonal de la matriz *MGA* es necesario probar 1, 2, ..., *n - 1* puntos de división y para cada uno de estos verificar el lugar que maximice a *z*. Por tanto, la complejidad del algoritmo alcanzaría un valor de $O(2^n)$. Sin embargo, puede llevarse a cabo una solución alternativa consistente en una llamada iterativa al algoritmo de fragmentación binaria para cada uno de los fragmentos obtenidos en la iteración previa. Esta solución cuyo coste computacional es de $O(n^3)$ se presenta más adelante.

La segunda complicación se refiere a la localización de un bloque de atributos que debería formar un fragmento. Nuestra discusión no va más allá de considerar que el punto de escisión es único y simple, y que divide a la matriz *MGA* en una partición superior izquierda y en una partición con el resto de los atributos. La partición, sin embargo, podría formarse en el medio de la matriz. En este caso, necesitamos modificar el algoritmo levemente. La columna más a la izquierda de la matriz *MGA* se desplaza para convertirse en la columna más a la derecha y la fila superior se desplaza para pasar a ser la inferior. A la operación de desplazamiento le sigue la verificación de las *n - 1* posiciones de la diagonal para encontrar el máximo de *z*. La idea que esconde el desplazamiento consiste en mover el bloque de atributos que debería formar un grupo a la esquina superior izquierda, donde se identificará más fácilmente. Con el añadido de la operación de desplazamiento, la complejidad del algoritmo de partición se eleva en un factor *n* convirtiéndose en $O(n^2)$.

Asumiendo que el procedimiento de desplazamiento, denominado DESPLAZAR, ya se ha desarrollado, el algoritmo se presenta a continuación. La entrada del algoritmo PARTICIÓN[1] es la matriz de grupos afines y la relación *R* a fragmentar. La salida es el conjunto de fragmentos *FR* = {*R1, R2*}, donde *Ri* $\subseteq${*A1, A2, ..., An*} y *R1*$\cap$*R2* = los atributos clave de *R*. Tenga en cuenta, que para el algoritmo de *n-formas* esta rutina debería invocarse interactivamente, o desarrollarse como procedimiento recursivo.

ALGORITMO DE PARTICIÓN
Entrada:
MAA: matriz de atributos afines
R: relación
Salida:
F: conjunto de fragmentos
Inicio:
{calcular el valor de z para la primera columna}
{los subíndices en las ecuaciones de costo indican el punto de escisión}
calcular CSQ_{n-r}
calcular CIQ_{n-r}
calcular CXQ_{n-r}
mejor$\leftarrow CSQ_{n-r} * CIQ_{n-r} - (CXQ_{n-r})^2$
hacer *{obtención de la mejor partición*
inicio
para *i=n-2* ***hasta*** *l por –1* ***hacer***
inicio
calcular CSQ_{n-r}
calcular CIQ_{n-r}
calcular CXQ_{n-r}
mejor$\leftarrow CSQ_{n-r} * CIQ_{n-r} - (CXQ_{n-r})^2$
si *z>mejor* ***entonces***
inicio
mejor$\leftarrow z$
guardar la posición del desplazamiento
fin-si
fin-para
llamar DESPLAZAR (MAA)
fin-inicio
hasta *que MAA no pueda desplazar mas*
reconstruir la matriz según la posición de desplazamiento;
Rr$\leftarrow \Pi$*AS(R)UK*
R2$\leftarrow \Pi$*Ar(R)UK*
{K es el conjunto de claves primarias de los atributos de R}
F$\leftarrow$*(Rr, R2)*
Fin. *{PARTICION}*

Ejemplo 14. Al aplicar el algoritmo PARTICIÓN *sobre la matriz MGA obtenida para la relación* CLIENTES *se obtiene el grupo de fragmentos FCLIENTES* = {CLIENTES1, CLIENTES2} *donde* CLIENTES1 = {*A7, A5, A1*} *y* CLIENTES2 = {*A1, A2, A3, A8, A4, A6*} *teniendo en cuenta que se considera clave primaria a A1. Por lo tanto,*

CLIENTES1 = {CPTLCLI, CPOBCLI, CLIENTES}
CLIENTES2 = {CLIENTES, CCODCLI, CNOMCLI, CDNICIF, CDIRCLI, NCODPROV}

Fragmentando relaciones grandes.

Se ha citado anteriormente la necesidad existente en muchos casos de fragmentar una relación en más de dos partes, especialmente si ésta es grande, contiene un número considerable de atributos. Existen dos alternativas para desarrollar el algoritmo de n-formas que haga esto posible. Por un lado podríamos modificar el algoritmo de partición para que llevase a cabo esta tarea, lo cual resultaría bastante complicado y, por otra, hacer uso de éste tal cual pero siendo la base del nuevo algoritmo, es decir, lo utilizaríamos iterativamente para fragmentar binariamente cada uno de los fragmentos que se generen en el paso anterior. La opción elegida es esta última.

El algoritmo [7] mostrado a continuación es bastante sencillo, sólo hay que tener en cuenta el número de grupos de fragmentación que se van generando.

```
Algoritmo N-FORMAS
Entrada:
        MGA: matriz de grupos afines
        Q: conjunto de aplicaciones
Salida:
        EEF_n: n esquemas de fragmentación
Inicio
        {i vale 1 en la primera llamada al algoritmo}
        i←i+1
        si el número de columnas de MGA>1
        si el numero de columnas de MGA>1 entonces
        inicio
                llamar PARTICIÓN
                TF←FS∪FI
                {TF es le conjunto de todos lo s fragmentos. FS es el fragmento
                superior generado pro PARTICIÓN y FI el inferior}
                EEF←EEF – {TF}
                EEF←EEF∪{FS}∪{FI}
                EEF_i←EEF
                NGF_i←{FS}∪{FI}
                {cada NGFi es un conjunto de dos nuevos grupos de fragmentos
                obtenidos a partir del conjunto de fragmentos existentes en MGA}
                llamar N-FORMAS(MGA_s,SQ)
                llamar N-FORMAS(MGA_r,IQ)
                {MGA_s y MGA_r son los atributos de MGA que después de la escisión
                quedan en la parte superior  e inferior, respectivamente}
        fin-si
fin. {N-FORMAS}
```

Evidentemente el coste computacional de todo el proceso aumenta. Partiendo de que el coste del algoritmo PARTICIÓN es de $O(n^2)$, el coste añadido es el siguiente. En los pasos 2, 4, 5 y 6 es de $O(1)$. En el paso 3 se necesita buscar el esquema de grupo de atributos actual para *TF*, es decir, el conjunto de fragmentos que representa la actual *MGA*. En otras palabras, $TF = FS \cup FI$. Esta búsqueda puede llevar consigo como mucho n iteraciones dado que el tamaño del grupo de fragmentos actual no puede ser mayor que *n*, es decir, cada grupo en el grupo de fragmentos actual sólo puede contener un fragmento en él. Los pasos 7 y 8 se ejecutan como máximo *n* veces. Por todo ello, la complejidad del algoritmo n-formas se incrementa en un factor n a partir de la complejidad temporal de PARTICIÓN, entonces el coste total alcanza un valor de $O(n^3)$.

Ejemplo 15. Al emplear el algoritmo N-FORMAS *sobre la relación clientes, se obtienen los siguientes esquemas de fragmentación . Observe como en cada paso se produce un fragmento más respecto al inmediato anterior. Debido a esta característica se dice que los esquemas de fragmentación generados son jerárquicos.*

Esquema	Fragmentos	Esquema de fragmentación
1	1	(12345678)
2	2	(751) (23846)
3	3	(751) (234) (86)
4	4	(751) (23) (4) (86)
5	5	(7) (51) (23) (4) (86)
6	6	(7) (5) (1) (23) (4) (86)
7	7	(7) (5) (1) (23) (4) (8) (6)
8	8	(7) (5) (1) (2) (3) (4) (8) (6)

Dado que el método *n-formas* para una relación *R* que se desee fragmentar y que consta de m atributos, puede generar desde 2 hasta *m* fragmentos diferentes, es fácil preguntarse cuál de todos los esquemas generados es el mejor. Para responder a esto se va a proceder a estudiar en detalle el Examinador de Particiones (EP) [8], que no es más que una función de coste que determina el mejor esquema. Este punto muy relacionada con la fase de asignación se expone en siguientes apartados. Pero antes de ello, resulta necesario verificar la corrección del algoritmo de partición.

Comprobación de la corrección.

Emplearemos argumentos similares a los de la fragmentación horizontal para probar que el algoritmo de fragmentación vertical es correcto.

1. Compleción. La compleción se garantiza ya que cada atributo de la relación global se asigna a uno de los fragmentos. Independientemente del tamaño del conjunto de atributos A sobre el que se define la relación *R*, este está formado por $A = AS \cup AI$, por lo que la compleción está asegurada.
2. Reconstrucción. Ya se ha mencionado que la reconstrucción de la relación es posible si se hace uso de la operación de yunto. Entonces, para una relación *R* con un conjunto de fragmentos verticales $FR = \{R1, R2, ..., Rr\}$ y el atributo o los atributos clave *K*,

$$R = \rhd\lhd_k R_i, \forall R_i \in F_R$$

Por tanto, dado que cada *Ri* es completo, la operación de yunto podrá reconstruir adecuadamente *R*. Otro punto importante es que cada *Ri* debería contener el o los atributos claves de *R*, o bien, albergar el identificador de tupla asignado por el sistema.

3. Disyunción. La disyunción de los fragmentos no es tan importante en la fragmentación vertical como en la horizontal. Existen dos casos:

    a. Se usan identificadores de tuplas, en este caso los fragmentos son disjuntos ya que los identificadores que se replican en cada fragmento se asignan por el sistema y son totalmente invisibles al usuario.
    b. Los atributos clave se replican en cada fragmento, en este caso no se puede decir que sean disjuntos en un sentido estricto. Sin embargo, es importante que esta duplicación de los atributos clave se conozca y gestione por el sistema. Resumiendo, los fragmentos son disjuntos excepto para los atributos clave

FRAGMENTACION HORIZONTAL

Como se ha explicada anteriormente, la fragmentación horizontal se realiza sobre las tuplas de la relación. Cada fragmento será un subconjunto de las tuplas de la relación. Existen dos variantes de la fragmentación horizontal: la primaria y la derivada. La fragmentación horizontal primaria de una relación se desarrolla empleando los predicados definidos en esa relación. Por el contrario, la fragmentación horizontal derivada consiste en dividir una relación partiendo de los predicados definidos sobre alguna otra.

Información necesaria para la fragmentación horizontal.

Información sobre la base de datos. Esta información implica al esquema conceptual global. Es importante señalar cómo las relaciones de la base de datos se conectan con otras. En una conexión de relaciones normalmente se denomina relación propietaria a aquella situada en la cola del enlace, mientras que se llama relación miembro a la ubicada en la cabecera del vínculo. Dicho de otra forma podemos pensar en relaciones de origen cuando nos refiramos a las propietarias y relaciones destino cuando lo hagamos con las miembro. Definiremos dos funciones: propietaria y miembro, las cuales proyectarán un conjunto de enlaces sobre un conjunto de relaciones. Además, dado un enlace, devolverán el miembro y el propietario de la relación, respectivamente. La información cuantitativa necesaria gira en torno a la cardinalidad de cada relación, notada como *card*(*R*).

Información sobre la aplicación. Necesitaremos tanto información cualitativa como cuantitativa. La información cualitativa guiará la fragmentación, mientras que la cuantitativa se necesitará en los modelos de asignación. La principal información de carácter cualitativo son los predicados empleados en las consultas de usuario. Si no fuese posible investigar todas las aplicaciones para determinar estos predicados, al menos se deberían investigar las más importantes. Podemos pensar en la regla "80/20" para guiarnos en nuestro análisis, esta regla dice que el 20% de las consultas existentes acceden al 80% de los datos. Llegados a este punto, sería interesante determinar los predicados simples. Dada una relación *R*(*A1, A2, ..., An*), donde *Ai* es un atributo definido sobre el dominio *Di*, un predicado simple *pj* definido sobre *R* es de la forma

$$p_i : A_i \theta Valor$$

donde θ es un operador relacional y *Valor* se escoge de entre el dominio de *Ai*. Usaremos *Pri* para notar al conjunto de todos los predicados simples definidos sobre una relación *Ri*. Los miembros de *Pri* se notan por *pij*.

Ejemplo 1. Para la relación PROVINC de la base de datos ejemplo, un par de predicados simples serían los siguientes:

CNOMPROV = "Salamanca" NCODPROV≤ 25

A parte de los predicados simples, las consultas emplean predicados más complejos resultado de combinaciones lógicas de los simples. Una combinación especialmente interesante es la conjunción de predicados simples, al predicado resultante se le denomina predicado mintérmino. Partiendo de que siempre es posible transformar una expresión lógica en su forma normal conjuntiva, usaremos los predicados mintérmino en los algoritmos para no causar ninguna pérdida de generalidad.

Dado un conjunto de predicados simples de una relación *R*, *Pri* = {*pi1, pi2, ..., pim*}, el conjunto de predicados mintérmino *Mi* = {*mi1, mi2, ..., miz*} se define como

$$M_i = \{mi_j = \bigwedge_{pjk \in \Pr j} p'_{jk}\}, 1 \le k \le m, 1 \le j \le z$$

donde *p*ik* = *pik* o *p*ik* = ¬*pik*. Es decir, cada predicado mintérmino puede ser igual a la forma natural o a la forma negada del predicado simple. Es importante señalar que la referencia a la negación de un predicado es significativa para predicados de igualdad de la forma

Atributo = Valor

Para predicados de desigualdad, la negación debería tratarse como su complemento. Por ejemplo, la negación del predicado simple

Atributo≤Valor es *Atributo > Valor*

Pero existen casos en los que el complemento es difícil de definir. Por ejemplo, si dos predicados son de la forma

Límite_inferior ≤Atributo_1
Atributo_1≤Límite_superior

sus complementos son

¬(*Límite_inferior ≤Atributo_1*)
¬ (*Atributo_1 ≤Límite_superior*)

Sin embargo, si los dos predicados simples se escriben como

Límite_inferior ≤Atributo_1 ≤Límite_superior

con un complemento,

¬ (*Límite_inferior ≤Atributo_1 ≤Límite_superior*),

éste no es fácil de definir.

Ejemplo 2. Considere la relación ALBCLIT. Los siguientes predicados son algunos de los posibles predicados simples que pueden definirse sobre la relación:

p1 : CCODCLI = "250104"
p2 : CCODCLI = "250102"
p3 : CCODCLI = "250103"
p4 : CCODCLI = "250108"
p5 : CCODCLI = "250107"
p6 : CCODCLI = "250109"
p7 : NNUMALB ≤980000
p8 : NNUMALB > 980000

Los siguientes son algunos de los predicados mintérmino basados en los predicados simples anteriores:

m1 : CCODCLI = "250104"∧ NNUMALB ≤980000
m2 : CCODCLI = "250104" ∧ NNUMALB > 980000
m3 : ¬ (CCODCLI = "250104") ∧ NNUMALB ≤980000
m4 : ¬ (CCODCLI = "250104") ∧ NNUMALB > 980000
m5 : CCODCLI = "250102" ∧ NNUMALB ≤980000
m6 : CCODCLI = "250102" ∧ NNUMALB > 980000

m7 : ¬ (CCODCLI = "250102") ∧ NNUMALB ≤980000
m8 : ¬ (CCODCLI = "250102") ∧ NNUMALB > 980000

Tenga en cuenta que estos ocho predicados mintérmino no son todos los que se pueden definir, es sólo un ejemplo.

Sobre la información cuantitativa necesaria relativa a las aplicaciones, necesitaremos definir dos conjuntos de datos.

1. Selectividad mintérmino. Es el número de tuplas de una relación a las que accede una consulta de acuerdo a un predicado mintérmino dado. Por ejemplo, en el ejemplo anterior, la selectividad de m*6* es 0 ya que no existe ninguna tupla que satisfaga las condiciones; en cambio, la selectividad de *m1* es 2. Notaremos la selectividad de un mintérmino mi como *sel*(*mi*).
2. Frecuencia de acceso. Es la frecuencia con la que un usuario accede a los datos. Si *Q* = {*q1, q2, ..., qq*} es un conjunto de consultas de usuario, *acc*(*qi*) indica la frecuencia de acceso a la consulta *qi* en un periodo dado.

Observe, que las frecuencias de acceso mintérmino pueden establecerse a partir de las frecuencias de acceso de la consulta. Nos referiremos a las frecuencias de acceso de un mintérmino *mi* como *acc*(*mi*).

Fragmentación horizontal primaria.

Antes de presentar un algoritmo formal que lleve a cabo la fragmentación horizontal, intentaremos explicar de manera intuitiva los procesos de fragmentación horizontal primaria y derivada. La fragmentación horizontal primaria se define como una operación de selección de las relaciones propietarias del esquema de la base de datos. Por tanto, dada una relación *Ri*, sus fragmentos horizontales son

$$R_i^j = \sigma_{F_j}(Rj), 1 \le j \le w$$

donde *Fj* es la fórmula de selección empleada para obtener el fragmento *Rji*. Observe que si *Fj* está en forma normal conjuntiva, es un predicado mintérmino (*mij*). El algoritmo expuesto más adelante, de hecho, necesita que *Fj* sea un predicado mintérmino.

Ejemplo 3. Podríamos descomponer la relación ALBCLIT del ejemplo en dos fragmentos horizontales ALBCLIT1 y ALBCLIT2 definidos de la manera siguiente:

ALBCLIT1 =σ NNUMALB ≤990001 (ALBCLIT)
ALBCLIT2 = σ NNUMALB > 990001 (ALBCLIT)

El ejemplo anterior ilustra uno de los problemas de la fragmentación horizontal. Si el dominio de los atributos implicados en la fórmula de selección son continuos e infinitos, resulta muy difícil definir un conjunto de fórmulas $F = \{F1, F2, ..., Fn\}$ que fragmentasen la relación de forma adecuada. Una posible vía de solución consistiría en definir una serie de rangos, como se ha hecho en el ejemplo. Sin embargo, siempre aparecerá el problema de tratar los dos extremos. Es decir, supongamos que se añade una nueva tupla a ALBCLIT con un valor para el atributo NNUMALB de 20000001, entonces deberíamos revisar el punto de fragmentación para decidir si la nueva tupla se incluye en el segundo fragmento ALBCLIT2 o si se ha de realizar una nueva fragmentación modificando el criterio de selección, tal que

ALBCLIT1 = σ 990001 < NUMALB≥ 10495001 (ALBCLIT)
ALBCLIT2 = σ NNUMALB > 10495001 (ALBCLIT)

En este caso resulta evidente que la mejor opción sería destinar directamente la nueva tupla al segundo fragmentos. Sin embargo, si entrasen 8 nuevas tuplas con un número de albarán (NNUMALB) relativo al año 2.000, es decir, 2000xxxx sería más adecuado elegir otro punto de fragmentación basándonos en la operación de selección.

Este problema, en la práctica, puede resolverse limitando el dominio de los atributos de acuerdo con los requisitos de la aplicación.

Ejemplo 4. Considere la relación PROVINC del ejemplo que venimos empleando. Podríamos definir una serie de fragmentos horizontales basándonos en el código de zona de la siguiente manera.

PROVINC1 = σ CCODZONA = "CYL" (PROVINC)
PROVINC2 = σ CCODZONA = "CLM" (PROVINC)
PROVINC3 = σ CCODZONA = "CAT" (PROVINC)
PROVINC4 = σ CCODZONA = "MAD" (PROVINC)

Cuyo resultado sería el que se muestra a continuación:

PROVINC	NCODPROV	CNOMPROV	CCODZONA
2	37	Salamanca	CYL
3	24	León	CYL
6	49	Zamora	CYL
7	5	Ávila	CYL
PROVINC1			
PROVINC	NCODPROV	CNOMPROV	CCODZONA
1	13	Ciudad Real	CLM
5	45	Toledo	CLM
PROVINC2			
PROVINC	NCODPROV	CNOMPROV	CCODZONA
4	08	Barcelona	CAT
PROVINC3			
PROVINC	NCODPROV	CNOMPROV	CCODZONA
8	28	Madrid	MAD
PROVINC4			

Ahora definiremos la fragmentación horizontal más formalmente. Un fragmento horizontal *Ri* de una relación *R* contiene todas las tuplas de *R* que satisfacen un predicado mintérmino *mi*. Por tanto, dado un conjunto de predicados mintérmino *M*, existen tantos fragmentos horizontales de la relación *R* como predicados mintérmino. Este conjunto de fragmentos horizontales también se conocen como conjuntos de fragmentos mintérmino. En los párrafos siguientes se asumirá que la definición de fragmentos horizontales se basa en los predicados mintérmino. Además, el primer paso para el algoritmo de fragmentación consiste en establecer un conjunto de predicados con ciertas propiedades.

Un aspecto importante de los predicados simples es su compleción, así como su minimalidad. Un conjunto de predicados simples *Pr* se dice que es completo si y solo si existe una probabilidad idéntica de acceder por cada aplicación a cualquier par de tuplas pertenecientes a cualquier fragmento mintérmino que se define de acuerdo con *Pr*. Se puede apreciar como la definición de compleción de un conjunto de predicados simples difiere de la regla de compleción de la fragmentación.

Ejemplo 5. Considere la fragmentación llevada a cabo anteriormente sobre la relación PROVINC. *Si existiese únicamente una aplicación que accediese a las tuplas de acuerdo al código de zona, el conjunto sería completo ya que existe la misma probabilidad de acceder a cada tupla de los fragmentos. Sin embargo, si existe una segunda aplicación que accede a las tuplas que tienen un código provincial menor o igual que* 25, *entonces Pr no es completo. Algunas de las tuplas de un* PROVINCi *tienen mayor probabilidad de acceso por parte de esta segunda aplicación. Para hacer el conjunto de predicados completo, necesitaríamos añadir* (NCODPROV≤ 25, NCODPROV > 25) *a Pr*:

Pr = { CCODZONA = "CLM", CCODZONA = "CYL", CCODZONA = "CAT", CCODZONA = "MAD",NCODPROV ≤ 25, NCODPROV > 25 }

La segunda propiedad, la minimalidad, es muy intuitiva. Es sencillamente un estado que indica el grado de influencia de un predicado en el desarrollo de la fragmentación (es decir, provoca que un fragmento *f* se fragmente en *fi* y *fj*), siempre que al menos una aplicación acceda a *fi* y a *fj* de forma diferente. En otras palabras, el predicado simple debería ser relevante para provocar la fragmentación. Si todos los predicados de un conjunto *Pr* son relevantes, *Pr* es mínimo. Una definición formal de la relevancia podría ser la siguiente. Sean *mi* y *mj* dos predicados mintérmino idénticos en su definición, exceptuando que mi contiene el predicado simple *pi* en su forma natural, mientras que *mj* contiene a $\neg pi$. Sean, además, *fi* y *fj* dos fragmentos definidos de acuerdo a *mi* y *mj*, respectivamente. Entonces *pi* es relevante si y sólo si

$$\frac{acc(m_i)}{card(f_i)} \neq \frac{acc(m_j)}{card(f_i)}$$

Ejemplo 6. Sea el conjunto Pr definido anteriormente. Este conjunto es completo y mínimo. Sin embargo, si le añadimos el predicado CNOMPROV = "Barcelona", *el resultado no sería mínimo ya que el nuevo predicado no es relevante para Pr.*

A continuación se presenta un algoritmo iterativo, llamado COM_MIN [1], que generará un conjunto de predicados completo y mínimo *Pr'* a partir de un conjunto de predicados simples *Pr*. Tenga en cuenta la notación siguiente: *Regla 1*. Es la regla fundamental de la compleción y la minimalidad, que afirma que una relación o fragmento se divide "en al menos dos partes a las cuales se accede de forma diferente por al menos una aplicación." *fi de Pr'*. El fragmento *fi* se define de acuerdo a un predicado mintérmino especificado sobre los predicados de *Pr'*.

```
Algoritmo COM_MIN
entrada:
        R: relación
        Pr: conjunto de predicados simples
salida:
        Pr': conjunto de predicados simples
declarar
        F: conjunto de predicados mintérmino
inicio
        encontrar un p_i ∈ Pr tal que p_i parta a R de acuerdo con la regla 1;
        Pr'←p_i
        Pr ←Pr-p_i
        F←f_i {f_i es el fragmento mintérmino respecto a p_i}
        hacer
                inicio
                        encontrar un p_j ∈ Pr tal que p_j parta a un f_k de Pr' de acuerdo con la
                        regla 1
                        Pr'←Pr'∪p_j
                        Pr←Pr-p_j
                        F←F∪F_j
                        si ∃ p_k∈Pr irrelevante entonces
                        inicio
                                Pr'←Pr'-pk
                                F←F-f_k
                        fin-si
                fin-inicio
        hasta que Pr' este completo
fin. {COM_MIN}
```

El algoritmo empieza encontrando un predicado relevante que parta la relación de entrada. El bucle **hacer – hasta** añade interactivamente predicados al conjunto, asegurando la minimalidad en cada paso. Además, al final, el conjunto *Pr'* es tanto mínimo como completo.

El segundo paso en el proceso de fragmentación primaria consiste en derivar el conjunto de predicados mintérmino que pueden definirse sobre los predicados del conjunto *Pr'*. Estos predicados mintérmino establecen los fragmentos candidatos para el proceso de asignación. El establecimiento de los predicados mintérmino es trivial; la dificultad radica en el tamaño del conjunto de predicados mintérmino, que puede ser muy grande (de hecho, exponencial respecto al número de predicados simples). En el paso siguiente se presentarán formas de reducir el número de predicados mintérmino necesarios para la fragmentación.

El tercer paso aborda, como ya se ha citado, la eliminación de algunos fragmentos mintérmino que puedan ser redundantes. Esta eliminación se desarrolla identificando aquellos mintérminos que puedan resultar contradictorios sobre un conjunto de implicaciones *I*. Por ejemplo, si *Pr'* = {*p1, p2*}, donde

p1 : *atr* = *valor_1*
p2 : *atr* = *valor_2*

y el dominio de *atr* es {*valor_1, valor_2*}, es obvio que *I* contiene las dos implicaciones siguientes

$i1 : (atr = valor\_1) \Rightarrow \neg\ (atr = valor\_2)$
$i2 : \neg\ (atr = valor\_1) \Rightarrow (atr = valor\_2)$

Los siguientes cuatro predicados mintérmino se definen de acuerdo a *Pr'*:

$m1 : (atr = valor\_1) \wedge (atr = valor\_2)$
$m2 : (atr = valor\_1) \wedge \neg (atr = valor\_2)$
$m3 : \neg (atr = valor\_1) \neg (atr = valor\_2)$
$m4 : \neg (atr = valor\_1) \wedge \neg (atr = valor\_2)$

En este caso los predicado mintérmino *m1* y *m4* resultan contradictorios respecto a las implicaciones *I* y pueden eliminarse de *M*.

El algoritmo de fragmentación horizontal primaria se presenta a continuación. La entrada al algoritmo HORIZONTALP[1] es una relación *Ri* sujeta a la fragmentación, y *Pri*, es decir, el conjunto de predicados simples que se establecieron de acuerdo a las aplicaciones definidas sobre la relación *Ri*.

```
Algoritmo HORIZONTALP
entrada:
        Ri: relación
        Pri: conjunto de predicados simples
salida:
        Mi: conjunto de fragmentos mintérmino
inicio
        Pr'←COM_MIN(Ri, Pri)
        determinar el conjunto Mi de predicados mintérmino;
        determinar el conjunto de Ii de implicaciones de pi ∈ Pr'i;
        para cada mi ∈ Mi hacer
                si mi es contradictorio respecto a I entonces
                        Mi←Mi-mi
                fin-si
        fin-para
fin. {HORIZONTALP}
```

Fragmentación horizontal derivada.

Una fragmentación horizontal derivada se define sobre una relación miembro de acuerdo a una operación de selección especificada sobre su propietaria. Se deben dejar claros dos puntos. Primero, el enlace entre las relaciones propietaria y miembro se define como un equi-yunto. Segundo, un equi-yunto puede desarrollarse a través de semiyuntos. Este segundo punto es especialmente importante para nuestros propósitos, ya que deseamos fraccionar una relación miembro según la fragmentación de su propietaria, además es necesario que el fragmento resultante se defina únicamente sobre los atributos de la relación miembro. Por tanto, dado un enlace *L* donde *propietaria*(*L*) = *S* y *miembro*(*L*) = *R*, los fragmentos horizontales derivados de *R* se definen como

$$Ri = R \triangleright\triangleleft Si, 1 \leq i \leq w$$

donde *w* es el número máximo de fragmentos que se definirán sobre *R*, y *Si* = ?*Fi*(*S*), donde *Fi* es la fórmula según la cual se define el fragmento horizontal primario *Si*.

Ejemplo 8. Considere un enlace L1 entre las relaciones PROVINC *y* CLIENTES *de la base de datos que venimos utilizando para los ejemplos, donde propietaria*(*L1*) = PROVINC *y miembro*(*L1*) = CLIENTES. *Supongamos que queremos dividir a nuestro grupo de clientes en función de la comunidad autónoma donde se encuentran situados. Entonces podemos definir cuatro fragmentos de la siguiente forma:*

$$CLIENTES_1 = CLIENTES \triangleright\triangleleft PROVINC_1$$
$$CLIENTES_2 = CLIENTES \triangleright\triangleleft PROVINC_2$$
$$CLIENTES_3 = CLIENTES \triangleright\triangleleft PROVINC_3$$
$$CLIENTES_4 = CLIENTES \triangleright\triangleleft PROVINC_4$$

donde

$$PROVINC_1 = \sigma_{CCODZONA="CYL"}(PROVINC)$$
$$PROVINC_2 = \sigma_{CCODZONA="CLM"}(PROVINC)$$
$$PROVINC_3 = \sigma_{CCODZONA="CAT"}(PROVINC)$$
$$PROVINC_4 = \sigma_{CCODZONA="MAD"}(PROVINC)$$

El resultado de esta fragmentación puede verlo pulsando aquí.

Las tres entradas necesarias para desarrollar la fragmentación horizontal derivada son las siguientes: el conjunto de particiones de la relación propietaria, la relación miembro y el conjunto se predicados resultados de aplicar el semi-yunto entre la propietaria y la miembro. El algoritmo de fragmentación resulta tan trivial que no se ve la necesidad de entrar en detalles.

Existe una posible complicación que necesita nuestro estudio. En un esquema de base de datos, resulta frecuente que existan más de dos enlaces sobre una relación R. En este caso, aparece más de una posibilidad de fragmentación horizontal derivada. La decisión para elegir una u otra se basa en dos criterios: Uno, la fragmentación con mejores características de yunto. Dos, la fragmentación empleada en más aplicaciones.

Discutamos el segundo criterio primero. Resulta sencillo de establecer si tomamos en consideración la frecuencia con la que cada aplicación accede a los datos. Si es posible, deberíamos intentar facilitar el acceso a los usuarios que hagan mayor uso de los datos para, de esta manera, minimizar el impacto total del rendimiento del sistema.

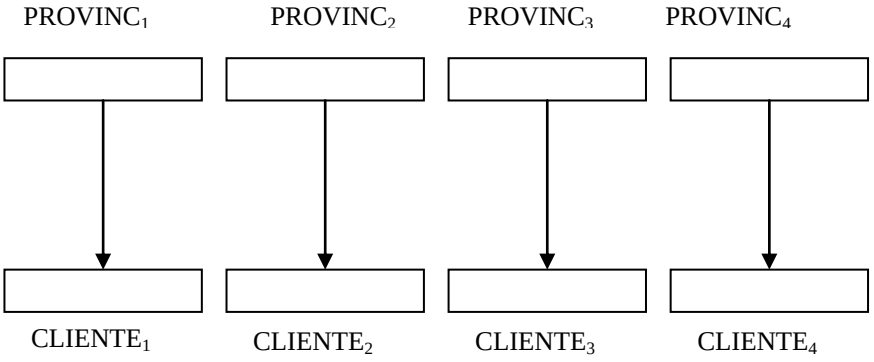

Figura 4. Grafo de yuntos entre fragmentos.

El primer criterio, sin embargo, no es tan sencillo. Considere, por ejemplo, la fragmentación expuesta en el ejemplo 8. El objetivo de esta fragmentación consiste en beneficiar a la consulta que haga uso de las dos relaciones al poder realizarse el yunto de CLIENTES y PROVINC sobre relaciones más pequeñas (es decir, fragmentos), y posibilitar la confección de yuntos de manera distribuida. El primer aspecto resulta obvio. Los fragmentos de CLIENTES son más pequeños que la propia relación CLIENTES. Por tanto, resultará más rápido llevar a cabo el yunto de un fragmento de PROVINC con otro de CLIENTES que trabajar con las propias relaciones. El segundo punto, sin embargo, es más importante ya que es la esencia de las bases de datos distribuidas. Si, además de estar ejecutando un número de consultas en diferentes sitios, podemos ejecutar una consulta en paralelo, se espera que el tiempo de respuesta del sistema aumente. En el caso de yuntos, esto es posible bajo ciertas circunstancias. Considere, por ejemplo, el grafo de yunto (los enlaces) entre los fragmentos de CLIENTES y la derivada PROVINC. Hay únicamente un enlace entrando o saliendo de un fragmento. De ahí, que se denomine a este grafo, grafo simple. La ventaja de este diseño donde la relación de yunto entre los fragmentos es simple, radica en la asignación a un sitio tanto de la propietaria como de la miembro y los yuntos entre pares diferentes de fragmentos pueden realizarse independientemente y en paralelo. Desgraciadamente, la obtención de grafos de yunto simples no siempre es posible. En tal caso, la mejor alternativa sería realizar un diseño que provoque un grafo de yuntos fragmentados. Un grafo fragmentado consiste en dos o más subgrafos que no están enlazados entre ellos. Por tanto, los fragmentos que se obtengan no se distribuirán para ejecuciones paralelas de un modo tan fácil como aquellos obtenidos a través de grafos simples, pero su asignación aún será posible.

Ejemplo 9. Continuaremos con el diseño de la distribución para la base de datos iniciado en el ejemplo relativo a la fragmentación horizontal primaria. En el ejemplo anterior, número 8, ya se decidió fragmentar la relación CLIENTES *de acuerdo a los fragmentos generados para* PROVINC. *Ahora vamos a fragmentar la relación* ALBCLIT. *Consideremos que existen dos aplicaciones:*

1. *Por un lado, se desean obtener los códigos de los clientes a los que se expidieron albaranes en función del año de dicha expedición.*
2. *Por otra parte, se desea obtener el número de albarán expedido a los clientes en función de la situación geográfica de los mismos.*

Para la primera aplicación la relación ALBCLIT *se fragmentará a partir de los fragmentos que se obtuvieron anteriormente para la relación* ALBCLIL. *Recuerde que*

ALBCLIL1 :σ NNUMALB≤ 990000 (ALBCLIL)
ALBCLIL2 : σ NNUMALB > 990000 (ALBCLIL)

Entonces, la fragmentación derivada de ALBCLIT *acorde con* {ALBCLIL1, ALBCLIL2} *se define como*

$$ALBCLIT_1 = ALBCLIT \triangleright\!\triangleleft ALBCLIT_1$$
$$ALBCLIT_2 = ALBCLIT \triangleright\!\triangleleft ALBCLIT_2$$

La segunda aplicación podría definirse en SQL como

SELECT NNUMALB FROM ALBCLIT, CLIENTESi WHERE
ALBCLIT.CCODCLI = CLIENTESi.CCODCLI

donde i *= 1, 2, 3, 4, dependiendo del sitio al que se dirija la consulta. Entonces, la fragmentación derivada de* ALBCLIT *de acuerdo con la fragmentación de* CLIENTES *sería*

$$ALBCLIT_1 = ALBCLIT \triangleright\!\triangleleft CLIENTES_1$$
$$ALBCLIT_2 = ALBCLIT \triangleright\!\triangleleft CLIENTES_2$$
$$ALBCLIT_3 = ALBCLIT \triangleright\!\triangleleft CLIENTES_3$$
$$ALBCLIT_4 = ALBCLIT \triangleright\!\triangleleft CLIENTES_4$$

Este ejemplo demuestra dos cosas:

1. La fragmentación derivada debe seguir una cadena donde una relación se fragmenta como resultado del diseño de otra y, en cambio, provoca la fragmentación de otra relación. Por ejemplo, la cadena PROVINC-CLIENTES-ALBCLIT.
2. Normalmente, existirá más de una forma de fragmentar una relación (como en este caso la relación ALBCLIT). La elección final del esquema de fragmentación será una decisión guiada por la asignación.

Los fragmentos resultantes se mostrarán si pulsa aquí. Se habrá percatado de la ausencia de los fragmentos ALBCLIT2 y ALBCLIT3. Este hecho se debe a que no existen tuplas en ALBCLIT que puedan derivarse de los fragmentos 2 y 3 de la relación CLIENTES. Dicho con otras palabras, no hay albaranes expedidos a clientes residentes en Castilla – La Mancha y Cataluña. Por tanto, al realizar una fragmentación horizontal derivada pueden aparecer fragmentos vacíos.

Comprobación de la corrección.

Procederemos ahora a probar la corrección de los algoritmos presentados con respecto a los tres criterios enunciados páginas atrás.

1. Compleción. La compleción de una fragmentación horizontal primaria se basa en la selección de los predicados a usar. En la medida que los predicados seleccionados sean completos, se garantizará que el resultado de la fragmentación también lo será. Partiendo de la base que el algoritmo de fragmentación es un conjunto de predicados completos y mínimos *Pr'*, se garantiza la compleción siempre que no aparezcan errores al realizar la definición de *Pr'*. La compleción de una fragmentación horizontal derivada es algo más difícil de definir. La dificultad viene dada por el hecho de que los predicados que intervienen en la fragmentación forman parte de dos relaciones. Definamos la regla de compleción formalmente. Sea *R* la relación miembro de un enlace cuya propietaria es la relación *S*, la cual está fragmentada como *FS* = {*S1, S2, ..., Sw*}. Además, sea *A* el atributo de yunto entre *R* y *S*. Entonces para cada tupla *t* de *R*, existirá una tupla *t'* de *S* tal que *t*[*A*] = *t'*[*A*].

2. Reconstrucción. La reconstrucción de una relación global a partir de sus fragmentos se desarrolla con el operador de unión tanto para la fragmentación horizontal primaria como para la derivada. Por tanto, para una relación *R* fragmentada en *FR* = {*R1, R2, ..., Rw*},

$$R = \bigcup R_i, \forall R_i \in F_R$$

3. Disyunción. Resulta sencillo establecer la disyunción de la fragmentación tanto para la primaria como para la derivada. En el primer caso, la disyunción se garantiza en la medida en que los predicados mintérmino que determinan la fragmentación son mutuamente exclusivos. En la fragmentación derivada, sin embargo, implica un semi-yunto que añade complejidad al asunto. La disyunción puede garantizarse si el grafo de yunto es simple. Si no es simple, será necesario investigar los valores de las tuplas. En general, no se desea juntar una tupla de una relación miembro con dos o más tuplas de una relación propietaria cuando estas tuplas se encuentran en fragmentos diferentes a los de la propietaria. Esto no es fácil de establecer, e ilustra por qué los esquemas de la fragmentación derivada que generan un grafo de yunto simple son siempre más atractivos.

FRAGMENTACIÓN MIXTA

En muchos casos la fragmentación vertical u horizontal del esquema de la base de datos no será suficiente para satisfacer los requisitos de las aplicaciones. Como ya se citó al comienzo de este documento podemos combinar ambas, utilizando por ello la denominada fragmentación mixta. Cuando al proceso de fragmentación vertical le sigue una horizontal, es decir, se fragmentan horizontalmente los fragmentos verticales resultantes, se habla de la fragmentación mixta HV. En el caso contrario, estaremos ante una fragmentación VH. Una característica común a ambas es la generación de árboles que representan la estructura de fragmentación (vea la figura 5).

Considere, por ejemplo, la relación PROVINC. Recordará que se le aplicó una fragmentación horizontal de acuerdo al valor del atributo CCODZONA resultando cuatro fragmentos horizontales. Podríamos pensar en aplicarle una nueva fragmentación de carácter vertical. Entonces resultarían cuatro fragmentos horizontales divididos, por ejemplo, en dos fragmentos verticales. En este caso el número total de fragmentos ascendería, lógicamente, a ocho.

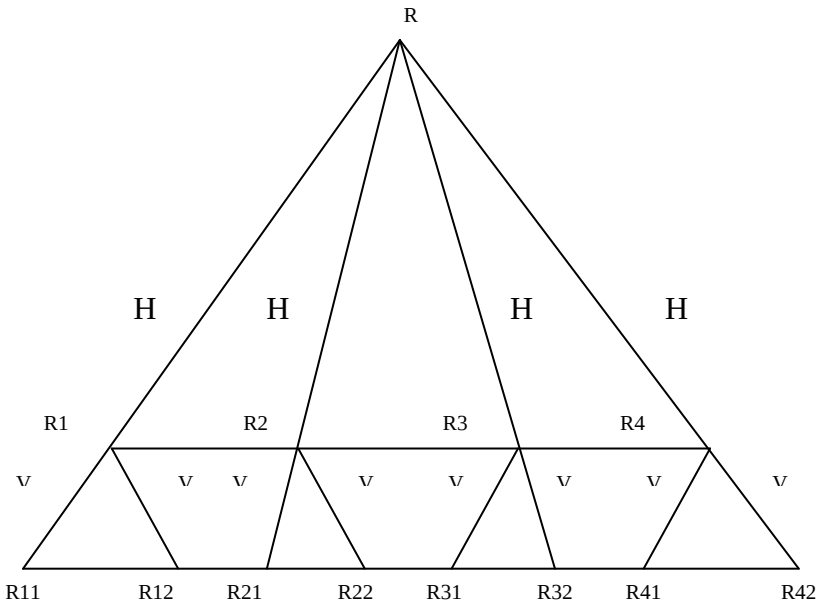

Figura 5. Estructura arbórea de fragmentación mixta

No se desea entrar en excesivos detalles sobre las reglas y condiciones para efectuar la fragmentación mixta. Entre otras razones porque, tanto a la fragmentación HV como la fragmentación VH, se le pueden aplicar los mismos criterios y reglas que a la fragmentación horizontal y vertical. Es decir, volviendo al ejemplo anterior, al cual le practicamos la fragmentación HV, al realizar la fragmentación horizontal tal como se ha expuesto, lo que se obtienen no son más que subrelaciones, la unión de las cuales da lugar a la relación PROVINC. Por tanto, para fragmentar cada subrelación sería perfectamente viable aplicarle el método de fragmentación vertical que se ha desarrollado. Como, en este caso, se han querido generar dos fragmentos verticales por cada uno horizontal, simplemente deberíamos confeccionar la matriz de grupos afines (a través del algoritmo *BEA*) para cada fragmento horizontal y aplicarle, posteriormente, el algoritmo de fragmentación binaria PARTICIÓN.

También debe tenerse en cuenta el número de niveles arbóreos que se generen, es decir, nadie impide que tras realizar una fragmentación VH, podamos aplicar a los fragmentos resultantes una nueva fragmentación vertical, y a estos últimos una nueva fragmentación horizontal, etc.

Dicho número puede ser grande, pero también será ciertamente finito. En el caso horizontal, el nivel máximo de profundidad se alcanzará cuando cada fragmento albergue una única tupla, mientras que en el caso vertical el final llegará cuando cada fragmento contenga un único atributo. Sin embargo, aunque no deba tomarse como dogma, el número de niveles no debería superar el par (VH y HV). El porqué de esta afirmación es bien sencillo, piense, por ejemplo, en el coste que supondría realizar la unión o el yunto de una relación con fragmentación nivel 7.

Evidentemente, el coste sería muy elevado y ese aumento de rendimiento que se persigue al aplicar estas técnicas, quizás, no se produzca.

Antes de pasar a estudiar el problema de la asignación se desea comentar la técnica de fragmentación mixta basada en celdas [2]. Esta técnica se basa en la generación de celdas de rejilla. Qué es una celda de rejilla, podríamos definirla como un fragmento horizontal y vertical simultáneo. La técnica aplica un algoritmo de fragmentación vertical y otro horizontal de manera concurrente sobre la relación. Los algoritmos realizan una fragmentación máxima, es decir, se persigue que en cada celda únicamente haya un atributo y una tupla. Quizá el lector pueda encontrar el método contradictorio con lo citado anteriormente respecto a la eficiencia, dada la gran cantidad de fragmentos generados, el número es, efectivamente, el máximo. Sin embargo, este sólo es el primer paso del proceso. Una vez generadas las celdas se aplica un método para optimizar la rejilla mediante fusión o desfragmentación, de acuerdo, fundamentalmente, a las aplicaciones que actúen sobre esos fragmentos.

El método, por tanto, persigue una fragmentación los más específica posible acorde con las aplicaciones y los sitios existentes en la red.

ASIGNACION

Definición del problema.

Sean un conjunto de fragmentos $F = \{F1, F2, ..., Fn\}$ y una red formada por el conjunto de sitios $S = \{S1, S2, ..., Sm\}$ en la cual un conjunto de aplicaciones $Q = \{q1, q2, ..., qq\}$ se ejecutan. El problema de la asignación implica encontrar la distribución óptima de *F* sobre *S*.

Uno de los problemas más importantes que necesita discusión es el significado de distribución óptima. La distribución óptima puede definirse con respecto a dos medidas:

1. Coste mínimo. La función de coste consiste en el coste de almacenamiento de cada *Fi* en un sitio *Sj*, el coste de practicar una consulta en *Fi* en el sitio *Sj*, el coste de actualizar *Fi* en todos los sitios donde se almacene y el coste de las comunicaciones de datos. El problema de la asignación, entonces, intenta encontrar un esquema de asignación tal que minimice esta función de coste combinado.

2. Rendimiento. La estrategia de asignación se diseña para mantener una medida del rendimiento. Dos medidas habituales de este rendimiento son el tiempo de respuesta y la salida del sistema en cada sitio. Evidentemente, se debe intentar minimizar la primera y maximizar la segunda.

Se han propuesto modelos de asignación que enfocan el concepto de distribución óptima desde diferentes puntos de vista. Sin embargo, no resulta descabellado pensar en la inclusión, tanto del rendimiento como de los factores de coste, dentro del concepto. En otras palabras, deberíamos buscar un esquema de asignación tal que, por ejemplo, la respuesta a las consultas de los usuarios se realizase en el menor tiempo posible mientras que el coste de procesamiento fuese mínimo. Una afirmación similar podría hacerse respecto a la maximización de la salida del sistema.

Consideremos ahora una formulación del problema muy simple. Definamos *F* y *S* como se hizo anteriormente. Por el momento, consideraremos únicamente un fragmento sencillo, *Fk*. Daremos un número de definiciones que nos permitan modelar el problema de la asignación.

1. Asumiremos que *Q* puede modificarse de tal manera que sea posible identificar las consultas de actualización de las de lectura, y definiremos lo siguiente para ese fragmento simple *Fk*:

$$T = \{t_1, t_2, ..., t_m\}$$

donde *ti* es el tráfico de lectura que se genera en el sitio *Si* para *Fk*, y

$$U = \{u_1, u_2, ..., u_m\}$$

donde *ui* es el tráfico de actualización que se genera en el sitio *Si* para *Fk*.

2. Asumiremos que el coste de comunicaciones entre un par de sitios *Si* y *Sj* es fijo para una unidad de transmisión. Además, asumiremos que éste es diferente para actualizaciones y para lecturas, por lo que definimos:

$$C(T) = \{C_{12}, C_{13}, ..., C_{1m}, ..., C_{m-1}, C_m\}$$
$$C'(T) = \{C'_{12}, C'_{13}, ..., C'_{1m}, ...C'_{m-1}, C'_m\}$$

donde *cij* es el coste de la unidad de comunicación para las peticiones de lectura entre los sitios *Si* y *Sj* y *c'ij* es el coste de la unidad de comunicación para las peticiones de lectura entre los sitios *Si* y *Sj*.

3. Sea di el coste de almacenar el fragmento en el sitio *Si*. Entonces definimos *D* = {*d1, d2, ..., dm*} como el coste de almacenar *Fk* en todos los sitios.

4. Asumiremos que no existen restricciones de capacidad en los sitios o en los enlaces de comunicaciones.

Entonces el problema de asignación puede especificarse como un problema de minimización de costes por el cual se intenta encontrar el conjunto *I S* que especifique el lugar donde han de ubicarse las copias de los fragmentos. La expresión matemática hace uso de la variable de decisión para la ubicación *xj* es,

xj = 1 si el fragmento *Fk* se asigna al sitio *Sjxj* = 0 en otro caso

entonces, definida *xj*,

$$\min\left[\sum_{i=1|}^{m}\left(\sum_{j|s_j\in i} x_j u_j c'_{ij} + t_j \min_{j|sj\in i} c_{ij}\right) + \sum_{j|s_j\in i} x_j d_j\right]$$

El segundo término de la función calcula el coste total de almacenar todas las copias duplicadas del fragmento. El primer término corresponde al coste de transmisión de las actualizaciones a todos los sitios que mantienen réplicas de un fragmento y al coste de ejecución de las peticiones de lectura en el sitio, lo cual resultará un coste mínimo de transmisión de datos.

Esta es una formulación muy simple que no es válida para el diseño de bases de datos distribuidas. Pero en el caso que lo fuera, existiría un problema. Para un gran número de fragmentos y de sitios, obtener soluciones óptimas resultaría probablemente totalmente inviable. Las investigaciones, por tanto, deben girar en torno a la búsqueda de buenos heurísticos que proporciones soluciones parcialmente óptimas.

Hay un número de razones del porqué de formulaciones tan simples que no sirven para el diseño de bases de datos distribuidas. Generalmente, se heredan de los modelos de asignación de archivos para redes, pero

1. No se pueden tratar los fragmentos como archivos individuales que se asignen aisladamente. La ubicación de un fragmento generalmente tiene influencia sobre las decisiones de asignación de los otros fragmentos, a los cuales se acceden a la vez, puesto que el coste de acceso de los fragmentos restantes puede variar. Por tanto, las relaciones entre fragmentos deben tenerse en consideración.
2. El acceso de las aplicaciones a los datos se modela muy sencillamente. Una petición de usuario se resuelve en un sitio y todos los datos necesarios se transfieren a ese sitio. En los sistemas de bases de datos distribuidos, el acceso a los datos es más complicado que el simple acceso a archivos remotos. Por tanto, la relación entre la asignación y el procesamiento de consultas debería también tenerse en cuenta.
3. Estos modelos no tienen en cuenta el coste de mantenimiento de la integridad, aún localizando dos fragmentos implicados con las mismas restricciones de integridad en dos sitios diferentes podría resultar costoso dicho mantenimiento.
4. Igualmente, el coste derivado del control de concurrencia debería tenerse en cuenta.

En resumen, debemos distinguir entre el problema tradicional de asignación de archivos de la asignación de fragmentos en los sistemas de bases de datos distribuidos. No existen modelos heurísticos generales que tomen como entrada un conjunto de fragmentos y produzcan una asignación cercana a lo óptimo que además esté influenciada por los tipos de restricciones descritas antes. Los modelos desarrollados realizan una serie de simples suposiciones y pueden aplicarse a ciertas formulaciones específicas. Por tanto, presentaremos un modelo general y discutiremos una serie de posibles heurísticos que puedan emplearse para resolver el problema. Posteriormente, describiremos un algoritmo concreto de asignación.

Información necesaria.

En esta etapa de la asignación, necesitaremos datos cuantitativos sobre la base de datos, las aplicaciones que funcionan sobre ella, la red de comunicaciones, las características de proceso, y el límite de almacenamiento de cada sitio de la red. Procederemos a discutirlos en detalle.

Información de la base de datos. Para desarrollar la fragmentación horizontal, definimos la selectividad de los mintérminos. Ahora, necesitamos extender esta definición a los fragmentos y definir la selectividad de un fragmento *Fj* con respecto a una consulta *qi*. Es el número de tuplas de *Fj* a las que se necesita acceder para procesar *qi*. Este valor lo notaremos como *seli*(*Fj*). Otro elemento informativo de los fragmentos de la base de datos es su tamaño. El tamaño de un fragmento *Fj* viene dado por *tamaño*(*Fj*) = *card*(*Fj*)**long*(*Fj*), donde long(Fj) es la longitud (en octetos) de una tupla del fragmento Fj.

Información de las aplicaciones. Mucha de la información relativa a las aplicaciones se recoge durante el proceso de fragmentación, pero se necesita un poco más para el modelo de asignación. Las dos medidas más importantes son el número de accesos de lectura que una consulta qi realiza sobre un fragmento *Fj* durante su ejecución (llamada *RRij*), y el número de accesos de actualización que una consulta *qi* realiza sobre un fragmento *Fj* durante su ejecución (llamada *URij*). También necesitamos definir dos matrices *UM* y *RM*, con elementos *uij* y *rij*, respectivamente, que se especifican como sigue:

uij = 1 si la consulta *qi* actualiza el fragmento *Fj uij* = 0 en otro caso

rij = 1 si la consulta *qi* lee del fragmento *Fj rij* = 0 en otro caso

También debe definirse un vector *O* de valores *o*(*i*), donde *o*(*i*) especifica el sitio origen de la consulta *qi*. Finalmente, especificaremos las restricciones impuestas por el tiempo de respuesta, asignando a cada aplicación el máximo tiempo de respuesta permitido.

Información de los sitios. Sobre cada ordenador necesitamos conocer sus capacidades de procesamiento y almacenamiento. Obviamente, estos valores pueden calcularse a través de funciones elaboradas o por simples estimaciones. La unidad de coste de almacenar datos en el sitio *Sk* será denotada como *UCAk*. Así mismo, especificaremos como medida de coste *UPTk* al coste de procesar una unidad de trabajo en el sitio *Sk*. La unidad de trabajo debería ser idéntica a aquella utilizada en las medidas *RR* y *UR*.

Información sobre la red. En nuestro modelo asumiremos la existencia de una red simple donde el coste de comunicaciones se define respecto a una trama de datos. Entonces *gij* nota el coste de comunicación por trama entre los sitios *Si* y *Sj*. Para permitir el cálculo del número de mensajes, usaremos *ftamaño* como el tamaño (en octetos) de una trama. Es evidente que existen modelos de red mucho más elaborados que toman en cuenta las capacidades del canal, las distancias entre sitios, las características del protocolo, etc. Sin embargo, se cree que la derivación de estas ecuaciones se sale fuera de este documento.

Modelo de asignación.

Discutiremos un modelo de asignación que intente minimizar el coste total de procesamiento y almacenamiento a la vez que intenta reunir ciertas restricciones en el tiempo de respuesta. El modelo que emplearemos tiene la forma *mín*(*Coste Total*), la cual está sujeta a restricciones del tiempo de respuesta, restricciones de almacenamiento y restricciones de procesamiento.

En el resto de este punto desarrollaremos los componentes de este modelo basándonos en la información necesaria presentada anteriormente. La variable de decisión es *xij*, la cual se define como

xij = 1 si el fragmento *Fi* se almacena en el sitio *Sj*

xij = 0 en otro caso

Coste total. La función de coste total tiene dos componentes: el procesamiento de la consulta y el almacenamiento. Entonces podríamos expresarla como

$$CTO = \sum_{\forall qi \in Q} CPQ_i + \sum_{\forall Sk \in S} \sum_{\forall Fj \in F} CAF_{jk}$$

donde *CPQi* es el coste de procesar una consulta de la aplicación *qi*, y *CAFjk* es el coste de almacenar el fragmento *Fj* en el sitio *Sk*.

Consideremos primero el coste de almacenamiento. Su fórmula viene dada por

$$CAF_{jk} = UCA_k * tamaño(F_j) * x_{jk}$$

donde se representa el coste total de almacenamiento en todos los sitios y para todos los fragmentos.

El coste de procesamiento de consultas es más difícil de especificar. Muchos modelos de asignación de archivos se dividen en dos componentes: el coste de procesar las lecturas y el coste de procesar las actualizaciones. Nosotros escogeremos un enfoque diferente para el problema de asignación en las bases de datos y lo especificaremos a partir del coste de procesamiento (*CP*) y el coste de transmisión (*CT*). El coste de procesamiento de una consulta (*CPQ*) para una aplicación *qi* es

$$CPQ_i = CP_i + CT_i$$

De acuerdo con las líneas presentadas anteriormente, el componente de procesamiento *CP* se basa en tres factores: el coste de acceso (*CA*), el coste de mantenimiento de la integridad (*MI*) y el coste de control de la concurrencia (*CC*):

$$CP_i = CA_i + MI_i + CC_i$$

La especificación detallada de cada uno de estos factores depende del algoritmo que se emplee para desarrollar estas tareas. Sin embargo, se especificará *CA* detalladamente:

$$CA_i = \sum_{\forall S_k \in S} \sum_{\forall F_j \in F} \left(u_{ij} UR_{ij} + r_{ij} RR_{ij}\right) x_{jk} UPT_k$$

El primero de los términos de la fórmula calcula el número de accesos de la consulta *qi* al fragmento *Fj*. Advierta que (*URij* + *RRij*) da el número total de accesos de lectura y actualización. Asumiremos que los costes locales de procesamiento de ambos son idénticos. El sumatorio proporciona el número total de accesos para todos los fragmentos a los que accede *qi*. El producto por *UPTk* da el coste de este acceso al sitio *Sk*. Usamos de nuevo, *xjk* para seleccionar únicamente los valores de coste para los sitios donde se almacenan los fragmentos.

Se debe tener en cuenta que la función de coste de acceso asume que el procesamiento de una consulta implica su descomposición en una serie de subconsultas, cada una de las cuales trabaja sobre un fragmento almacenado en un sitio, seguido de una transmisión de los resultados al sitio del cual partió la consulta. Se vio, anteriormente, que es un enfoque muy simplista no tener en cuenta la complejidad del procesamiento de la base de datos. Por ejemplo, la función de coste no tiene en cuenta el coste de desarrollar yuntos (si fuese necesario), lo cual puede ejecutarse de varias formas. En un modelo más realista, que el modelo genérico considerado, esto problemas no deberían omitirse.

El factor de coste del esfuerzo de integridad puede especificarse como el componente de procesamiento, excepto que la unidad de coste de procesamiento local, probablemente, cambiaría para reflejar el coste real del esfuerzo de integridad.

La función del coste de transmisión puede formularse sobre las líneas de la función del coste de acceso. Sin embargo, los gastos de la transmisión de datos para actualizaciones y para lecturas no es el mismo. En las consultas de actualización, es necesario informar a todos los sitios donde existen réplicas, mientras que en las consultas de lectura, es suficiente con acceder al sitio que alberga las copias. En suma, al final de una petición de actualización, no existe una transmisión de datos al sitio origen de ésta, sino un mensaje de confirmación, mientras que en las consultas de lectura, los datos a transmitir al origen son significativos.

El componente de actualización de la función de transmisión es

$$CTA_i = \sum_{\forall S_k \in S} \sum_{\forall F_j \in F} u_{ij} * x_{jk} * g_{o(i),k} + \sum_{\forall S_k \in S} \sum_{\forall F_j \in F} u_{ij} * x_{jk} * g_{k,o(i)}$$

El primer término es para el envío del mensaje de actualización de *qi* desde el sitio origen *o*(*i*) a todas las réplicas de los fragmentos que necesiten actualizarse. El segundo término hace referencia a la confirmación. El coste de lectura puede especificarse como

$$CTL_i = \sum_{\forall S_k \in S} \min_{S_k \in S} \left(u_{ij} * x_{jk} * g_{o(i),k} + r_{ij} * x_{jk} \frac{sel_i(F_j)}{f_{tamaño}} g_{k,o(i)} \right)$$

El primer término de *CTL* representa el coste de transmitir la petición de lectura a los sitios que contienen copias de los fragmentos a los que se necesita acceder. El segundo término cuenta para la transmisión de los resultados desde estos sitios al sitio origen. La ecuación afirma que para todos los sitios con copias del mismo fragmento, sólo el sitio que produzca el coste total de transmisión más pequeño debería seleccionarse para la ejecución de la operación.

Ahora, la función del coste de la transmisión para la consulta qi puede especificarse como

$$CT_i = CTA_i + CTL_i$$

que indica la función de coste total.

Restricciones. Las funciones restrictivas pueden especificarse de forma similar. Sin embargo, en lugar de describir estas funciones con detalle, simplemente indicaremos el aspecto que deberían tener. El tiempo de respuesta debería especificarse como tiempo de ejecución de $qi \leq$ máximo tiempo de respuesta de $qi, \forall\ qi \in Q$

Preferiblemente, la medida de coste en la función objetiva debería especificarse en términos de tiempo, para hacer la especificación del tiempo de ejecución relativamente sencilla.

La restricción de almacenamiento es

$$\sum_{\forall F_j \in F} RAL_{jk} \leq capacidad \quad de \quad almacenamiento \quad de\, S_k, \forall S_k \in S$$

Así misma, la restricción de procesamiento es

$$\sum_{\forall q_i \in Q} c\arg a \quad de \quad procesamiento \quad de \quad q_i \, en \quad el \quad sitio \quad Sk \leq$$

$$capacidad \quad de \quad proceso \quad de\, S_{k^1} \, \forall Sk \in S$$

Esto completa el desarrollo del modelo de asignación.

Desarrollo práctico.

Vamos a presentar ahora dos alternativas prácticas de desarrollo de la asignación. Una primera consistiría en el cálculo de todos los costes y, a partir de sus resultados y con el mejor esquema de partición determinado por el Examinador de Particiones, decidir los fragmentos que deberían asignarse a cada sitio. Este método manual evidentemente implica la realización de muchos cálculos muy engorrosos, y deberíamos partir de una serie de datos que no siempre es fácil obtener. Una segunda alternativa, es el uso de algún algoritmo de asignación desarrollado a partir de los distintos parámetros del modelo de asignación. Existen varios de estos algoritmos, pero se ha decidido exponer el algoritmo divide y vencerás [7] porque hace uso del esquema de fragmentación que genera el algoritmo de fragmentación n-formas presentado en la correspondiente sección.

Método de asignación "divide y vencerás".

Partiremos del conjunto de esquemas de asignación que produce el algoritmo N-FORMAS. Un aspecto importante de este algoritmo es que, como ya se comentó, produce esquemas de fragmentación jerárquicos, es decir, partiendo de *k* fragmentos obtiene *k+1* fragmentos para el siguiente esquema.

Seguidamente, describiremos el funcionamiento del algoritmo. En la primera parte del mismo (antes del primer bucle para), se intenta encontrar la mejor asignación del fragmento único (el esquema número 1) y el coste de esta operación. Una vez que se tiene el fragmento en el mejor sitio, se calcula el coste de mantenerlo en este sitio invocando a la función *CosteConsulta*. Una llamada a esta función devuelve el coste para una determinada consulta basada en su plan de ejecución.

La segunda parte del algoritmo va desde el paso 2 hasta el final del mismo. Dado un esquema de fragmentación, esta parte calcula el mejor coste de mover los dos nuevos fragmentos a los diferentes sitios de la red, dejando el resto de los fragmentos quietos en los sitios que se determinaron mejores para ellos. Se repite el bucle para del paso 2 para todos los esquemas de fragmentación de tal manera que el algoritmo termina proporcionando el mejor esquema de partición y los sitios en donde se debería ubicar cada fragmento. Advierta que este algoritmo lleva implícita la tarea realizada por el Examinador de Particiones, por lo que no es necesario emplearlo en este enfoque. Por último, se desea señalar que la complejidad del algoritmo es de $O(ns^2qx)$ donde *n, s* y *q* son el número de fragmentos, sitios y consultas, respectivamente, y $O(x)$ el coste de llamar a la función *CosteConsulta*.

```
Algoritmo de ASIGNACIÓN-DYV
Entrada:
        EEF_n: esquema de Fragmentación
        {cada EEF_i es un esquema de fragmentación}
        {cada P_i es un par fragmento – sitio para el esquema de fragmentación actual i}
        {EEF_1 es el único esquema de fragmentación que contiene un único fragmento con todos
        los atributos}
declarar
        P_r: el mejor lugar de ubicación del esquema EEF_r resultado de moverlo como un único
        elemento alrededor de toda la red;
        mejorcoste: el coste de ubicar P_r
inicio
        esquema←1
        i←2
        para cada fragmento de EEF_i hacer
        inicio
                sean c_j y c_k dos nuevos grupos de fragmentos NGF_i;
                P_i←P_i-r –{(cj ∪ ck), S((cj ∪ ck))}
                {costec_jc_k es el coste de mover únicamente dos nuevos grupos de fragmentos
                alrededor de la red manteniendo el resto de grupos de fragmentos fijos en sus
                mejores sitios, calculado en interacciones previas}
                costec_jc_k←∞
                para cada posible s_j del grupo c_j hacer
                        para cada posible s_k del grupo c_k hacer
                        inicio
                                {Pi ahora contiene la ubicación de cada grupo de fragmentos
                                en EEF excepto los dos nuevos grupos c_j y c_k}
                                z←suma para toda consulta q de costeconsulta(q,
                                Pi∪{(c_j,s_j),(c_k,s_k)},OQD);
                                si z<costec_jc_k entonces
                                inicio
                                        costec_jc_k←z
                                        s_jmejor←s_j
                                        s_kmejor←s_k
                                fin-si
                        fin-para
                        P_i←P_i∪{(c_j,s_jmejor),(c_k,s_kmejor)}
                        si costec_jc_k<mejorcoste entonces
                        inicio
                                mejorcoste←costec_jc_k
                                {el EEF actual i debe ser el mejor que conduzca al menor
                                coste de procesamiento de la consulta}
                                esquema←i
                        fin-si
                        i←i+1
        fin-para
fin. {ASIGNACIÓ-DYV}
```

A lo largo de este documento se ha intentado dar una visión global y genérica de los problemas y características que contiene el diseño de una base de datos distribuida. Se ha hecho especial hincapié en las técnicas de fragmentación horizontal y vertical a través de métodos y algoritmos muy frecuentes en la literatura referida al tema. Se espera que el lector no haya tenido demasiados problemas para su comprensión, las técnicas son sencillas y se ha procurado incluir distintos ejemplos para facilitar el entendimiento. Igualmente, la puesta en práctica de los algoritmos, es decir, su codificación, no es un proceso complicado si se poseen nociones en el desarrollo de algoritmos. Piense, por ejemplo, que los dos algoritmos de partición vertical presentados, no hacen más que manipular matrices.

También debería tenerse presente la existencia de enfoques de fragmentación distintos y, posiblemente, más complejos, pero se debe pensar que más eficientes. Sean, por ejemplo, las técnicas de fragmentación vertical basadas en grafos, como el algoritmo de Navathe y Ra que genera en un solo paso fragmentos verticales. Además, están apareciendo métodos de fragmentación mixta como el que se ha comentado. Si bien, estos métodos son enfoques formales más que prácticos, desarrollados por insignes investigadores en universidades, por tanto, lejos todavía de su desarrollo comercial.

Pese a la aparición de los métodos de bases de datos distribuidas hace ya años, parece que el salto de lo centralizado a lo distribuido a escala comercial está por venir. Todavía no se ha extendido suficientemente el esquema distribuido, pero se espera que próximamente se produzca el avance definitivo. Considere los dos componentes básicos de los sistemas de bases de datos distribuidos (la propia base de datos y la red de ordenadores) y piense en la situación actual de la informática. Si las bases de datos es una de las ramas más antiguas e importantes de la informática, muchas empresas compran ordenadores para dedicarlos exclusivamente a la gestión de sus datos (pienso que, prácticamente, en el 100% de las PYMES se produce este hecho) y, como parece ser que se ha asumido por parte de todo tipo de empresarios los beneficios que acarrea la conexión de los ordenadores, la instalación de una red, se puede concluir diciendo que el terreno ya está abonado para su extensión comercial. Sólo falta que determinadas multinacionales decidan apostar más fuerte por este enfoque a través de sus famosos sistemas gestores de bases de datos y que se produzca la consolidación de la resolución de los problemas que el enfoque distribuido acarrea.

ENLACES Y BIBLIOGRAFÍA

[1]* Özsu M.T., Valduriez P., *Principles of Distributed Database Systems*, Prentice Hall, 1991.

[2]* Navathe S.B., Karlapalem K., Minyoung Ra., *A Mixed Fragmentation Methodology for Initial Distributed Database Design*, 1997. http://www.cs.ust.hk

[3] Niamir B., *Attribute Partitioning in a Self-Adaptive Relational Database System*, Technical Report 192, Laboratory for Computer Science, Massachusetts Institute of Technology, 1978.

[4] Hoffer H.A., Severance D.G.,*The Use of Cluster Analysis in Physical Data Base Design*,

First International Conference on Very Large Data Bases, Massachusetts, septiembre 1975.

[5] Navathe S.B., Ceri S., Wiederhold G., Dou J., *Vertical Partitioning of Algorithms for Database Design*, ACM, diciembre 1984.

[6] McCormick W.T., Schweitzer P.J., White T.W., *Problem Decomposition and Data Reorganization by a Clustering Technique*, Oper. Res., 1972.

[7]* Shepherd J.A., Harangsri B., Chen H.L., Ngu A.H.H., *A Two-Phase Approach to Data Allocation in Distributed Databases*, Fourth International Conference on Database Systems for Advanced Applications, World Scientific Press, Singapur, Singapur Abril 1995, 1996. http://www.cse.unse.edu.au

[8]* Muthuraj J., Chakravarthy S., Varadarajan R., Navathe S.B., *A Formal Approach to the Vertical Partitioning Problem in Distributed Database Design*. PDIS, 1993. http://www.cis.efl.edu

[9] Jain A., Dubes R., *Algorithms for Clustering Data*, Prentice Hall Advanced Series, 1988.

[10]* Smith, J.K., *Survey Paper on Vertical Partitioning*, Universidad de Hawaii, Noviembre 1997. http://www.ics.hawaii.edu

[11]* Bell D., Grimson J., *Distributed Database Systems*, Addison – Wesley, 1992.

[12] Karlapalem K., Ng Moon Pun, *Query-Driven Data Allocation Algorithms for Distributed Database Systems*, DEXA 1997. http://www.cs.ust.hk

[13] Brunstrom A., Leutenegger S.T., Simha R., *Experimental Evaluation of Dynamic Data Allocation Strategies in a Distributed Database With Changing Workloads*, Fourth International Conference on Information and Knowledge Management, November 1995. http://www.cs.du.edu