

Tema 2 - Diseño de Bases de Datos Distribuidas

1. El proceso de diseño de una base de datos distribuida

El diseño de una base de datos centralizada consiste en:

- **Diseño del esquema conceptual** (diagrama E/R si seguimos un modelo E-R)
- **Diseño del esquema lógico** (el diagrama E/R, lo traducimos a las estructuras que use el DBMS (en nuestro caso, tablas). Después de esta etapa podríamos tener una etapa de optimización/normalización.
- **Diseño del esquema físico** (lo suele hacer el DBA).

En una **base de datos distribuida**, estas etapas equivalen a:

- **Diseño del esquema conceptual.**
- **Diseño del esquema lógico global.**
- **Diseño de los esquemas físicos globales** (en una localidad se ha de decidir cómo se van a almacenar los datos).

(la distribución de una base de datos **añade**):

- **Diseño de la fragmentación; “criterio lógico”** que motiva la fragmentación.
- **Diseño de la asignación de fragmentos;** localización **“física”** de los datos.

Hemos de considerar también los requerimientos de las aplicación más “importantes” que van a usar luego la base de datos distribuida:

Por un lado, está la **localidad de origen de la aplicación**, es decir, dónde, en qué localidad se va a ejecutar la aplicación (si es en una, en varias, etc.). Por otro, la **frecuencia de activación de cada aplicación en cada localidad**, es decir, cuántas veces por unidad de tiempo se ejecuta la aplicación. Y por último, el **número, tipo y distribución estadística de los accesos de cada aplicación a cada objeto de datos**.

2. Objetivos del diseño de la distribución de datos

En esta sección se recogen los objetivos principales que se persiguen a la hora de diseñar la distribución de los datos, en el diseño de una BDD:

- **Procesamiento local** (el que más se usa): consiste en distribuir los datos de forma que se maximice el procesamiento local. Es decir, que la mayor parte de las aplicaciones

utilicen datos de la localidad en la que se ejecutan. En definitiva, distribuir los datos lo más cerca posible de las aplicaciones que los van a utilizar.

Hemos de considerar dos tipos de **referencias de datos**: las **referencias locales** y las **remotas**. Pues, **maximizar el procesamiento local**, va a equivaler a **minimizar el número de referencias remotas**.

Una aplicación que no haga referencias remotas, es decir, que puede ejecutarse completamente en su localidad de origen, es lo que se llama **aplicación completamente local**.

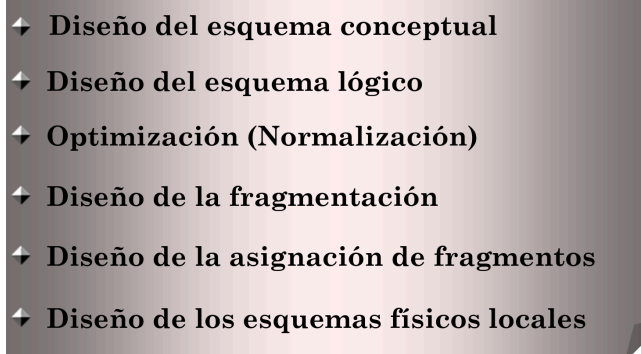
- **Disponibilidad y fiabilidad de los datos distribuidos**: alto grado de disponibilidad (para aplicaciones de solo lectura) y fiabilidad puede conseguirse almacenando múltiples copias de los mismos datos (réplicas).
- **Distribución de la carga de trabajo**: Aprovechar la potencia y uso de los computadores de cada localidad, y maximizar el grado de paralelismo en la ejecución de las aplicaciones (puede afectar negativamente al procesamiento local).
- **Costes y disponibilidad de almacenamiento**: No es un factor relevante, pero debería tenerse en cuenta.

(hoy en día estos dos últimos objetivos no tienen mucho sentido. No se usan mucho)

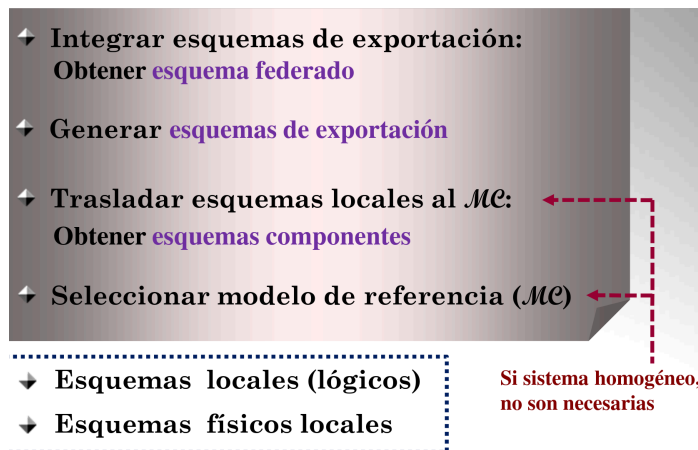
3. Estrategias de diseño de una base de datos distribuida

A la hora de diseñar una base de datos distribuida, podemos seguir **dos estrategias**:

- **Estrategia descendente (top-down)**: se usa para el diseño de Bases de datos distribuidas autonómicas. Partimos de un nivel alto de abstracción y vamos “descendiendo” hasta un nivel menos abstracto:

- 
- Diseño del esquema conceptual
 - Diseño del esquema lógico
 - Optimización (Normalización)
 - Diseño de la fragmentación
 - Diseño de la asignación de fragmentos
 - Diseño de los esquemas físicos locales

- **Estrategia ascendente (botton-up):** está basada en la “integración” (integrar varios esquemas en uno) de esquemas existentes, es decir, en la combinación de definiciones de datos comunes y en la resolución de conflictos entre diferentes representaciones de los mismos datos; detectar homónimos, sinónimos, conflictos de tipo, etc. Es usada en el diseño de **Bases de datos federadas**.



Empezando por abajo, partimos de los esquemas locales (lógicos) y físicos locales. A continuación, seleccionamos el modelo de datos de referencia, obtenemos los esquemas componentes (esquemas locales, pero expresados en otro modelo (pueden coincidir)) (estas dos últimas etapas no son necesarias, en

caso de que el sistema sea homogéneo), para después generar los esquemas de exportación (vista de un esquema componente que contiene información de datos y relaciones que una localidad va a poner a disposición de las demás). Y finalmente, integramos los esquemas de exportación para obtener el esquema federado, que es un proceso bastante complejo.

4. Tipos de fragmentación de datos

Existen tres tipos de fragmentación de datos:

- **Fragmentación horizontal:** consiste en dividir el conjunto de tuplas en subconjuntos de tuplas.
- **Fragmentación vertical:** consiste en dividir el conjunto de atributos en subconjuntos.
- **Fragmentación mixta:** partimos de un tipo de fragmentación (horizontal o vertical) y luego aplicamos a algún fragmento, otro tipo de fragmentación. **Hay que alternar los tipos de fragmentación (H-V o V-H. No H-H o V-V).**

Definimos como **fragmento** a toda expresión en un lenguaje relacional (en nuestro caso, el álgebra relacional) que considera las relaciones globales como operandos y produce como resultado el fragmento.

Los **operadores básicos del álgebra relacional** son: la **proyección** ($PJ_{\{A\}}$; SELECT), el **producto cartesiano** (CP; from T,P) y la **selección** ($SL_{\{c\}}$; where c). El operador **producto**

natural se obtiene a partir de los ya mencionados: $SN_{\{c\}} \simeq SL_{\{c\}}(CP) (T SN_{\{c\}} S \simeq SL_{\{c\}}(T CP S))$.

A continuación se presentan las **reglas** fundamentales de **fragmentación**:

- **Compleitud:** todos los datos de la relación global deben transformarse en fragmentos. Es decir, cuando hacemos una fragmentación todos los elementos tienen que estar en algún fragmento, independientemente de la fragmentación.
- **Reconstrucción:** siempre debe ser posible reconstruir una relación global a partir de sus fragmentos.
- **Fragmentación disjunta:** es conveniente que los fragmentos sean disjuntos. La fragmentación es estrictamente disjunta, siempre, en la fragmentación horizontal, y no siempre en los fragmentos verticales porque si no se relaja no se podría cumplir la regla de reconstrucción (al menos los fragmentos de la vertical han de tener la clave primaria).

4.1 Fragmentación horizontal

Consiste en partir las tuplas de una relación global en subconjuntos. Se define con el operador **SELECCIÓN** ($SL_{\{q\}}$) y se reconstruye con el operador de **UNIÓN** (**UN**) (operador básico dentro de los operadores de conjuntos. El otro básico es la diferencia). A la condición “q” (lo que hay en el where), se le denomina **cualificación**.

La fragmentación se dice **completa** si el conjunto de cualificaciones es completo con respecto al conjunto de valores permitidos.

La fragmentación se dice **disjunta** si el conjunto de cualificaciones es mutuamente excluyente.

Veamos un ejemplo:

Ejemplo:

Proveedor = (Pnum, nombre, ciudad)

Proveedor₁ = $SL_{ciudad='Granada'}$ (**Proveedor**)

Proveedor₂ = $SL_{ciudad='Jaén'}$ (**Proveedor**)

Proveedor₃ = $SL_{ciudad='Toledo'}$ (**Proveedor**)

Proveedor = **Proveedor**₁ **UN** **Proveedor**₂ **UN** **Proveedor**₃

Esta fragmentación es horizontal primaria. Además es completa porque el conjunto de cualificaciones $C = (\{ciudad='Granada', ciudad='Jaén'\})$ es completo con respecto a los valores permitidos del atributo ciudad (\nexists otro proveedor que esté en otra ciudad de las que hay en el conjunto de cualificaciones). También es disjunta la

fragmentación, dado que C es mutuamente excluyente (si un proveedor es de Granada, no puede serlo de Jaén y viceversa).

4.2 Fragmentación horizontal derivada

Fragmentación no basada en los atributos de la relación sino en la fragmentación horizontal de otra relación. Se define con el operador **SEMI-PRODUCTO NATURAL** ($SJN_{\{c\}}$; $T SJN_{\{c\}} S = PJ_{\{Atr(T)\}}(T SN_{\{c\}} S)$; **no es conmutativo**). Se reconstruye con el operador **UNIÓN** (UN).

Las cualificaciones son condiciones de existencia de tuplas y no incluyen atributos de la relación.

Ejemplo:

$Suministro = (Pnum, Anum, Dnum, cantidad)$

$Suministro_1 = Suministro SJN_{Pnum=Pnum} Proveedor_1$

$Suministro_2 = Suministro SJN_{Pnum=Pnum} Proveedor_2$

$Suministro_3 = Suministro SJN_{Pnum=Pnum} Proveedor_3$

$Suministro = Suministro_1 UN Suministro_2 UN Suministro_3$

Cualificaciones:

q1: $S.Pnum = P.Pnum$ and $P.ciudad = 'Granada'$

q2: $S.Pnum = P.Pnum$ and $P.ciudad = 'Jaén'$

q3: $S.Pnum = P.Pnum$ and $P.ciudad = 'Toledo'$

En este ejemplo se quiere fragmentar la tabla Suministro, basándonos en la fragmentación de Proveedor. Observamos que en las cualificaciones hay condiciones de existencia (no se le da valor a Pnum) y tampoco se incluyen atributos de Suministro.

4.3 Fragmentación vertical

Consiste en dividir los atributos de una relación global en grupos. Se define con el operador de **PROYECCIÓN** ($PJ_{\{Lista\}}$). Se reconstruye con el operador de **PRODUCTO NATURAL** ($JN_{\{c\}}$); aquí es donde entra en juego el hecho de que la fragmentación vertical no va a ser completamente disjunta, porque como mínimo los fragmentos tienen que tener la llave para poder reconstruir luego. Se ha de garantizar que la fragmentación es completa, es decir, que todo atributo quede luego en al menos algún fragmento.

Ejemplo:

$Empleado = (Enum, nombre, salario, IRPF, mgrnum, Dnum)$

$Empleado_1 = PJ_{Lista1} Empleado$

$Empleado_2 = PJ_{Lista2} Empleado$

$Lista1 = (Enum, nombre, mgrnum, Dnum)$

$Lista2 = (Enum, salario, IRPF)$

$Empleado = Empleado_1 JN_{Enum=Enum} Empleado_2$

Esta fragmentación, por ejemplo, es completa porque todos los atributos de Empleado están en alguno de los dos fragmentos. No es disjunta porque los fragmentos comparten la clave

primaria. No pasa nada que esté repetida en todos los fragmentos porque se supone que es un dato que es poco probable que cambie.

4.4 Fragmentación mixta

Consiste en aplicar operaciones de fragmentación recursivamente. Se reconstruye aplicando operaciones de reconstrucción en orden inverso, es decir, si primero se ha hecho una vertical y luego una horizontal, para reconstruir la relación, primero se unen los fragmentos que se obtuvieron con la horizontal y posteriormente, se hace el producto natural de los fragmentos que se obtuvieron con la vertical.

Ejemplo:

$\text{Empleado} = (\text{Enum}, \text{nombre}, \text{salario}, \text{IRPF}, \text{mgrnum}, \text{Dnum})$

$\text{Empleado}_1 = \text{PJ}_{\text{Lista1}} \text{Empleado}$

$\text{Empleado}_2 = \text{PJ}_{\text{Lista2}} \text{Empleado}$

$\text{Lista1} = (\text{Enum}, \text{nombre}, \text{mgrnum}, \text{Dnum})$

$\text{Lista2} = (\text{Enum}, \text{salario}, \text{IRPF})$

$\text{Empleado}_3 = \text{SL}_{\text{Dnum} \leq 10} (\text{Empleado}_1)$

$\text{Empleado}_4 = \text{SL}_{10 < \text{Dnum} \leq 20} (\text{Empleado}_1)$

$\text{Empleado}_5 = \text{SL}_{\text{Dnum} > 20} (\text{Empleado}_1)$

$\text{Empleado} = (\text{Empleado}_3 \text{ UN } \text{Empleado}_4 \text{ UN } \text{Empleado}_5) \text{ JN}_{\text{Enum}=\text{Enum}} \text{Empleado}_2$

5. Diseño de la fragmentación de una base de datos

A partir de la **distribución de los datos**, tenemos se tiene el propósito de determinar los fragmentos que serán “**unidades lógicas de asignación**”, es decir, determinar el conjunto de tuplas que luego vamos a asignar a las distintas localidades. El **diseño de la fragmentación** consiste en agrupar tuplas (o atributos) con las “**mismas propiedades**”, desde el punto de vista de su asignación. A cada grupo, lo llamamos Fragmento.

5.1 Fragmentación horizontal

Consiste en determinar propiedades lógicas de los datos; **predicados de fragmentación**, que son conjuntos de cualificaciones que nos van a permitir hacer los fragmentos, y en determinar propiedades estadísticas de los datos; número de referencias de las aplicaciones a los fragmentos.

5.2 Fragmentación horizontal primaria

Consiste en determinar un **conjunto completo y disjunto** de predicados de selección (o cualificaciones). Los **elementos** que componen cada **fragmento** deben ser **referenciados homogéneamente** por las aplicaciones, es decir, si tengo un fragmento en el que estén todos los empleados de una localidad, la aplicación tiene que referenciarlos con la misma probabilidad.

Para definir la **fragmentación horizontal primaria** de una relación global, se introducen los siguientes conceptos:

- **Predicado simple:** atributo + operador de relación + valor, donde operador de relación = $\{=, \neq, \leq, \geq, <, >\}$
- **Término de predicados:** Sea P un conjunto de predicados simples (p.ej $P = \{Dnum=1, Dnum=2\}$, que se evalúan a verdadero. Definimos un término de predicado como una conjunción de **predicados simples de P** , incluyendo afirmaciones y negaciones (el número de conjunciones es 2^n , donde n es el número de predicados simples):

$$y = \bigwedge_{p_i \in P} p_i^* \text{ donde } p_i^* = p_i \text{ o } \neg p_i \text{ e } y \neq \text{falso}$$

- **Fragmento:** conjunto de tuplas que verifican un término de predicados.
- **Predicado simple relevante:** Un predicado simple p_i es relevante con respecto a un conjunto de predicados simples P , si existen al menos dos términos de predicados de P cuyas expresiones difieren, únicamente en p_i , y tal que los correspondientes fragmentos son referenciados de manera diferente por el menos una aplicación.

Ejemplo:

Ejemplo:

- ✚ Relación global **Empleado**
- ✚ Aplicación que requiere información sobre los empleados de los departamentos
- ✚ Aplicación que requiere información sobre los empleados que son programadores
- ✚ Cada departamento es una localidad del sistema distribuido
- ✚ Las aplicaciones se puede generar en cualquier departamento (localidad)
- ✚ Por simplicidad se consideran, únicamente dos departamentos

Conjunto P de predicados simples:

$P = \{Dnum = 1, Job = 'Programador'\}$

$(Dnum = 2 \text{ es equivalente a } Dnum \neq 1)$

Términos de predicados:

$y_1: Dnum = 1 \wedge Job = 'Programador'$
 $y_2: Dnum = 1 \wedge Job \neq 'Programador'$
 $y_3: Dnum \neq 1 \wedge Job = 'Programador'$
 $y_4: Dnum \neq 1 \wedge Job \neq 'Programador'$

Todos los predicados son relevantes

Ejemplo de predicado no relevante

Salario > 5000

¿Por qué?

En este ejemplo, a partir de los datos que nos dan acerca de las aplicaciones que usarán los datos de la relación Empleado, sacamos el conjunto de predicados simples, $P = \{Dnum=1, Job="Programador"\}$, porque hay una aplicación que requiere información sobre los empleados de los departamentos (estar en $Dnum=2$ o $Dnum=3$, es lo mismo que $Dnum \neq 1$), y sobre los empleados que son programadores (de ahí el predicado $Job="Programador"$): A partir de este conjunto de predicados simples, sacamos $2^2=4$ conjunciones; cuatro términos de predicados. Observamos que los dos predicados simples de P son relevantes. Veámoslo: el predicado $Dnum=1$ es relevante porque existen dos términos de predicados que difieren en él: y_1 e y_3 , y lo mismo ocurre con $Job="Programador"$, pues y_1 e y_2 difieren solo en él.

Por otra parte, si quisieramos introducir otro predicado simple como $salario > 5000$, observamos que no sería relevante. Más que nada porque no es un dato que lo requieran las aplicaciones del enunciado. Por lo que no sirve de nada la fragmentación con este predicado (con él tendríamos 8 fragmentos en lugar de 4).

➤ Propiedades de un conjunto P de predicados simples

- **Completo:** si cualquier par de tuplas contenidas en el mismo fragmento son referenciadas con la misma probabilidad por cualquier aplicación.
- **Minimal:** si todos los predicados simples contenidos en él son relevantes.

Ejemplo	
$P = \{Dnum = 1\}$	Es minimal, pero no completo
$P = \{Dnum = 1, Job = 'Programador'\}$	Es completo y minimal
$P = \{Dnum = 1, Job = 'Programador', Salario > 5000\}$	Es completo, pero no minimal

En este ejemplo, el primero es minimal porque no hay predicados irrelevantes, pero no es completo: Supongamos que tenemos las siguientes dos tuplas:

1. **Tupla A:** { $Dnum = 1$, $Job = 'Programador'$, $Salario = 4000$ }
2. **Tupla B:** { $Dnum = 1$, $Job = 'Analista'$, $Salario = 4500$ }

Ambas tuplas tienen el mismo valor para el atributo $Dnum = 1$, pero difieren en Job y $Salario$. Si solo usamos el conjunto de predicados $P=\{Dnum=1\}$, no podemos distinguir entre la tupla A y la tupla B, ya que este conjunto de predicados no incluye información sobre el Job o el $Salario$.

Esto significa que cualquier aplicación que intente referenciar o identificar estas tuplas usando solo $P=\{Dnum=1\}$ no podrá hacerlo de manera precisa. No tendrá suficiente información para diferenciar entre la tupla A y la tupla B, lo cual causa que el conjunto de predicados P no sea completo.

● MIRAR EJEMPLOS DE LAS TRANSPARENCIAS 21-22-23

5.3 Fragmentación horizontal derivada

Se utiliza para facilitar el producto natural distribuido. Un **producto natural distribuido** es un producto natural entre relaciones fragmentadas horizontalmente.

Un producto natural entre dos relaciones globales R y S requiere comparar todas las tuplas de R con todas las tuplas de S. Si es **distribuido**, se requiere **comparar todos los fragmentos R_i de R con todos los fragmentos S_j de S**. A veces, se puede deducir que algunos productos parciales $R_i \bowtie S_j$ son intrínsecamente vacíos.

Se define el **grafo de un producto natural distribuido** como un grafo $G=(N,E)$, donde los nodos N representan fragmentos y las flechas no dirigidas E representan productos naturales no vacíos entre fragmentos. Puede ser **total**, donde están todas las flechas no dirigidas; todos los productos naturales posibles entre todos los fragmentos, o **reducido**, cuando al menos falta una flecha. Dentro de este grupo puede ser **particionado**, cuando el grafo está dividido en varios subgrafos independientes, o bien **simple**, donde cada subgrafo está formado por una sola flecha. Para que este tipo de grafo se pueda hacer es necesario que $\#R = \#S$.

Teorema

Sea R una relación global cuyos fragmentos R_i se derivan de la fragmentación de otra relación global S, es decir $R_i = R \bowtie S_i$. Si se verifican las condiciones de fragmentación completa y disjunta, entonces $R \bowtie S$ tiene un grafo de producto natural simple.

5.4 Fragmentación vertical

Consiste en agrupar atributos que son referenciados de la misma manera por las aplicaciones.

- **Correctitud:** cada atributo de R tiene que estar contenido en, al menos, un fragmento (completo) y cada fragmento debe incluir una llave de R o un “identificador de tupla”.
- **Propósito:** Identificar los fragmentos R_i tales que muchas aplicaciones puedan ejecutarse utilizando exáctamente uno de ellos.

A la hora de diseñar la fragmentación de una base de datos, pueden seguir las siguientes **heurísticas de fragmentación:**

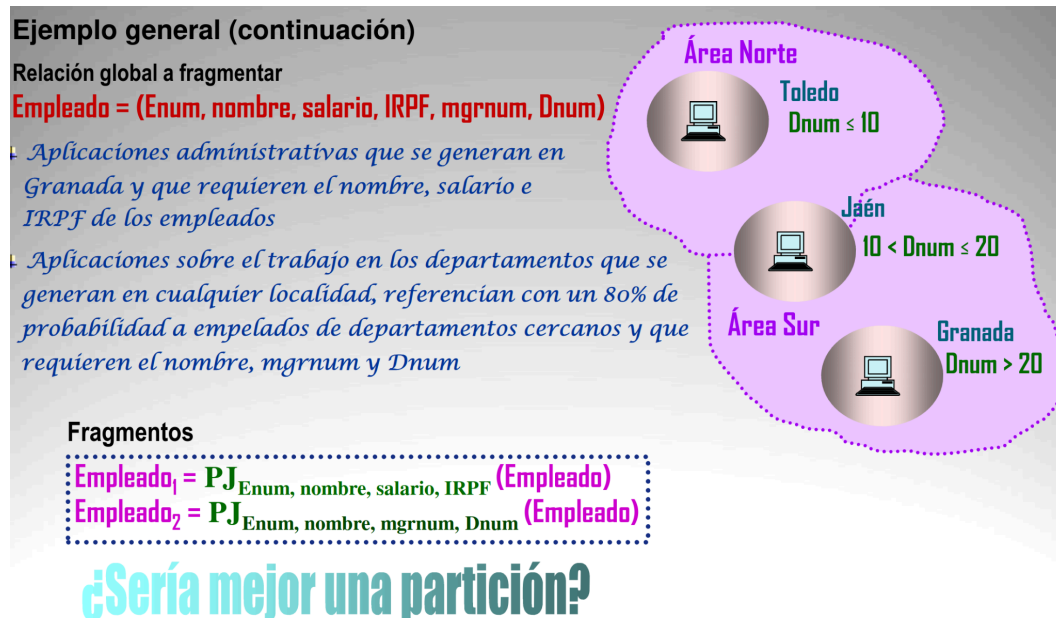
- **Escindir:** las relaciones globales son progresivamente divididas en fragmentos.
- **Agrupar:** los atributos son progresivamente añadidos para construir los fragmentos.

Distinguimos dos **tipos de fragmentación vertical:**

- **Partición:** conjunto de atributos disjuntos (aunque no totalmente disjuntos porque la llave tendrá que estar en todos los fragmentos).

- **Agrupación:** conjuntos de atributos que pueden solaparse. Esto introduce replicación, la cual es ventajosa para aplicaciones de sólo lectura y no conveniente para aplicaciones de actualización.

Ejemplo:



En este caso no sería mejor una partición porque si no el atributo nombre tendríamos que haber decidido dejarlo en un solo fragmento. Esto perjudica a alguna aplicación.

6. Criterios generales para la asignación de fragmentos

El diseño de la fragmentación, aunque se hace primero, está influenciado por el diseño de la asignación de los fragmentos. La asignación de fragmentos puede ser:

- **Asignación no redundante** (no vamos a tener copias de los fragmentos; sólo uno): se sigue una **aproximación “más adecuada”**, que consiste en que a cada posible asignación se le asocia una medida, seleccionando la localidad con mejor valor, es decir, vamos a determinar para cada posible asignación de un fragmento, un beneficio (medida), y seleccionaremos la de mayor beneficio.

¿Cuál es el **problema** de esta asignación? que descuida el efecto de asignar un fragmento en una localidad en la que existe otro fragmento relacionado.

- **Asignación redundante:** tenemos dos posibles aproximaciones:

- **Aproximación “todas las localidades beneficiosas”**: consiste en determinar el conjunto de localidades en las que el beneficio de asignar una copia es mayor que el coste, que viene dado en términos de las actualizaciones que se hacen a los datos.
- **Aproximación “replicación adicional”**: consiste en determinar la solución al problema no redundante, e introducir progresivamente copias hasta que ninguna réplica sea beneficiosa.

¿Qué **problemas** tiene esta asignación? Que el grado de replicación de un fragmento es una variable del problema, por lo que cuantas más réplicas, más aumenta la complejidad. Y que el hecho de modelar las aplicaciones de sólo lectura es más complicado porque si solo tenemos una copia, es más fácil porque solo está ese fragmento y las lecturas se hacen de ahí (y punto). Si tenemos varias copias, el sistema tiene que hacer cálculos para ver dónde están los cuellos de botella y redirigir las lecturas.

7. Medida del coste y beneficio de la asignación de fragmentos

En lo que sigue, vamos a seguir la siguiente notación:

Notación

- **i** es el índice del fragmento
- **j** es el índice de la localidad
- **k** es el índice de la aplicación
- **f_{kj}** frecuencia de activación de la aplicación k en la localidad j
- **r_{ki}** número de referencias que la aplicación k necesita realizar para recuperar datos del fragmento i
- **u_{ki}** número de referencias que la aplicación k necesita realizar para actualizar datos del fragmento i
- **$n_{ki} = r_{ki} + u_{ki}$**

7.1 Fragmentación horizontal

➤ Aproximación “más adecuada”:

$$B_{ij} = \sum_k f_{kj} \cdot n_{ki}$$

El beneficio de asignar el fragmento i a la localidad j viene dado por: El fragmento i se asigna a la localidad j en la que B_{ij} sea máximo.

➤ Aproximación “todas las localidades beneficiosas”

$$B_{ij} = \sum_k f_{kj} \cdot n_{ki} - C * \sum_k \sum_{j' \neq j} f_{kj'} \cdot u_{ki}$$

En este caso, para calcular el beneficio, tenemos en cuenta el coste de ir introduciendo copias (C es una constante que mide la proporción entre el coste de

actualizar y recuperar ($C \geq 1$). Observamos que en la expresión que va restando, solo se tienen en cuenta las referencias de actualización, porque en las copias es lo único que se propaga. El fragmento i se asigna a todas las localidades j en las que B_{ij} sea positivo. No introduciremos la copia que haga B_{ij} negativo. Si $B_{ij} = 0$, el coste y el beneficio son iguales, por lo que en principio, si no nos dan más datos, podemos hacer lo que queramos (introducir o no).

➤ Aproximación “replicación adicional”

$$B_{ij} = \sum_k f_{kj} \cdot n_{ki} - C * \sum_k \sum_{j' \neq j} f_{kj'} \cdot u_{ki} + \beta(d_i)$$

$$\beta(d_i) = (1 - 2^{1-d_i}) F_i$$

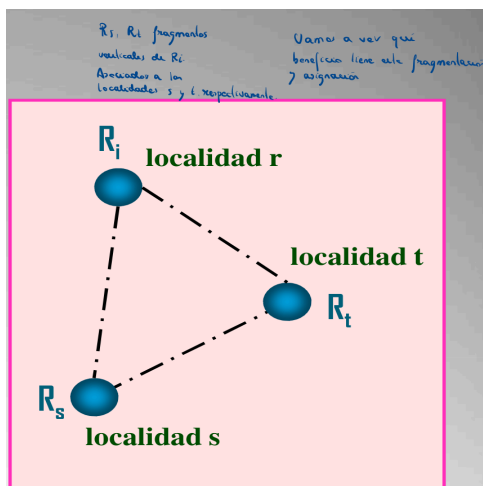
d_i = grado de replicación del fragmento i

F_i = beneficio de tener el fragmento i completamente replicado



En esta aproximación, el beneficio (beneficio de asignar una copia al fragmento i en términos del aumento de la fiabilidad y disponibilidad) varía con respecto al anterior en el hecho de que se ha introducido un factor de fiabilidad (cuantas más copias introducimos, más fiable se vuelve el sistema. Esto es beneficioso; por eso va sumando). Lo mismo que antes, paramos de introducir copias cuando $B_{ij} < 0$. Esta aproximación va a permitir introducir más copias por el factor beta que va sumando.

7.2 Fragmentación vertical



Supongamos que tenemos las aplicaciones A_s y A_t generadas en s o t , que usan, solamente atributos de R_s o $R_t \rightarrow$ evitan una referencia remota (o varias).

Si tenemos aplicaciones A_1 , antes locales a r , que usan solamente, atributos de R_s o $R_t \rightarrow$ añaden una referencia remota (antes de la fragmentación, estas aplicaciones eran completamente locales. Ahora tiene que hacer referencias remotas (al menos una)).

Si tenemos aplicaciones A_2 , antes locales a r , que usan atributos de R_s y de $R_t \rightarrow$ añaden dos referencias remotas.

Si tenemos A_3 , generadas en localidades diferentes a r, s y t , que usan atributos de R_s o $R_t \rightarrow$ añaden una referencia remota (antes de la fragmentación, hacían una referencia remota (a la localidad r)).

El **beneficio** de fragmentar verticalmente el fragmento i en los fragmentos s y t viene dado por:

$$B_{ist} = \sum_{k \in A_s} f_{ks} n_{ki} + \sum_{k \in A_t} f_{kt} n_{ki} - \sum_{k \in A_i} f_{kr} n_{ki} - 2 \sum_{k \in A_2} f_{ks} n_{kr} - \sum_{k \in A_3} \sum_{j \neq r, s, t} f_{kj} n_{ki}$$

Se realiza la fragmentación vertical de R_i si B_{ist} es positivo.