Almacenamiento y recuperación de objetos persistentes

En este apartado veremos las técnicas que los ODBMSs (*Object DataBase Management Systems*) emplean para almacenar objetos y gestionar el movimiento de estos objetos entre el almacenamiento permanente y la memoria de programa

Es esta un área en donde la tecnología de los ODBMSs difieren notablemente de la empleada en generaciones previas:

- Estos sistemas están íntimamente ligados al entorno de su lenguaje de programación.
- Este acoplamiento proporciona la ilusión de que sólo existe una jerarquía de memoria, aún cuando algunos de los objetos se encuentran en la memoria de programa, otros en áreas de almacenamiento permanente locales y algunos se localizan remotamente en algún lugar de la red.

No todos los sistemas ODBMSs comerciales emplean las mismas técnicas para almacenar los objetos: veremos algunas soluciones representativas. En este apartado trataremos con:

- Modelos de memoria y configuraciones cliente-servidor.
- Identificación y localización de objetos.
- Mejora del rendimiento a través de las técnicas de *caching*, *clustering* e *indexing*.
- Replicación de objetos.

La gestión del almacenamiento en los DBMSs relacionales

Terminología que vamos a emplear:

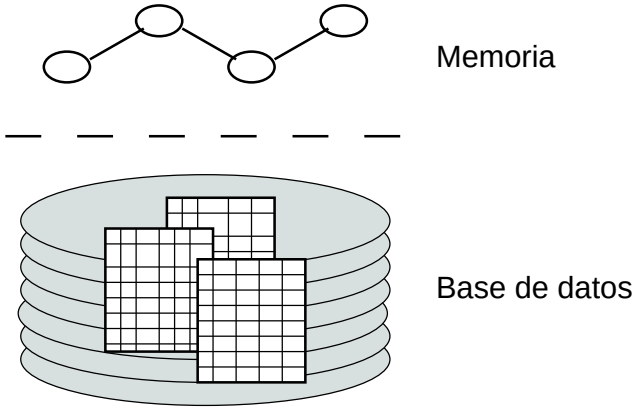
- El término "memoria" se refiere a la memoria principal o virtual directamente accesible desde el entorno de un lenguaje de programación (las variables de programas definidas en C++, C, COBOL, etc...).
- El término "almacenamiento" hace referencia a los dispositivos de almacenamiento secundario no volátiles. El almacenamiento es persistente y los objetos que se encuentran en el espacio de almacenamiento son objetos persistentes.

El modelo de almacenamiento de dos niveles

El modelo de almacenamiento empleado en los DBMSs tradicionales está constituido por al menos dos niveles:

1. Almacenamiento en dispositivos secundarios tales como discos.
2. Memoria de programa en áreas de memoria principal y virtual.

En ocasiones, se introduce un tercer nivel de apoyo para operaciones de archivado, aunque de momento no consideraremos este tipo.



En contraste el modelo de almacenamiento de un ODBMS presenta sólo un nivel, que unifica la memoria del programa y de la base de datos.

El modelo de almacenamiento de dos niveles ...

Las bases de datos convencionales tratan la memoria y el almacenamiento de forma diferente:

- La fuente primaria de los datos está en almacenamiento estable no volátil.
- La base de datos persiste independientemente de las aplicaciones de base de datos en ejecución.
- El DBMS almacena todas los accesos a los datos en ficheros de *log*, que utiliza para actualizar los datos almacenados como parte de las transacciones de validación (*commit*).
- Para mantener la consistencia, el DBMS emplea técnicas de gestión de transacciones y tolerancia a fallos. La recuperación de la consistencia cuando se produce un fallo implica recorrer el fichero de *log* deshaciendo actualizaciones de la transacciones incompletas y rehaciendo las transacciones que han sido validadas pero que no se han escrito en la copia primaria.
- La responsabilidad del DBMS es la protección de la base de datos en almacenamiento persistente.
- El DBMS no tiene responsabilidad sobre los datos transitorios, es decir, sobre los datos en memoria de programa.

El modelo de almacenamiento de dos niveles ...

Existe, por tanto, una separación bien establecida entre el espacio de variables de la base de datos y el espacio de variables del programa en memoria

Esta distinción se manifiesta de diferentes formas, pero:

Especialmente en los lenguajes y modelos usados por las aplicaciones para acceder a los datos en cada uno de estos espacios

Lenguajes y tipos de datos en los DBMSs relacionales

Los DBMSs relacionales proporcionan:

- Sus propios tipos para acceder a los datos almacenados en la base de datos.
- Su propio lenguaje (SQL) para acceder a la memoria de programa. El lenguaje SQL no sólo tiene una sintáxis propia para la formulación de comandos, sino que también tiene su propia noción de variable y operador.
- Las variables empleadas en los comandos SQL son variable de SQL, no variables de C++, C o COBOL, excepto en el caso de las variables de programa asignadas a variables SQL:

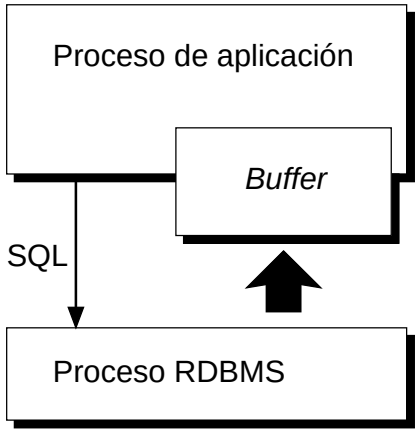
```
SELECT AVG emp.sal
FROM Employee emp
WHERE emp.level <= :next_level;
```

- `next_level` es una variable del programa, mientras que `sal` y `level` son variables de la base de datos.
- El valor de la variable `next_level` es comparado con el campo de la base de datos `emp.level`.
- Algunos operadores de SQL, como `AVG` son aplicables sólo a los campos de la base de datos y nunca a variables del programa.

El modelo de procesos en los DBMSs relacionales

Los DBMSs relacionales típicamente emplean un modelo de dos procesos:

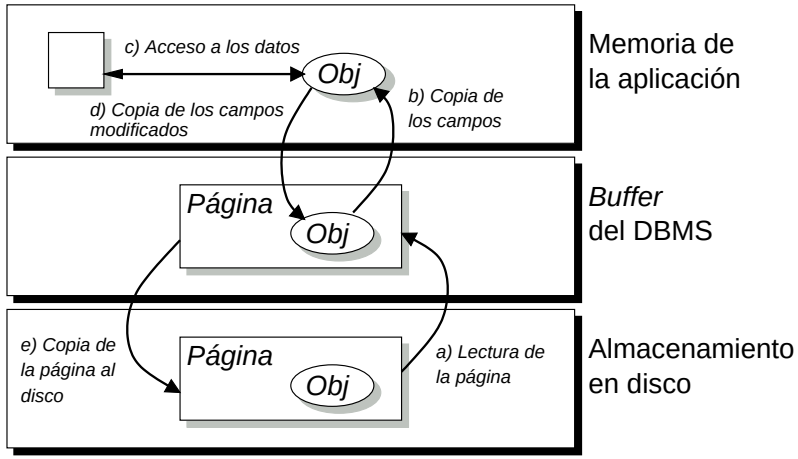
- El entorno de ejecución del DBMS está aislado en su propio proceso, independiente del proceso de aplicación.
- La interacción entre la aplicación y el DBMS requiere un mecanismo de comunicación entre procesos. Las llamadas entre procesos (IPCs, *Interproccess Calls*) son relativamente caras (del orden de 10 milisegundos) y comparables al tiempo que requiere un acceso a disco.



Procesos en los DBMSs relacionales ...

Analicemos la secuencia de eventos que tienen lugar cuando una aplicación lee y actualiza un registro de la base de datos.

Supongamos que la aplicación ejecuta una instrucción `SELECT` para encontrar a un empleado con un determinado nombre, que el resultado es único y que no se encuentra en una tabla temporal resultado de una consulta previa.



Procesos en los DBMSs relacionales ...

La secuencia de eventos que tiene lugar es:

- El DBMS relacional emplea índices o búsquedas a través de tablas para determinar que página del disco contiene el registro buscado.
- El DBMS lee la página de disco y la copia en una página de su propio espacio de direcciones principal. El conjunto de páginas reservadas constituye el *buffer* de la base de datos.
- El DBMS copia los campos de los registros de la base de datos en variables del programa.
- La aplicación puede ahora acceder directamente a las variables y llevar a cabo las operaciones que estime convenientes.
- Para actualizar los registros de la base de datos, la aplicación empleará un `UPDATE`. El DBMS copia los datos desde las variables del programa a los campos de un registro en el *buffer*.
- Cuando la transacción se valida (*commit*), el DBMS copia el registro actualizado al área correspondiente en almacenamiento secundario.

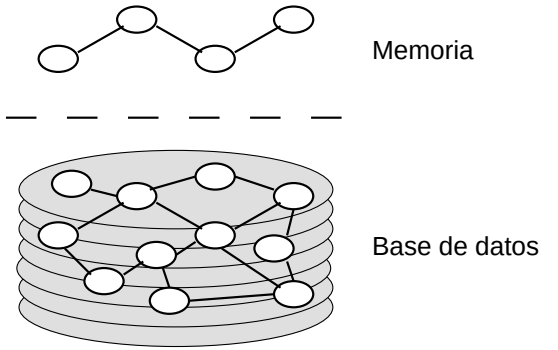
El acceso a un registro en disco pasando por el *buffer* de la base de datos y la copia hasta el espacio de direcciones de la aplicación requiere más de 10.000 instrucciones.

La gestión del almacenamiento en los DBMSs orientados a objetos

El modelo de memoria de un único nivel

Uno de los objetivos de los ODBMSs es simplificar la forma en que un programador accede al almacenamiento persistente proporcionando la ilusión de que existe una única jerarquía de memoria.

- Detrás de esta ilusión puede haber en realidad varios niveles de almacenamiento persistente.
- Sin embargo, el programador no necesita gestionar directamente el movimiento de objetos entre los diferentes niveles.
- La filosofía de diseño se basa en que es responsabilidad del DBMS saber cuando es necesario acceder al almacenamiento secundario.



El modelo de memoria de un único nivel ...

Cuando un programa C++ hace referencia a un objeto pueden suceder dos cosas:

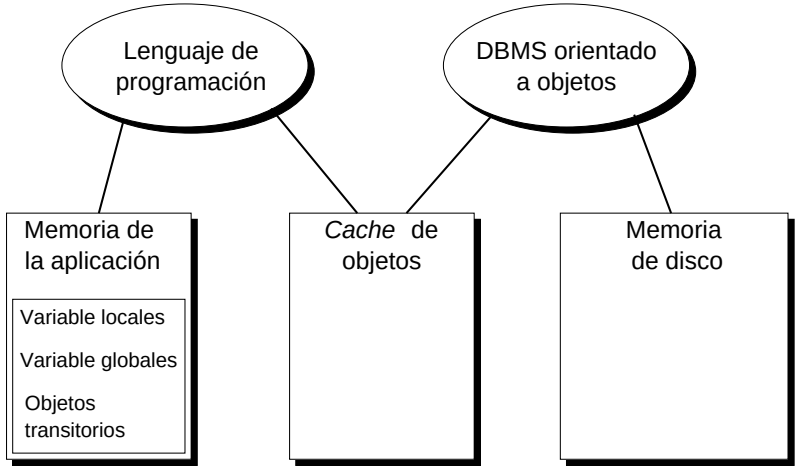
- Que el objeto ya se encuentre en el espacio de memoria del programa, en cuyo caso es accesible directamente.
- Que el objeto no se encuentre en dicho espacio. En este caso el ODBMS debe capturar la referencia, determinar dónde se encuentra almacenado el objeto y llevarlo hasta el espacio de memoria del programa.

Es interesante comparar este modelo de acceso con el estudiado en el apartado anterior, en donde el programador de bases de datos relacionales necesita saber cuando se requiere acceder a la base de datos y emplear SQL para ello.

El modelo de memoria de un único nivel ...

Es necesaria la cooperación entre el DBMS orientado a objetos y el entorno de programación para crear la ilusión de que sólo existe un nivel de memoria:

- Ambos comparten y pueden direccionar un espacio de memoria común que se conoce con el nombre de "*cache* de objetos".
- Esta *cache* de objetos se manifiesta al lenguaje de programación como otra clase de almacenamiento mientras que para el ODBMS es un *buffer* entre el almacenamiento secundario y la aplicación.

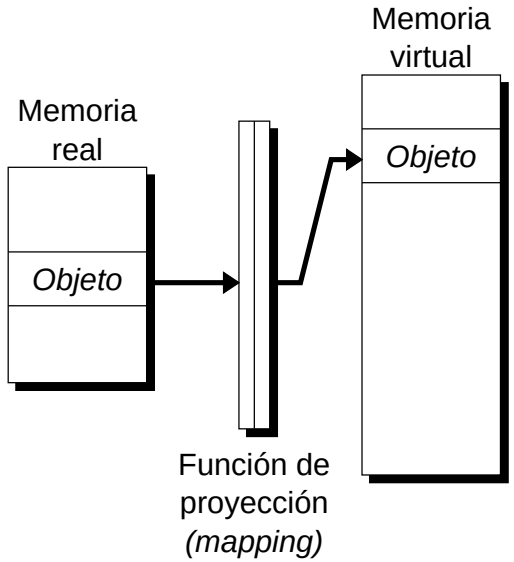


El modelo de memoria de un único nivel ...

- El modelo de memoria de un sólo nivel constituye la extensión natural del concepto de memoria virtual para incluir el almacenamiento persistente (que puede ser local o remoto).
- El concepto de memoria virtual ha evolucionado enormemente en los últimos 30 años. En los principios de la computación los programas estaban confinados al espacio ofrecido por la memoria central. El programador era responsable de determinar cómo segmentar el código de forma que sólo las porciones relevantes residieran en memoria en un momento dado además de coordinar la carga y descarga de esos segmentos durante la ejecución (esta técnica, conocida como *overlaying*, no resultaba trivial).
- A mediados de la década de 1970 se desarrolló el concepto de memoria virtual como una técnica para permitir que el espacio de direcciones sobrepasara el límite impuesto por la memoria central. De este modo, es posible dar cabida en el espacio de direcciones virtuales a un programa que excede la memoria física del sistema.

El modelo de memoria de un único nivel ...

- La memoria virtual se implementa mediante técnicas de translación de direcciones que permiten al programa direccionar el espacio de memoria virtual exactamente del mismo modo que se direcciona la memoria física.
- Un traductor de direcciones virtuales establece la equivalencia (*maps*) entre la dirección virtual y la dirección real.



El modelo de memoria de un único nivel ...

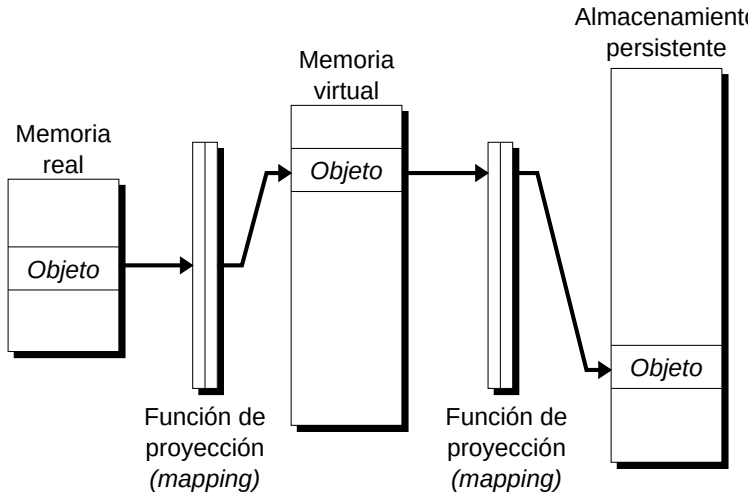
El espacio de memoria virtual normalmente se gestiona usando técnicas de paginado:

- La memoria real (o parte de ella) se particiona en bloques de tamaño fijo.
- La memoria virtual se organiza empleando bloques del mismo tamaño que los de la memoria real denominados páginas.
- El sistema operativo es responsable de detectar cuando un programa hace referencia a una página que no reside en memoria real (*page fault*), en cuyo caso debe leer la página pedida del almacenamiento secundario, asignarle un bloque de memoria real y continuar la ejecución del programa.

El modelo de memoria de un único nivel ...

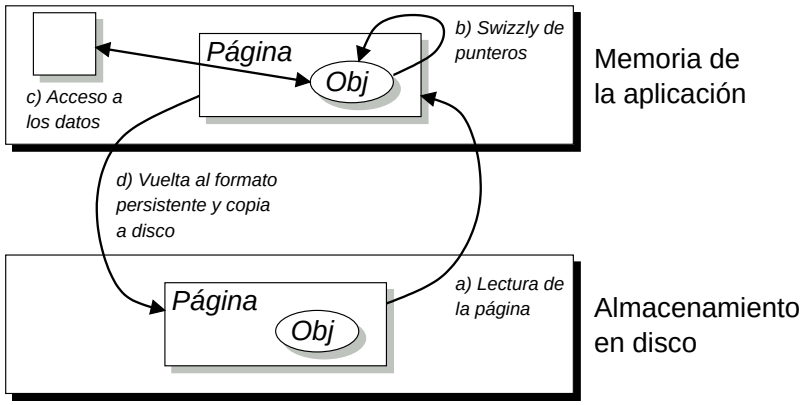
El ODBMS es responsable de mantener la equivalencia entre las páginas en almacenamiento secundario y el esquema de memoria virtual del sistema operativo.

En ese sentido, el esquema de memoria única de un DBMS orientado a objetos constituye un paso más en el concepto de memoria virtual, permitiendo la extensión a objetos en almacenamiento persistente



El modelo de memoria de un único nivel ...

¿Qué sucede cuando un programa intenta acceder a un objeto que no se encuentra en el espacio de memoria?



El modelo de memoria de un único nivel ...

La referencia al objeto causará un *page fault*. La secuencia de acciones es la siguiente:

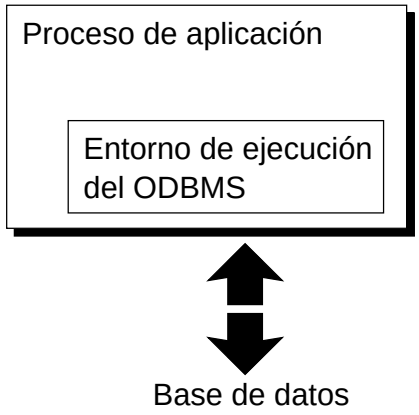
1. El ODBMS localiza la dirección de la página en el almacenamiento secundario y establece la equivalencia (*mapping*) con una página de memoria real en el espacio de direcciones de la aplicación.
2. Si el objeto contiene referencias a otros objetos, el ODBMS convierte esas referencias a direcciones de memoria virtual, de modo que la aplicación pueda utilizarlas como punteros del lenguaje de programación. Este proceso se denomina "*swizzling* de punteros".
3. La aplicación puede ahora direccionar directamente el objeto y llevar a cabo cualquier manipulación que se requiera.
4. Cuando el programa se completa o cuando el ODBMS necesita eliminar esa página para hacer hueco a una nueva página, el propio ODBMS convierte la página que contiene el objeto modificado de nuevo al formato de disco y lo copia en el almacenamiento secundario, reemplazando la versión original de esa página.

Si la aplicación hace referencia de nuevo al objeto, se producirá otro *page fault*, desencadenándose otra iteración de la secuencia de procesos descrita.

Implicaciones de la jerarquía única de memoria: el modelo mono-proceso

La jerarquía única de memoria no sólo integra el lenguaje de programación y la base de datos orientada a objetos, además afecta significativamente al rendimiento de las aplicaciones.

El entorno de ejecución del ODBMS está ligado a la aplicación y no se trata de un proceso independiente como en el caso de los DBMS relacionales. Los objetos leídos de disco son almacenados en el área *cache* compartida por el ODBMS y el lenguaje de programación.



El ODBMS y el lenguaje de programación comparten los mismos tipos de datos básicos, de modo que los objetos almacenados en la *cache* están en el formato del lenguaje de programación empleado.

. . . el modelo mono-proceso . . .

Mover un objeto desde la representación en disco a la representación utilizada por el lenguaje de programación requiere que el ODBMS lleve a cabo las siguientes operaciones:

1. *Swizzle* de las referencias inter-objetos. Es decir, convertir los punteros en disco a punteros en la memoria central.
2. Realizar cambios estructurales menores y entradas en las estructuras del lenguaje de programación.
3. Convertir la representación de los datos del objeto si éste proviene de un servidor con arquitectura diferente o de un compilador diferente.
4. La aplicación puede entonces acceder directamente al objeto.

Puesto que ambos, el lenguaje de programación y el ODBMS, pueden acceder directamente al objeto una vez que éste está en la *cache*, se evita la comunicación entre procesos que se requiere en el modelo bi-proceso utilizado por los sistemas relacionales.

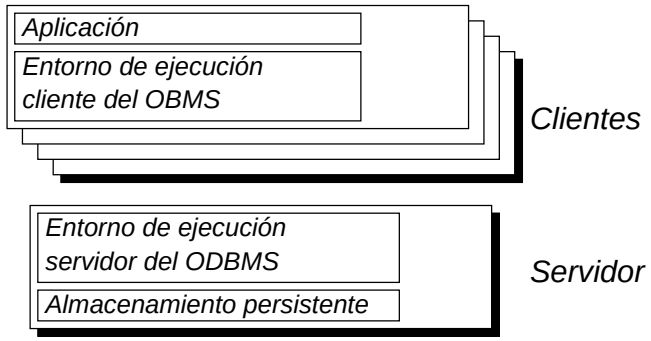
. . . el modelo mono-proceso . . .

En resumen:

- El ODBMS extiende la memoria virtual para incluir los datos almacenados en disco.
- De hecho, nada impide que este mismo esquema se extienda a la red de interconexión entre sistemas, de modo que el programa sea capaz de acceder a cualquier objeto almacenado en cualquier punto de la base de datos distribuida usando sólo el lenguaje de programación.
- Es el ODBMS quien gestiona el proceso de localizar y acceder a un objeto independientemente de si éste es local o remoto.

Configuraciones cliente-servidor en ODBMS

- Las bases de datos orientadas a objetos se utilizan normalmente en entornos de computación distribuidos.
- Una configuración distribuida relativamente simple es la constituida por un conjunto de máquinas cliente (estaciones de trabajo y ordenadores personales) y un nodo servidor interconectados mediante una red de área local.
- En esta configuración, las máquinas *desktop* son los nodos de aplicación mientras que el servidor actúa como el nodo de la base de datos.
- En general, las aplicaciones de ODBMSs se configuran ligando el código de la aplicación con el módulo cliente de la ODBMS en la parte del cliente. En el lado del servidor, es el módulo servidor de la ODBMS el que se liga al código del programa.



Configuraciones cliente-servidor ...

- La interacción entre los componentes cliente y servidor requiere algún tipo de comunicación (por ejemplo, un cliente puede necesitar acceder a un objeto que está almacenado en la base de datos del servidor).
- Esta comunicación normalmente se implementa como una llamada a un procedimiento remoto (*Remote Procedure Call*, RPC), que es mucho más costosa que una IPC.
- Sin embargo, una vez que el objeto ha sido transferido a la *cache* del cliente, no se requiere ningún IPC adicional cada vez que se accede al objeto.

Algunos desarrolladores de ODBMSs prefieren implementar en sus productos un esquema de funcionamiento con dos procesos (cliente/servidor):

- El objetivo es garantizar la protección de la integridad de la base de datos.
- Ya que la aplicación de base de datos y el ODBMS comparten el mismo espacio de direcciones, hay una posibilidad de que la aplicación lleve a cabo algunos cambios sin el control del ODBMS.
- El precio, por supuesto, es un impacto negativo en el rendimiento debido al uso de IPCs.

Acceso a objetos persistentes

Identificación de objetos

- Enlazar el entorno de ejecución cliente del ODBMS con la aplicación de base de datos requiere que éste maneje el acceso a los objetos individuales de una forma compatible con la aproximación utilizada por el lenguaje de programación utilizado.
- Un aspecto importante para alcanzar esta compatibilidad es la utilización de identificadores de objetos (OIDs), que se utilizan para identificar y localizar unívocamente a los objetos.

Tanto el lenguaje de programación como el ODBMS saben como manejar estos identificadores de objetos (que además constituyen la dirección del objeto dentro de la base de datos).

Normalmente los lenguajes de programación orientados a objetos utilizan la dirección de memoria virtual como identificador del objeto. Una aproximación simple consiste en utilizar como identificador lógico del objeto su dirección de memoria virtual. Ya que dos objetos no se pueden almacenar en la misma dirección de memoria, la unicidad del identificador está garantizada. La referencia de un objeto a otro se implementa como un puntero, cuyo valor es la dirección en memoria virtual del objeto referenciado.

Identificación de objetos ...

En algunas bases de datos orientadas a objetos, esta aproximación no es satisfactoria fundamentalmente por dos motivos:

1. El espacio de direcciones resultante (generalmente de 32 bits) no es lo suficientemente grande para incluir los entornos de red con objetos distribuidos.
2. Una relación tan estrecha entre identificador lógico y dirección virtual impide al ODBMS mover objetos de forma eficiente y localizar y reparar todas las referencias a esos objetos.

Por otra parte, esta aproximación presenta la ventaja de que una vez que se le ha asignado al objeto una dirección de memoria virtual, las futuras referencias al objeto pueden llevarse a cabo con rendimientos excelentes.

La técnica utilizada por el ODBMS para implementar los identificadores de objetos es un factor determinante del rendimiento y la escalabilidad de la base de datos.

- Por un lado, si la técnica de implementación esta "próxima" a las direcciones virtuales, se mejora el acceso al objeto.
- Por otro lado, la introducción de un cierto nivel de indirección mejora la escalabilidad del sistema. Un ODBMS debe ser capaz de anticipar la necesidad de manipular poblaciones crecientes de objetos.

Identificación de objetos ...

Algunos ODBMSs emplean direccionamiento de 64-bits en lugar del esquema de 32-bits empleado por el lenguaje de programación.

- Una aproximación es utilizar los bits de mayor orden (por ejemplo, los 16 primeros) para representar el lugar de origen del objeto y emplear los bits de menor orden (los 48 restantes) para representar un número identificativo único.
- Otra alternativa consiste en incorporar en el identificador la fecha de creación del objeto, aunque en un entorno distribuido esto puede dar a lugar a problemas al requerir la sincronización de relojes entre los diferentes nodos.

Aunque el direccionamiento de 64-bits puede acomodar a un gran número de objetos, puede no ser suficiente para cubrir las necesidades de una base de datos distribuida. Por ejemplo, pueden surgir problemas cuando se interconectan dos redes con distinto espacio de direcciones (esto puede ocurrir cuando una organización compra a otra).

Identificación de objetos: Un ejemplo

- Objetivity/DB garantiza la integridad de los objetos utilizando referencias a objetos en lugar de punteros directos para acceder a los objetos.
- Esta características estructural fundamental sigue las directrices del ODMG y protege a la base de datos de los problemas de corrupción serios que puede causar una aplicación errática.
- Cualquier transacción en Objetivity/DB puede acceder transparente y simultáneamente a todos los datos de una base de datos a través de referencias a objetos de 64-bits. En arquitecturas de 64-bits, esto permite acceder a unos 10^{19} bytes (10 millones de terabytes) de datos y a unos 10^{17} objetos. En máquinas de 32-bits, debido a limitaciones en el sistema de archivos, sólo es posible acceder a uno 100 terabytes de datos y 10^{13} objetos.
- En el nivel más bajo, Objetivity/DB accede a los objetos persistentes a través de identificadores de objetos (OIDs) de 64-bits. Cada objeto en un una base de datos tienen un OID único que permite al ODBMS localizar y manejar el objeto de forma flexible y segura. Los OIDs contienen información sobre la base de datos, el contenedor, la página y el *slot* en donde se localiza el objeto.

Identificación de objetos: Un ejemplo

- El modelo físico es un reflejo del modelo lógico.
- Existe un único espacio lógico de direcciones soportado por un espacio físico estructurado jerárquicamente.
- Esto proporciona un medio eficiente y seguro para almacenar y manipular objetos de todos los tamaños.

El OID se forma a partir de cuatro componentes del siguiente modo:

- Los primeros 16-bits constituyen el identificador lógico de la base de datos.
- Los 16-bits siguientes son el identificador lógico del contenedor (1 bit para uso interno y 15 bits para el número del contenedor).
- Otros 16-bits determinan la página lógica dentro del contenedor.
- Los últimos 16-bits indican el *slot* lógico dentro de la página.

Acceso a objetos

- Normalmente, un programa C++ o SmallTalk accede a un objeto a través de un nombre de variable, un puntero o una referencia al objeto.
- Los objetos contenidos en la base de datos son accedidos de modo similar siguiendo las referencias a través de la base de datos o bien empleando un método más tradicional, mediante una consulta (*query*).

Aquí trataremos el acceso a objetos mediante referencia

El ODMG especifica una clase prototipo, `Ref`, que proporciona punteros inteligentes. `Ref` actúa en muchos aspectos como un puntero de C++ pero proporciona mecanismos adicionales para garantizar la integridad de las referencias a almacenamiento permanente.

Un ejemplo:

```
Ref <Ciudad> ciudad_origen;
```

declara `ciudad_origen` como una referencia a un objeto de tipo `Ciudad`. Aunque la sintáxis empleada para declarar la referencia a la base de datos es diferente a la empleada para declarar un puntero C++, el modo de empleo es formalmente igual.

Acceso a objetos ...

- El ODBMS implementa el `Ref` proyectando (`mapping`) un identificador de objeto (que es una dirección de la base de datos) en una dirección de la memoria física.
- Los ODBMSs emplean diferentes técnicas para llevar a cabo esta proyección.
- Sea cual sea la técnica empleada, cuando un objeto referencia a cualquier otro objeto, el ODBMS debe localizar ese segundo objeto y llevarlo hasta la *cache*, en donde podrá ser accedido directamente por el entorno de ejecución del lenguaje de programación orientado a objetos.
- Las referencias entre objetos almacenados en disco se representan mediante direcciones en disco. Esas direcciones no son utilizables por C++, que sólo entiende direcciones de memoria virtual.
- El ODBMS debe cambiar esas direcciones por referencias que puedan ser procesadas por C++.
- La técnica empleada por los ODBMSs para cambiar las referencias a objetos por punteros en memoria virtual se denomina *swizzling*. Cuando un ODBMS recupera un objeto puede llevar a cabo el *swizzle* de punteros de una sola vez o incrementalmente. Si el proceso tiene lugar de una sola vez se suele denominar *eager swizzling*.

Acceso a objetos ...

El ODBMS no sólo tiene que transformar en un puntero la referencia del objeto recuperado, sino que:

- También debe analizar cualquier referencia que puede hacer ese objeto a otros objetos.
- Recuperarlos de la base de datos y realizar el correspondiente *swizzle* de punteros en cada uno de ellos.

El resultado de este proceso es que tras la primera recuperación del objeto, C++ accede a los objetos persistentes del mismo modo que accede a los objetos transitorios.

Acceso a objetos ...

Un ejemplo:

- Supongamos un objeto persistente A que referencia a otro objeto persistente B. Supongamos que la página que contiene a A ya está accesible para el programa.
- El proceso de *swizzle* trata todos y cada uno de los punteros en la página de A, asignando y reservando una página de memoria virtual para la página de almacenamiento persistente que contiene al objeto apuntado.
- Si la aplicación necesita ahora dereferenciar la referencia de A a B, ésta ya ha sido *swizzled* y es ahora un puntero a una página de memoria reservada.
- Ya que el ODBMS todavía no ha "entrado" en la página que contiene a B, tiene lugar un *page fault* y el ODBMS carga la página que contiene B en su espacio de memoria previamente asignado.

La aplicación de base de datos sólo tiene que manejar direcciones de memoria virtual, ya que el ODBMS reserva con antelación las páginas que contienen todos los objetos que pueden ser referenciados.

Las referencias a los objetos, ya sean persistentes o transitorios, se llevan a cabo de forma uniforme

Factores de rendimiento

Además de la gestión de punteros y la comunicación entre procesos, muchos son los factores que afectan al rendimiento de una aplicación de base de datos.

El programador de aplicaciones de base de datos no desea que el acceso a la base de datos ralentice la ejecución: las expectativas se centran en que la combinación ODBMS–C++ funcione a velocidades del orden del C++.

Este objetivo es imposible de alcanzar ya que el acceso a almacenamiento secundario siempre será más lento que el acceso a memoria principal.

Existen algunos factores que el desarrollador de aplicaciones puede tener en cuenta para conseguir tiempos de acceso lo más bajo posible, entre los que se cuentan:

- Tamaño de las *caches*.
- *Clustering*.
- Indexación.
- Replicación.

Tamaño de la memoria *cache*

La *cache* del cliente está constituida por el espacio (*buffer*) de objetos accedidos y referenciados por la aplicación

- Los desarrolladores de bases de datos desarrollaron hace tiempo el concepto de *working space*: conjunto de datos que una aplicación utiliza durante un cierto periodo de tiempo.
- Muchas aplicaciones presentan una gran localidad (su *working set* cambia muy lentamente una vez realizada la carga inicial del espacio).
- Si la *cache* es suficientemente grande como para almacenar el *working set* de objetos, es posible minimizar los accesos a disco.
- En estas circunstancias, los accesos a disco son menores que los accesos a memoria principal y a *cache*, por lo que la técnica de *caching* proporciona un excelente rendimiento.

Tamaño de la memoria *cache* ...

El mejor rendimiento posible se alcanzaría si la aplicación realizara un único acceso. Esto se podría conseguir si la ODBMS proporcionara a la *cache* de una vez todos los objetos usados por la aplicación.

Sin embargo, esta aproximación presenta dos limitaciones:

- Conocer *a priori* qué objetos utilizará la aplicación.
- Disponer de una *cache* lo suficientemente grande como para acomodar a todos esos objetos.

Una posible solución:

- Algunos ODBMSs proporcionan a las aplicaciones facilidades para especificar qué objetos van a necesitar. Por ejemplo, algunos ODBMSs permiten el acceso a contenedores de objetos, que están constituidos por otros objetos. El contenedor de objetos entero se transfiere de una sola vez a la *cache*. Los contenedores de objetos son utilizados ampliamente en algunos tipos de aplicaciones, como CAD.

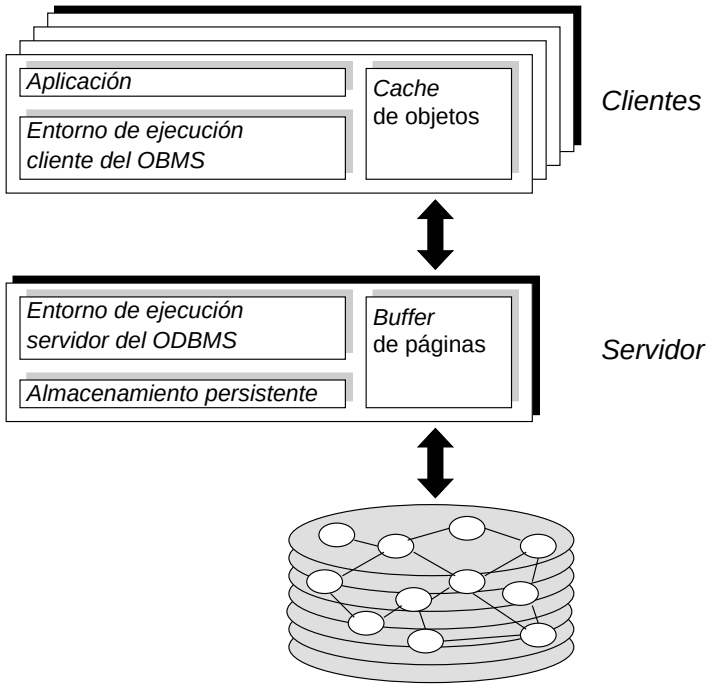
Tamaño de la memoria *cache* ...

En muchas aplicaciones interactivas, sin embargo, es imposible predecir qué objetos específicos serán accedidos por una determinada unidad del programa en ejecución.

- En estos casos, el ODBMS debe transferir los objetos a la *cache* de objetos a medida que estos son accedidos.
- Si son necesarios muchos objetos, el hecho de generar un *page fault* cada vez que se requiere un acceso puede ser muy ineficiente.
- Para soslayar este problema, cuando un objeto es accedido por una aplicación el ODBMS puede intentar predecir qué otros objetos se necesitarán en breve. El proceso de *swizzle* de punteros comentado anteriormente puede llevar a cabo esta tarea al hacer referencia a páginas que contienen objetos a los que todavía no ha hecho referencia la aplicación.
- En este sentido, los desarrolladores que utilicen un ODBMS debe asegurarse de que las *caches* son lo suficientemente grandes como para acomodar estas páginas.

Técnicas de *clustering*

Normalmente, los ODBMSs utilizan una técnica de doble *buffer* constituida por una *cache* de objetos en el lado del cliente y un *buffer* de páginas en el lado del servidor (ver figura).



Técnicas de *clustering*

Es necesario tener en cuenta, no sólo el tamaño de la *cache* de objetos y del *buffer* de páginas, sino también al tráfico entre el disco y el *buffer* de páginas.

Cuando un ODBMS accede a un objeto en disco, también copia el resto de los objetos contenidos en la misma página al *buffer* de páginas. Esto es debido a que la página es la unidad de transferencia entre el disco y el *buffer* de páginas. Parece lógico pensar que es posible minimizar el número de *page faults* si almacenamos en esta misma página aquellos objetos que es probable que la aplicación utilice.

Esta técnica, consistente en almacenar juntos aquellos objetos que es probable que se vayan a utilizar juntos, se denomina *clustering*. Un ODBMS que implemente una buena técnica de *clustering* será capaz de acceder a los objetos que se utilizan juntos con un número mínimo de *page faults*

Técnicas de *clustering*

Si los objetos no están bien agrupados puede tener lugar una gran actividad de paginado cuando se accede a un objeto. Una buena implementación de las técnicas de *clustering* puede dar lugar a un aumento considerable del rendimiento de las aplicaciones.

- Algunos ODBMSs permiten a la aplicación especificar qué objetos deben agruparse. Normalmente, esto se lleva a cabo permitiendo al programador indicar que un objeto debe almacenarse lo más próximo posible a otro objeto especificado (o al menos en el mismo segmento de la base de datos).
- Otros ODBMSs proporcionan agrupamiento por clases: intentan almacenar juntas todas las instancias de una determinada clase. Las técnicas de *clustering* basadas en la clase no son apropiadas en aplicaciones que necesitan recorrer relaciones entre objetos pero pueden ser apropiadas en aplicaciones que realizan *queries* sobre una clase.

  Las técnicas de *clustering* de clases son similares a las que emplean algunos DBMSs relacionales. Las técnicas de *clustering* son especialmente apropiadas en los RDBMSs debido a la orientación tabular de los operadores relacionales. Algunos sistemas relacionales incluso almacenan todas las filas de una tabla en un fichero separado dedicado a esa tabla.

Reubicación de objetos y compactación de huecos

En muchas ocasiones, un ODBMS debe ser capaz de mover objetos en almacenamiento para mejorar la agrupación de objetos.

- Supongamos una colección de objetos que están almacenados en una determinada página.
- A lo largo del tiempo, la aplicación de base de datos borrará y añadirá nuevos objetos.
- Cuando se produce el borrado de un objeto, se crea en la página un hueco sin utilizar. El ODBMS puede reutilizar este hueco para almacenar cualquier otro objeto siempre y cuando se ajuste al espacio disponible.
- Debido a este mecanismo, los huecos en una página tienden gradualmente a hacerse menores, aumentando la probabilidad de que un nuevo objeto no tenga cabida en el espacio disponible.

Reubicación de objetos y compactación de huecos

Las técnicas convencionales utilizadas por los DBMSs para manejar el almacenamiento incluyen la consolidación de huecos en un área de espacio libre mayor.

Un ODBMS debe implementar esta misma funcionalidad, es decir, ser capaz de mover los objetos dentro de la página que es la técnica empleada para consolidar huecos.

Sin la capacidad de consolidación de espacio libre, una transacción que inicialmente requería un único acceso a disco puede, con el paso del tiempo, acabar requiriendo múltiples accesos. Por este motivo, un ODBMS que mantiene un rendimiento alto sostenido requiere un sistema de gestión del almacenamiento bastante inteligente.

- La consolidación de huecos puede resultar un proceso extremadamente costoso si los OIDs se implementan como direcciones de memoria virtual. Si las referencias al objeto son físicas en lugar de lógicas, el ODBMS debe fijar todas las referencias de modo que apunten a la nueva posición del objeto relocalizado. Este proceso requiere, generalmente, poner *off-line* la base de datos.
- Por otra parte, las referencias lógicas siguen siendo válidas aunque el objeto haya cambiado de emplazamiento. Los ODBMSs que implementan referencias lógicas pueden consolidar espacios *on-line*.

Indexación

- Las aplicaciones de bases de datos no sólo acceden a los objetos mediante referencias. En ocasiones, el acceso a los datos se lleva a cabo mediante *queries* basadas en el valor de alguno de los atributos del objeto.
- El rendimiento de este tipo de accesos puede ser mejorado drásticamente si el ODBMS implementa índices sobre esos atributos.

Un índice puede ser, desde una simple tabla de pares de valores atributo-dirección, hasta una compleja estructura ramificada optimizada para manejar eficientemente grandes volúmenes de objetos:

- A lo largo de los años, los DBMSs relacionales han desarrollado un gran número de técnicas de indexación, muchas de las cuales son aplicables a los ODBMSs.
- Sin embargo, también se han desarrollado nuevas técnicas de indexación de objetos, que presentan una mayor complejidad estructural que las tablas del modelo relacional.

Indexación ...

Los patrones de acceso son el factor principal que determina qué atributos deben ser indexados. Existen varios criterios que permiten decidir cuando un atributo no debe ser indexado:

1. No tiene sentido indexar un atributo que sólo se utiliza raramente en los *queries* de las aplicaciones de bases de datos.
2. Indexar un atributo qur toma el mismo valor en un gran porcentaje de los elementos de una clase no es adecuado, puesto que no va a tener capacidad discriminante.
3. No debería indexarse un atributo que las aplicaciones modifican frecuentemente ya que el ODBMS deberá también actualizar el índice.

A la hora de decidir si un atributo debe indexarse, hay que considerar el compromiso que existe entre la velocidad de respuesta y la sobrecarga de trabajo que supone mantener y actualizar el índice.

La construcción y el mantenimiento de las estructuras de índices es responsabilidad del ODBMS. Es el gestor quien determina qué atributos deben indexarse. Algunos ODBMSs pueden construir índices dinámicamente basándose en los patrones de acceso detectados sin intervención explícita del administrador.

Replicación de objetos

Definiciones:

La replicación de objetos es otra de las técnicas de gestión de almacenamiento que los desarrolladores pueden utilizar para mejorar el rendimiento. Formalmente, un objeto replicado es aquel para el que existe más de una implementación (más de una copia).

- Los elementos individuales en el conjunto de implementaciones se denominan "réplicas" y el conjunto de réplicas de un objeto dado se denomina "conjunto de replicación".
- El número de elementos de un conjunto de replicación es el "grado de replicación del objeto".
- Los elementos de un conjunto de replicación son implementaciones físicas de un objeto lógico. Puede haber diversos conjuntos de replicación en un sistema, cada uno representando a un objeto dado y con un grado diferente.
- Se dice que un objeto está completamente replicado si existe al menos un elemento de su conjunto de replicación en cada nodo (*site*) de la red en el que el objeto puede ser accedido.
- En el extremo opuesto, un conjunto de replicación mínimo es aquel que contiene sólo dos elementos.

Replicación de objetos ...

El grado apropiado de replicación viene determinado por varios factores:

- Los patrones de acceso del objeto.
- La frecuencia relativa de las actividades de actualización y acceso.
- La fiabilidad de la red.
- La fiabilidad de los equipos y sistemas de almacenamiento secundario.
- El tiempo de respuesta requerido, etc...

Cuando una aplicación de base de datos hace referencia a un objeto replicado, el ODBMS debe escoger cual de las réplicas existentes es la que se va a acceder.

En general, la aplicación debe desentenderse completamente de cual es la réplica a la que está accediendo realmente en un momento dado. Ni tan sólo es necesario que conozca si un objeto al que está accediendo está replicado o tiene una implementación única. En el caso ideal, la gestión de réplicas es una función del sistema y nunca de la aplicación de base de datos.

Replicación de objetos ...

- El término "réplica" puede inducir una cierta ambigüedad. De acuerdo con su definición en el diccionario, réplica es una "copia de una obra artística que reproduce con igualdad la original". En un ODBMS, puede distinguirse o no entre el original y el resto de las copias. Más aún, el ODBMS puede tratar todas las implementaciones de un objeto como iguales o puede designar una copia primaria.
- No hay un estándar que defina como un ODBMS debe gestionar los identificadores de los objetos replicados. Una aproximación razonable consiste en emplear el mismo OID para todas las réplicas del mismo objeto. Esto proporciona la mayor flexibilidad a la base de datos. Si las réplicas no tuvieran el mismo OID, podría resultar difícil para el ODBMS determinar la existencia de copias de un objeto determinado.
- No todas las réplicas de un objeto deben ser necesariamente idénticas. Un ejemplo simple es el caso en el que las réplicas se almacenan en máquinas con diferentes arquitecturas. La implementación de las réplicas debe ser necesariamente diferente al nivel de implementación más bajo. Otro ejemplo lo tenemos cuando un objeto es replicado tanto como un objeto C++ como Smalltalk.

Replicación de objetos ...

Réplicas locales y distribuidas:

Son muchas las posibilidades que se plantean a la hora de decidir como y en que extensión distribuir las réplicas.

- En algunas circunstancias, el esquema más apropiado consiste en almacenar las copias en el mismo nodo de proceso pero en diferentes volúmenes, por ejemplo porque el ODBMS emplea *disk mirroring*. En este caso, el ODBMS almacena cada objeto de la base de datos en un disco primario y en un disco secundario que es una imagen exacta del primario. El ODBMS deberá ser capaz, además, de procesar cada actualización en ambos volúmenes.
- En aplicaciones distribuidas parece más apropiado almacenar las réplicas en diferentes nodos procesadores. Esta aproximación puede ser adecuada tanto en entornos de red local (*local-area*) como en redes extensas (*wide-area*). Algunos ODBMSs soportan bases de datos jerárquicas, permitiendo al usuario tomar objetos de una base de datos compartida ubicada en un servidor y copiarlos en la base de datos local. La réplica tiene el mismo OID que el objeto original en la base de datos principal.

Replicación de objetos ...

Disponibilidad y autonomía:

- Uno de los objetivos de replicar objetos es mejorar el rendimiento. La replicación puede ser especialmente útil en redes *wide-area*, en donde el tiempo de transmisión pueden afectar notablemente los tiempos de respuesta: evitar los accesos a través de la red conlleva la mejora del rendimiento. Sin embargo, esto puede implicar una replicación masiva.
- Otro de los objetivos de la replicación es aumentar la disponibilidad de los datos. La disponibilidad máxima se consigue cuando cada aplicación que necesita acceso a un objeto mantiene su propia copia de dicho objeto. En general, la replicación mejora la velocidad de acceso a los datos pero introduce la necesidad de procesar información adicional cuando se actualizan los objetos.
- Un tercer objetivo es la mejora de la autonomía. La autonomía total se consigue cuando el funcionamiento de cualquier nodo (*site*) no se ve afectado por el estado del resto de los nodos. Si un objeto se encuentra almacenado en un único nodo y por alguna razón deja de ser accesible, cualquier otro nodo que necesite ese dato resultará afectado. La existencia de copias alternativas proporciona la oportunidad al sistema de responder adecuadamente a las interrupciones.

Replicación de objetos ...

Factores asociados a la gestión de réplicas:

Son muchos los factores que hay que considerar en la gestión de los objetos replicados:

- Determinar el grado apropiado de replicación y decidir donde deben almacenarse las réplicas.
- Sincronizar la actualización de las réplicas, de modo que los objetos sean lo suficientemente actuales como para adaptarse a las necesidades de las aplicaciones.
- Determinar qué replica acceder de modo que tanto el rendimiento como la actualidad de los datos empleados sean adecuados.

No todos los objetos de un sistema requieren el mismo grado de replicación.

- Puede ser adecuado replicar completamente ciertos objetos mientras que otros no es necesario replicarlos en absoluto.
- En ocasiones, puede resultar adecuado que sean las aplicaciones o los usuarios los que soliciten la replicación de un objeto.
- En otros casos, debería ser el mismo ODBMS quien determine el grado apropiado de replicación basándose en los patrones de acceso de un objeto.

Replicación de objetos ...

Es necesario establecer un equilibrio entre la replicación de objetos y el esfuerzo requerido para mantener la coherencia:

- La replicación incrementa la probabilidad de acceder a objetos locales, reduciendo las transmisiones y mejorando el rendimiento.
- La replicación mejora la disponibilidad y la autonomía y reduce la dependencia de los objetos almacenados en otros nodos.
- Reducir la replicación disminuye la necesidad de sincronizar las actualizaciones, aumentando por tanto el rendimiento.
- Cuando aumenta la relación entre las actividades de actualización y las de consulta, aumentan las necesidades de proceso y transmisión para mantener la sincronización.
- Si el número de fuentes de actualización es grande, aumentan las necesidades de proceso y transmisión.
- Designar un elemento del conjunto de replicación como maestro para la actualización proporciona un punto de sincronización y simplifica este proceso.
- Cuanto mayor es la necesidad de datos actualizados, mayor es la frecuencia con la que dede sincronizarse.

Replicación de objetos ...

Entre los parámetros que deben tenerse en consideración a la hora de determinar un esquema de replicación y una distribución de objetos apropiada se incluyen:

- Los patrones de acceso a los objetos.
- La frecuencia relativa de los actividades de consulta y actualización.
- La tolerancia del sistema a los objetos casi-actuales frente a la necesidad de una sincronización instantánea.
- Fiabilidad del entorno, en donde la replicación proporciona una especie de copia de respaldo.
- Las necesidades de rendimiento, teniendo en cuenta la capacidad de transmisión de la red y su velocidad.

Debido a la necesidad de establecer este balance, no existe una solución única para la gestión de la replicación y distribución que sea adecuada para todos los sistemas distribuidos. En general, es necesario habilitar más de un mecanismos y varias políticas para replicar el sistema de forma adecuada.

Replicación de objetos ...

En la mayoría de los casos, las aplicaciones deben gestionar sus propias réplicas y políticas de distribución. Esto es especialmente cierto en aquellas situaciones en las que los objetos individuales tienen diferentes grados de replicación.

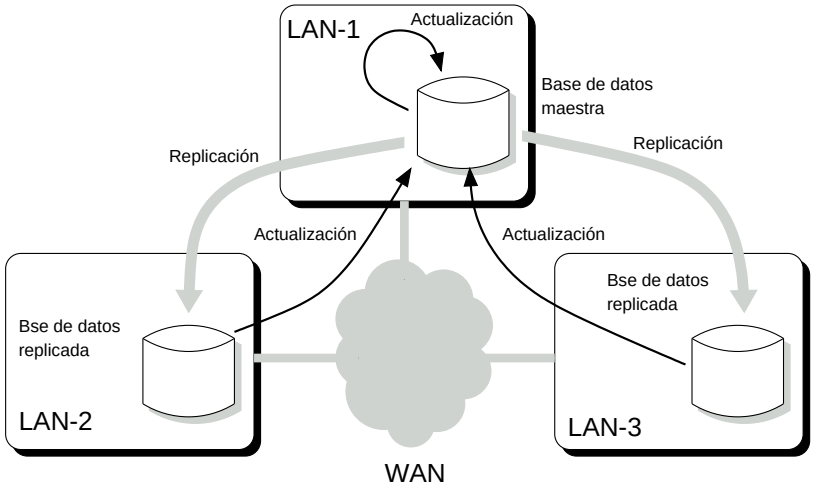
- Son las aplicaciones las que deben determinar donde residen las réplicas, cual de las réplicas disponibles debe utilizarse en las operaciones de consulta y actualización y cómo sincronizar las actualizaciones.
- Algunos ODBMSs proporcionan métodos de comparación para determinar si dos objetos son realmente el mismo (tienen la misma estructura y contenido). Esta funcionalidad es importante en aquellos casos en donde las réplicas gestionadas por la aplicación tienen diferentes OIDs asignados por el ODBMS, que no reconoce las réplicas como el mismo objeto.

Replicación de objetos ...

Como ejemplo, consideremos un sistema constituido por tres LANs. Un diseño apropiado podría consistir en replicar completamente la base de datos de objetos en cada una de las tres LANs.

- Cada base de datos será local en su LAN, minimizando el efecto de la comunicación a través de la WAN. Dentro de este esquema, los desarrolladores deberán determinar como actuar cuando uno de los objetos es actualizado. La gestión se simplifica notablemente si designamos a uno de los tres *sites* como master, de modo que todas las actualizaciones sean procesadas en dicho *site*.
- Dependiendo del volumen de actualizaciones, el nodo maestro puede transmitir copias completas de la base de datos o bien descargar lotes de transacciones para mantener la integridad de los datos.
- Es necesario determinar la frecuencia de sincronización de acuerdo con la actividad de actualización y la tolerancia de la aplicación a la actualidad de las réplicas. Sin embargo, este esquema simple en el que todas las actualizaciones se encaminan a un único servidor, no es razonable en muchos casos, por lo que es necesaria una política de sincronización más compleja.

Replicación de objetos ...



En resumen, la replicación de objetos proporciona mejoras en el rendimiento, pero introduce un número importante de complicaciones técnicas que no son triviales de resolver.

- Los productos ODBMS están comenzando a ofrecer ciertas facilidades que pueden ayudar a las aplicaciones a gestionar la replicación de objetos.
- Sin embargo, siguen siendo los desarrolladores los que deben determinan las políticas y las técnicas a emplear considerando los patrones de acceso a los objetos y la necesidad de información actualizada.