Bienvenidos a la segunda presentación dedicada a bases de datos distribuidas donde vamos a describir los modelos más relevantes de arquitectura para la construcción de sistemas distribuidos.

Bases de datos distribuidas

- Sistemas cliente/servidor
- Sistemas de 3 capas
- Sistemas *peer-to-peer*

EIMT.UOC.EDU

Principalmente existen tres modelos de arquitectura de sistemas distribuidos.

Estos modelos son los sistemas cliente/servidor (*client/server* en inglés), los sistemas de 3 capas (que constituyen un caso particular de los sistemas de N-capas –*multi-tier systems*, en inglés) y los sistemas entre iguales, más conocidos como sistemas *peer-to-peer* (P2P) que es su denominación en inglés.

Dentro de cada modelo existen variantes. El modelo de arquitectura elegido para el sistema distribuido también tiene un impacto en el desarrollo de las aplicaciones.

El objetivo fundamental es describir las características más importantes asociadas a estos sistemas, tomando como hilo conductor para la discusión la gestión de los datos.

Bases de datos distribuidas

- Sistemas cliente/servidor
- Sistemas de 3 capas
- Sistemas *peer-to-peer*

EIMT.UOC.EDU

Comenzamos hablando de los sistemas cliente/servidor. Estos sistemas fueron desarrollados a principios de los años 90.

Sistemas cliente/servidor

- Las características fundamentales de un sistema cliente/servidor son:
  - Los participantes en el sistema se clasifican en dos tipos (clientes y servidores), con funcionalidades diferentes.
  - Los clientes envían peticiones de servicio a ser resueltas por servidores.

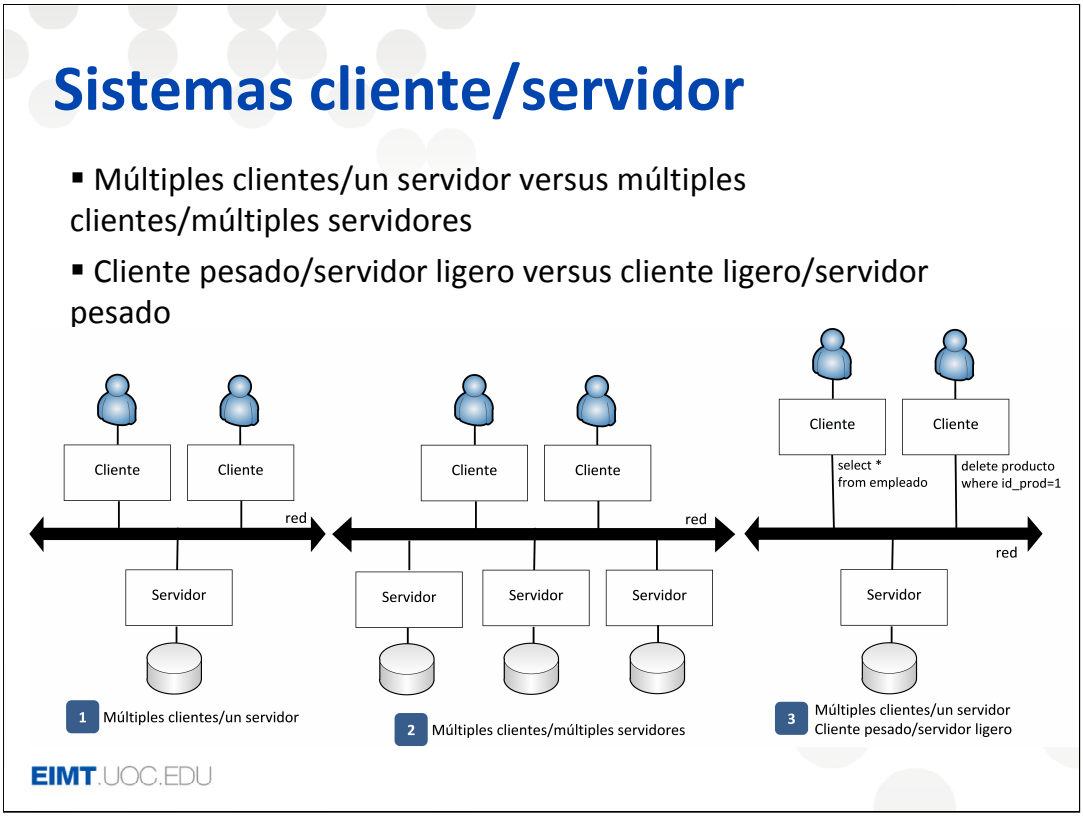
EIMT.UOC.EDU

Los sistemas cliente/servidor parten de la identificación de las funcionalidades a ser provistas por el sistema distribuido, y de su clasificación en dos grupos, las denominadas funciones de cliente y las funciones de servidor. Debido a esta separación en dos grupos, también se conocen en ocasiones como sistemas de dos capas.

La distinción entre cliente y servidor es de tipo lógico, por ejemplo, las funcionalidades de servidor no tienen porque ejecutarse en una única máquina, ni por un solo programa. Asimismo, nada impide que una misma máquina ejecute funcionalidades de tipo cliente y de tipo servidor. Con todo, es práctica habitual dedicar unas máquinas como servidor y otras que actúen como clientes en beneficio de las aplicaciones (y en consecuencia, de los usuarios).

Las funcionalidades provistas por los clientes y servidores también se conocen, respectivamente, como procesos cliente y procesos servidor. Por lo tanto, los nodos del sistema distribuido con atribuciones de cliente ejecutan procesos cliente, mientras que los nodos con atribuciones de servidor ejecutan procesos servidor.

Un ejemplo de funcionalidad a suministrar es la gestión de datos que es responsabilidad del servidor (o servidores) de la base de datos (es decir, del sistema o sistemas gestores de la base de datos). Por su parte, los clientes están al cargo de ejecutar el código asociado a las aplicaciones. Esto último puede incluir tanto la ejecución de la lógica de la aplicación (conocida también como lógica de negocio), como de la interacción con los usuarios.



Desde una perspectiva de bases de datos, existen diferentes tipos de sistemas cliente/servidor, dependiendo de los criterios de clasificación que se apliquen. A continuación analizamos dos posibles clasificaciones.

La primera tiene que ver con el número de puntos de almacenamiento en el sistema distribuido. Un sistema de múltiples clientes/un servidor (tal y como muestra la figura 1) se ajusta a una base de datos centralizada. Por su parte, un sistema de múltiples clientes/múltiples servidores se correspondería con una base de datos distribuida (véase la figura 2).

Por su parte, la segunda clasificación tiene que ver con el reparto de tareas, en relación a los datos, entre el cliente y el servidor. Si consideramos una base de datos relacional, en un sistema cliente pesado/servidor ligero, el nivel de comunicación entre las aplicaciones y el servidor de la base de datos son sentencias SQL individuales (p.e. *select*, *update*, *delete*), mientras que en un sistema cliente ligero/servidor pesado, parte de la ejecución de sentencias SQL que requiere la aplicación se encapsula en el servidor de la base de datos (en general, esto se consigue mediante procedimientos almacenados). En definitiva, en un sistema cliente ligero/servidor pesado, parte de la lógica de la aplicación se transfiere al servidor de la base de datos.

Las clasificaciones no son excluyentes entre sí, tal y como muestra la figura 3, que se corresponde a un sistema de múltiples clientes/un servidor de cliente pesado/servidor ligero.

Bases de datos distribuidas

- Sistemas cliente/servidor
- Sistemas de 3 capas
- Sistemas *peer-to-peer*

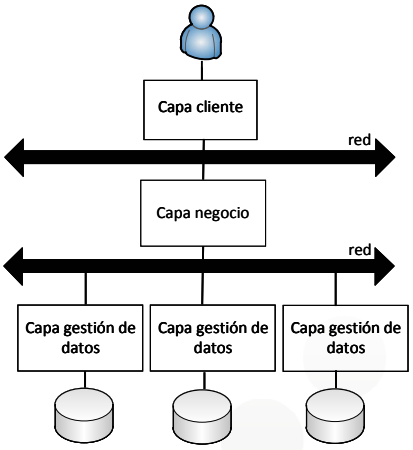
EIMT.UOC.EDU

Una vez examinadas las características asociadas a los sistemas cliente/servidor, pasamos a examinar los sistemas de 3 capas. Estos sistemas son un caso particular de los denominados sistemas de N-capas que, a su vez, constituyen una evolución de los sistemas cliente/servidor.

Sistemas de 3 capas

- Son un caso particular de los sistemas de N capas e incluyen:
  - Capa de cliente
  - Capa de negocio
  - Capa de gestión de datos

La capa de negocio actúa simultáneamente de servidor y cliente.



EIMT.UOC.EDU

Los sistemas cliente pesado/servidor ligero imponen bastantes desafíos ante cambios en la lógica de la aplicación. Además, tienen asociado un alto coste de transmisión de datos a través de la red, ya que el cliente es el encargado de procesarlos. Por su parte, los sistemas cliente ligero/servidor pesado pueden ser un cuello de botella en el rendimiento y pueden comprometer la disponibilidad ante situaciones de fallo.

Los sistemas de N-capas intentan solucionar los problemas que acabamos de mencionar. El más popular es el sistema de 3 capas. En este tipo de sistemas, se añade una capa intermedia entre el cliente y el servidor, de tal manera que estos quedan liberados de ejecutar el código que corresponde a la lógica de negocio. Más concretamente:

• La capa de cliente se encarga de la interacción con los usuarios.

• La capa de gestión de datos se encarga del acceso a los datos que pueden estar en uno o en varios servidores.

• La capa de negocio, por una parte, actúa como servidor de la capa de cliente, suministrando la información requerida por los usuarios, y por otra, actúa como cliente de la capa de gestión de datos, solicitando los datos necesarios para responder a las peticiones de los usuarios. La información dada a los usuarios surge como consecuencia de procesar los datos obtenidos de los servidores (es decir, surge como consecuencia de ejecutar el código asociado a la lógica de negocio).

Los sistemas cliente/servidor y los sistemas de 3 capas son ampliamente utilizados en el desarrollo de aplicaciones que acceden a bases de datos relacionales. También se usan en el desarrollo de aplicaciones Web.

Bases de datos distribuidas

- Sistemas cliente/servidor
- Sistemas de 3 capas
- Sistemas *peer-to-peer*

EIMT.UOC.EDU

El último modelo de arquitectura para la construcción de sistemas distribuidos que deseamos analizar son los sistemas P2P. Las primeras implementaciones son de finales de la década de los 90.

Sistemas *peer-to-peer*

- Las principales características de un sistema *peer-to-peer* (P2P) son las siguientes:
  - No hay diferenciación entre las funcionalidades asignadas a cada nodo.
  - Se orientan a la distribución masiva de datos.
  - Los participantes son heterogéneos y autónomos.
  - Existe un cierto grado de volatilidad.

EIMT.UOC.EDU

En un sistema P2P, todos los participantes (o nodos) se comportan como iguales entre sí, es decir, actúan simultáneamente como clientes y servidores con respecto al resto de integrantes del sistema distribuido. Esto no es contradictorio con el hecho de que ciertos participantes puedan tener atribuciones especiales.

En general, los sistemas P2P se orientan a la distribución masiva de datos, hecho que puede causar la existencia de cientos e incluso miles de nodos que se pueden encontrar muy dispersos (desde un punto de vista geográfico) y que trabajan de forma colaborativa.

Además, los participantes pueden ser de naturaleza diversa, y autónomos. La autonomía implica, por ejemplo, que un participante pueda poner limitaciones acerca de los datos que almacenará, o decidir con qué participantes se comunicará.

Finalmente, y también como consecuencia de la autonomía, se trata de sistemas sujetos a una cierta volatilidad. Un sistema P2P tiene que facilitar la adición y supresión de participantes, sin que ello comprometa la calidad del servicio desde el punto de vista de los usuarios.

Desde una perspectiva de gestión de datos, esto constituye una diferencia fundamental en relación, por ejemplo, a un sistema cliente/servidor, dado que este último es un entorno bien controlado, donde la adición o supresión de participantes es inusual, y si acontece, se planifica cuidadosamente.

Uno de los primeros sistemas P2P fue Napster (2000) que permitía el intercambio de ficheros (por ejemplo, ficheros de música) entre usuarios. Dejando al margen cuestiones legales, desde el punto de vista de la gestión de datos, el conjunto de operaciones a ofrecer era simple. Básicamente, incluía la posibilidad de añadir, buscar, leer y eliminar ficheros (es decir, subir, buscar, bajar y borrar canciones). A Naspter le siguieron otros sistemas, con diferentes o similares propósitos, como Gnutella, KazaA o el mismo Skype.

Sistemas *peer-to-peer*

- Un sistema P2P se construye sobre una red lógica (o red P2P) que se superpone sobre una física (por ejemplo, Internet).
- Existen diferentes tipos de redes P2P, por ejemplo las redes P2P no estructuradas versus las redes P2P estructuradas.

EIMT.UOC.EDU

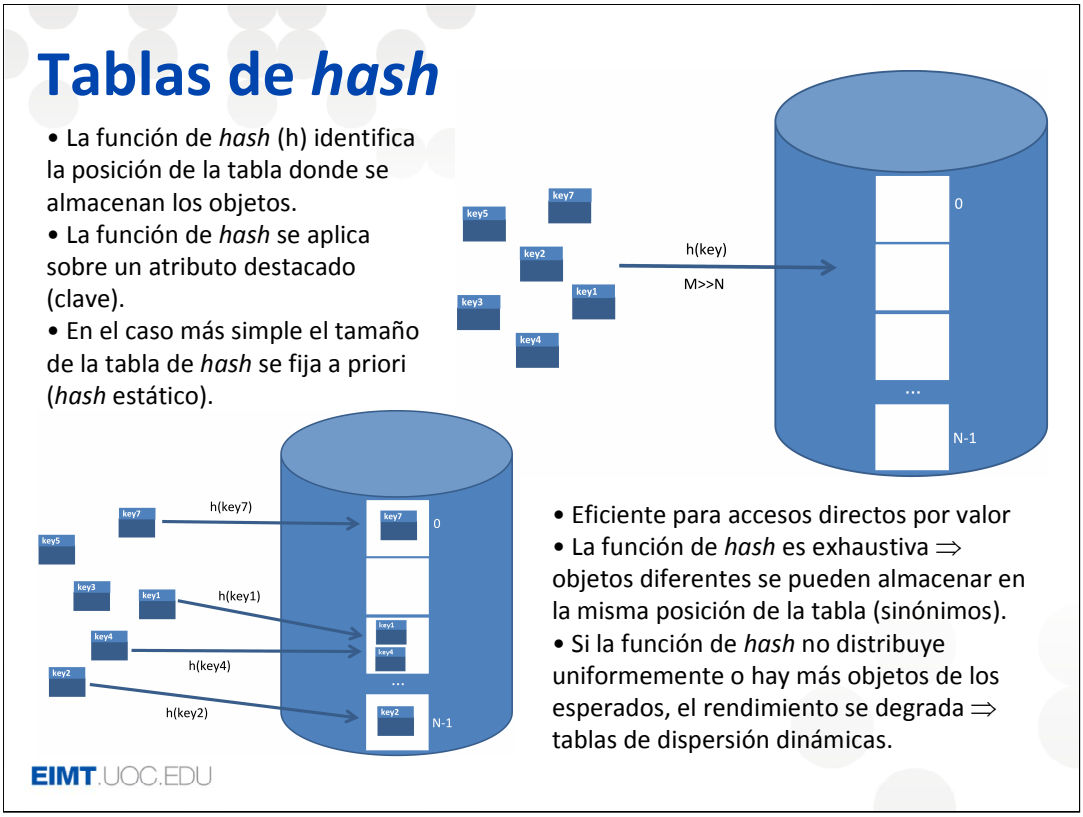
Un sistema P2P se construye sobre una red lógica que se superpone, a su vez, sobre una red física (usualmente Internet). La red lógica (o superpuesta) recibe el nombre de red P2P. Existen diferentes tipos de redes P2P que tienen un impacto en la implementación del sistema P2P. Establecer una clasificación que englobe todos los tipos es complicado, ante la diversidad y las dependencias que existen entre ellos.

A efectos de gestión de datos, nos interesa destacar dos tipos de redes P2P, las denominadas redes P2P no estructuradas y las estructuradas.

En una red P2P no estructurada no existen restricciones acerca de en qué nodos del sistema se tienen que almacenar los datos. En otras palabras, no existe una correlación entre un nodo y los datos que éste almacena. Esto puede dificultar la búsqueda eficiente de los mismos y puede causar una distribución no equitativa de la carga de trabajo entre los participantes.

Sin embargo, en una red P2P estructurada, existe un procedimiento que determina qué datos almacena cada participante. Esto facilita la indexación y recuperación eficiente de los mismos. Asimismo, ayuda a añadir y suprimir participantes en el sistema, causando un impacto mínimo en relación a la gestión de datos. Finalmente, también facilita distribuir la carga de trabajo de forma homogénea entre los participantes. En general, para decidir en qué participante (o participantes) se almacenan los datos, se usan técnicas de *hashing* distribuido. Para conseguir los beneficios descritos, los participantes sacrifican una parte de su autonomía.

Varias bases de datos NoSQL (en especial las de modelos de agregación) utilizan técnicas similares a las usadas en sistemas P2P basados en redes estructuradas, tal y como veremos a continuación. Antes recordaremos las nociones fundamentales de las tablas de *hash*.



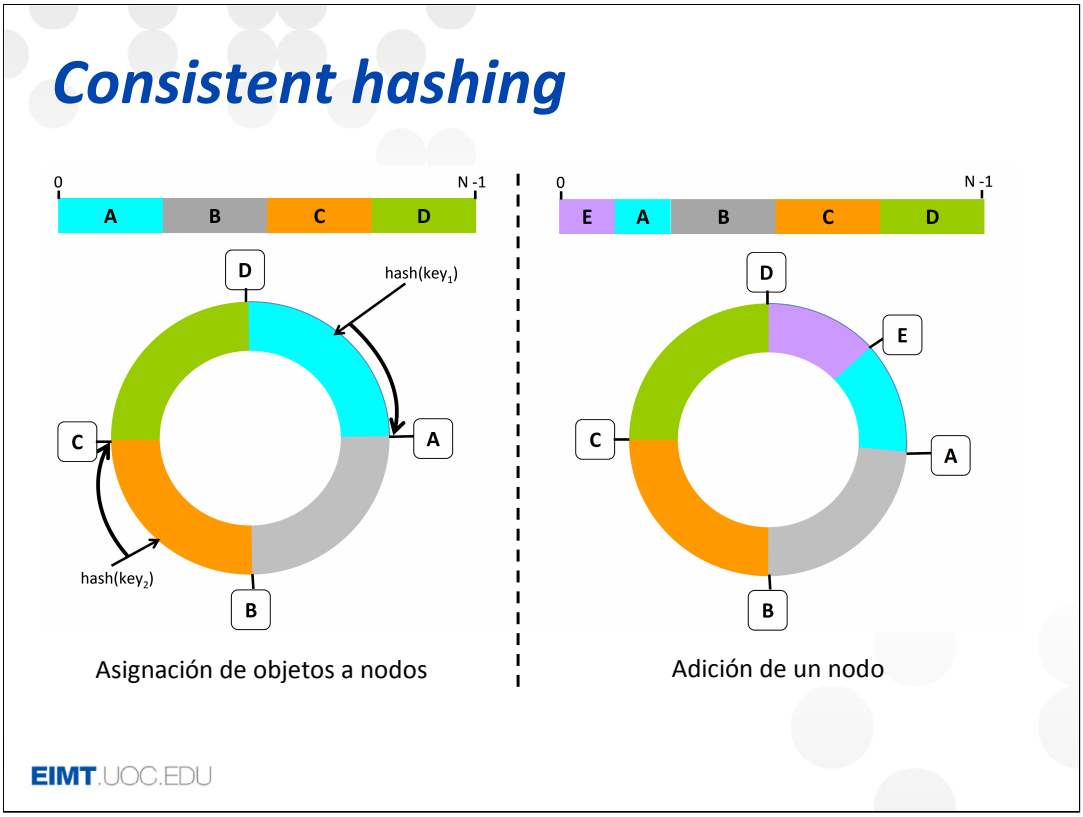
Las tablas de *hash* (en castellano tablas de dispersión) son estructuras de datos que se utilizan en diferentes ámbitos de la informática. Se basan en la existencia de una función (denominada *hash*) que determina en qué posición de la tabla se ubicarán los objetos que se desea almacenar. La función de *hash* se aplica sobre el valor que toma un atributo destacado del objeto que se denomina clave, y devuelve un número que constituye la posición de la tabla en donde se guardará el objeto. En la variante más simple, el número de posiciones de la tabla (N) está fijado a priori, y la función de *hash* tiene en cuenta dicho valor. En estos casos, hablamos de tablas de dispersión estáticas. El número de valores diferentes que puede tomar la clave (M) es mucho mayor que el número de posiciones de la tabla de *hash*. En consecuencia, las funciones de *hash* son exhaustivas. Aquellos objetos a los que la función de dispersión les asigna la misma posición en la tabla de *hash* se denominan sinónimos. Para minimizar la existencia de sinónimos, es vital que la función de dispersión distribuya uniformemente los objetos en la tabla.

Las tablas de dispersión permiten resolver eficientemente los accesos directos por valor (es decir, recuperar un objeto que tiene un valor determinado para la clave), pero no los accesos secuenciales por valor (por ejemplo, recuperar todos los objetos dentro de un rango de claves).

En el caso de una base de datos, nos podemos imaginar la tabla como un conjunto de páginas en disco, tal y como se muestra en las figuras. En la figura inferior, podemos ver que los objetos con valores de clave *key1* y *key4* son sinónimos.

Uno de los problemas que puede acontecer es que el rendimiento se degrade. Esto puede pasar cuando la función de *hash* no distribuye los objetos de forma uniforme, o cuando tenemos que almacenar muchos más objetos de los esperados (en definitiva, hemos asignado un número N de páginas insuficiente). En estos casos hay que gestionar objetos que no caben en la página que la función de *hash* ha determinado y que se denominan excedentes. Existen diversas alternativas para gestionarlos, la más simple es habilitar páginas de excedentes que los absorban y que se encadenan con las páginas inicialmente previstas. En el caso peor puede ser necesario reconstruir la tabla de forma completa, bien asignando un número mayor de páginas, bien cambiando la función de *hash*.

La situación descrita, claramente, no es deseable. Por ello, existen alternativas a las tablas de dispersión estáticas, las denominadas tablas de dispersión dinámicas. En ellas no es necesario que el número N de páginas esté fijado a priori. En otras palabras, el número de páginas crece a medida que se requiere más capacidad de almacenamiento. Cuando se añaden páginas, el número de objetos que es necesario recolocar es pequeño (en relación a una tabla de *hash* estática) y se puede determinar de forma exacta.



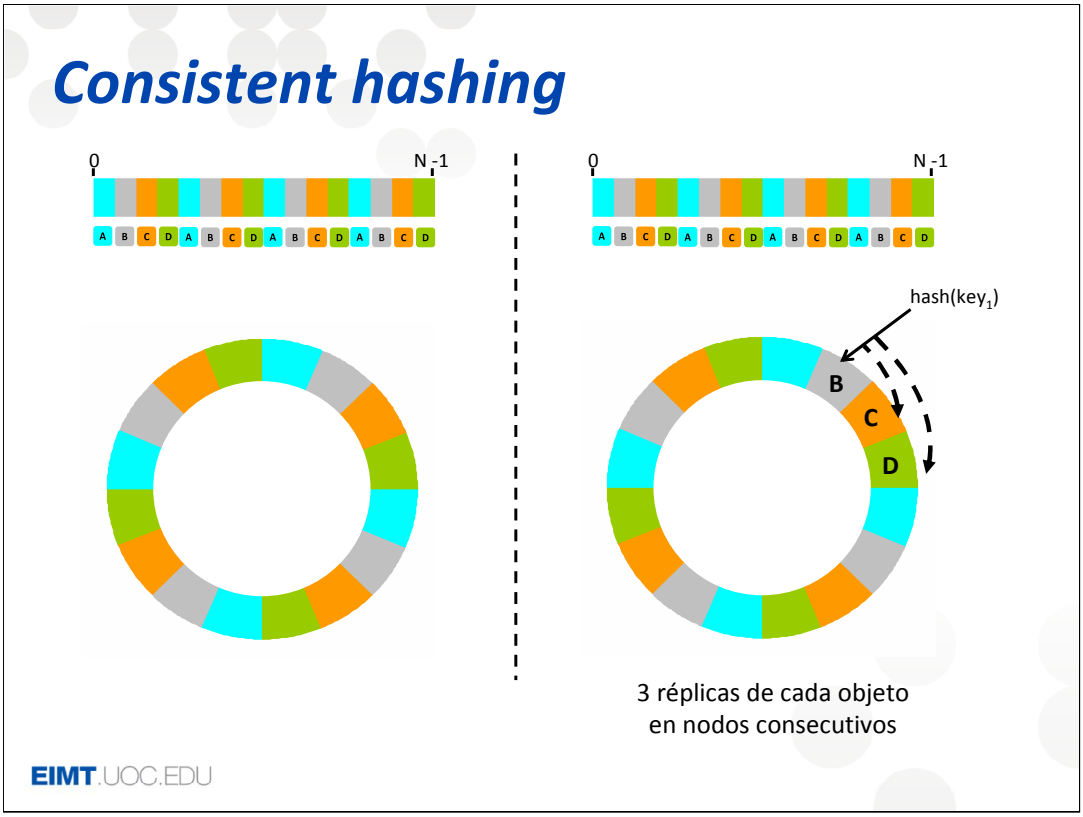
Como ya hemos comentado, algunas bases de datos NoSQL usan técnicas de *hashing* parecidas a las de las redes P2P estructuradas para determinar qué nodo se responsabiliza del almacenamiento de cada uno de los objetos a guardar. La base de datos distribuida que resulta de su aplicación se conoce bajo la denominación de tabla de *hash* distribuida. Una de las técnicas es la denominada *consistent hashing*. Vamos a explicar en qué consiste dicha técnica.

Tenemos una serie de nodos de partida. Tal y como muestra la figura de la izquierda, en nuestro ejemplo tenemos 4 (identificados como A, B, C y D). La función de *hash* se aplica sobre la clave del objeto, y dependiendo del valor devuelto por la función, el objeto se almacena en un nodo u otro. Para ello, los posibles valores que retorna la función de *hash* (de 0 a N-1) se agrupan por rangos que se asignan a nodos. Si nos fijamos de nuevo en la figura, los valores que pertenecen al rango pintado en azul serán responsabilidad del nodo A, los grises del B, los naranjas del C y los verdes del D. Para entender mejor el funcionamiento, conviene representar los rangos y nodos en un círculo (o anillo).

Cuando se desea almacenar un nuevo objeto, se aplica la función de *hash* sobre su clave. El valor devuelto, se coloca en el círculo. Una vez hecho esto, el nodo que almacenará el objeto será el primero que encontremos siguiendo el sentido de las agujas de un reloj. Sobre nuestro ejemplo, el objeto con valor de clave *key1* se almacena en el nodo A y uno con valor de clave *key2* se guarda en el nodo C. Fijémonos que la técnica asigna objetos a nodos, pero no dice cómo cada nodo debe guardar localmente los objetos. Una posibilidad sería que los nodos usasen tablas de dispersión.

Para añadir un nuevo nodo (es decir, para añadir más capacidad de almacenamiento), el nodo se incorpora al circulo. Esto implica una redistribución de los rangos que se asignan a cada nodo. Por ejemplo, tal y como muestra la figura de la derecha, la adición de un nodo E anterior al nodo A, implica repartir el rango de valores que retorna la función de *hash* entre E y A. En concreto, el nodo E tiene asignado el rango de valores coloreado en violeta. Esto significa que una parte de los objetos almacenados en el nodo A deben ser recolocados en el nodo E (esos objetos serán aquéllos a los que la función de *hash* les asignó un valor en la franja azul que ahora ha pasado a ser violeta). Fijémonos que los objetos almacenados en el resto de nodos, están en el lugar que les corresponde, y no deben ser recolocados.

Uno de problemas de la estrategia que acabamos de describir es que toda la sobrecarga de trabajo que comporta el proceso de recolocación de objetos es gestionada por dos nodos (en nuestro ejemplo, los nodos E y A).



Este problema de sobrecarga puede ser aliviado mediante la asignación de un conjunto de rangos a cada nodo, en lugar de un único rango. Esta es la situación que se muestra en la figura de la izquierda. Se trata de una solución de compromiso que consiste en repartir la carga de trabajo del proceso de recolocación de objetos (potencialmente) entre todos los nodos del sistema.

Para aumentar la disponibilidad puede ser necesario disponer de réplicas de un mismo objeto. En el caso del *consistent hashing*, las réplicas de un mismo objeto se almacenan en los nodos consecutivos al nodo que la función de *hash* determina que tiene que almacenar el objeto. Ésta es la situación que se muestra en la figura de la derecha. Tal y como se observa, la función de dispersión ha determinado que el nodo que tiene que almacenar el objeto con clave *key1* es el nodo B (en la figura se muestra con flecha de línea sólida). Adicionalmente, se guardan réplicas de ese mismo objeto en los nodos C y D (son las flechas de línea discontinua de la figura). En definitiva, tenemos un total de 3 copias (o réplicas) de un mismo objeto (el objeto con clave *key1*). La primera réplica es la que se almacena en el nodo B, la segunda la que se guarda en el C y la tercera réplica la que se almacena en el nodo D. El número de réplicas que se desea tener de cada objeto constituye un parámetro de configuración de la base de datos, sobre nuestro ejemplo este número se ha fijado a 3.

Una de las bases de datos NoSQL que utiliza *consistent hashing* es Riak (una de las bases de datos clave-valor más prominentes). Esta técnica facilita un esquema de crecimiento horizontal, es decir, un crecimiento acorde a las necesidades en base a añadir nuevos puntos de almacenamiento.

Bases de datos distribuidas

- Sistemas cliente/servidor
- Sistemas de 3 capas
- Sistemas *peer-to-peer*

EIMT.UOC.EDU

En esta presentación hemos descrito los modelos de arquitectura más habituales para la construcción de sistemas distribuidos.

A modo de conclusión, y desde una perspectiva de gestión de datos, la principal diferencia entre estos modelos (cliente/servidor, 3 capas y P2P) tiene que ver con las técnicas que se utilizan para proveer el máximo de funcionalidades asociadas a un sistema gestor de base de datos.

Entre estas funcionalidades, destaca la de ofrecer una visión integrada de la base de datos a los usuarios del sistema, entre los que se incluyen los desarrolladores de aplicaciones.

Referencias

I. Katsov (2012). "Distributed Algorithms in NoSQL Databases", *Highly Scalable Blog. Articles on Big Data, NoSQL, and Highly Scalable Software Engineering*. (http://highlyscalable.wordpress.com/2012/09/18/distributed-algorithms-in-nosql-databases/)

L. Liu & M.T. Özsu (Eds.) (2009). *Encyclopedia of Database Systems*. Springer.

E.K. Lua et al. (2005). "A Survey and Comparison of Peer-to-Peer Overlay Network Schemes", *IEEE Communications Surveys and Tutorials* vol. 1, pp 72-93.

M.T. Özsu & P. Valduriez (2011). *Principles of Distributed Systems*. 3rd edition. Springer.

R. Rodrigues & P. Druschel (2010). "Peer-to-Peer Systems", *Communications of the ACM* 53(10), pp 72-82. (http://cacm.acm.org/magazines/2010/10/99498-peer-to-peer-systems/fulltext)

O. Romero & M. Oliva (2012). Distributed Databases. Material docente UOC, asignatura Arquitectura de bases de datos.

EIMT.UOC.EDU

Esperamos que hayáis disfrutado y aprendido con este vídeo. A continuación encontraréis algunas referencias que os permitirán profundizar más en los conceptos que hemos tratado.

Que tengáis un buen día.