Bases de datos no convencionales © EDICIONES ROBLE, S.L.

Indice

Bases de datos no convencionales

I. Introducción

En la mayoría de los sistemas informáticos es necesario almacenar de forma persistente la información que se procesa. En las últimas décadas, los sistemas de persistencia que han dominado el mercado han sido las bases de datos relacionales. Esto se ha debido, entre otros motivos, a la eficiencia para procesar la información mediante el uso de transacciones ACID, a la solidez del modelo formal en que se sustentan (el modelo relacional) o a la existencia de un lenguaje estándar de manipulación de los datos (el lenguaje SQL). Además, conceptualmente, se trata de un modelo simple e intuitivo, ya que está basado en tablas en donde se almacena la información.

Sin embargo, en los últimos años esta situación ha cambiado, debido a la aparición de nuevas necesidades de persistencia de la información que ha generado el contexto del Big Data. En primer lugar, el tipo de datos que se manipulan se caracterizan porque son semiestructurados o no tienen estructura alguna, pueden tener datos erróneos, datos vacíos, etc. Esta condición choca con el tipo de dato gestionado por una base de datos relacional, donde se procesa, de una forma eficiente, información que sigue una estructura prefijada. Asimismo, las necesidades de procesamiento en este ámbito han crecido exponencialmente, de manera que para poderlas cubrir hace falta que se escalen horizontalmente, dando lugar así a soluciones distribuidas. Este punto también es conflictivo con las bases de datos relacionales, las cuales están diseñadas para que se ejecuten en sistemas centralizados, ya que, en la mayoría de los casos, su ejecución en un entorno distribuido no es posible.

Las razones comentadas, y otras, crearon la necesidad de buscar sistemas de persistencia más acordes al nuevo contexto. Así, surgieron diversas alternativas a las bases de datos relacionales, las cuales reciben el nombre genérico de bases de datos NoSQL. Su característica común es que todas ellas se basan en modelos conceptuales diferentes al modelo relacional, que se ajustan a la mayoría de las necesidades de persistencia de datos surgidas en el ámbito del Big Data.

En esta unidad, se va a realizar una introducción a las denominadas bases de datos NoSQL, analizando las razones que motivaron su aparición y revisando sus características principales. En este sentido, la unidad se estructura de la siguiente manera:

En primer lugar, se discutirán las limitaciones de las bases de datos. A continuación, se introducirán las bases de datos NoSQL y las familias que existen. Tras esta exposición, se realizará una presentación de sus principales características. En las siguientes secciones se discutirán los modelos de distribución y replicación de datos. Por último, se analizará el denominado teorema CAP y las bases de datos orientadas a agregados.

II. Objetivos

Los objetivos que se alcanzarán tras el estudio de esta unidad son:

- Comprender las limitaciones del modelo relacional en cuanto a las nuevas necesidades de procesamiento de datos.
- Conocer las principales características de las bases de datos NoSQL.
- Conocer los modelos de distribución y replicación de datos en el ámbito NoSQL.
- Entender las implicaciones del teorema CAP.
- Entender y conocer las características de las bases de datos orientadas hacia agregados.

III. Los datos y el Big Data

El Big Data se puede definir como un conjunto de tecnologías que permiten la recopilación, almacenamiento, gestión, análisis y visualización, potencialmente en condiciones de tiempo real, de grandes conjuntos de datos con características heterogéneas.

El concepto de Big Data se puede explicar mediante 5 características principales, conocidas como las “5 uves” (velocidad, volumen, variedad, veracidad y valor; véase la figura 1), que definen el tipo y naturaleza de los datos que son procesados y los fines que se buscan con el procesamiento de los mismos.

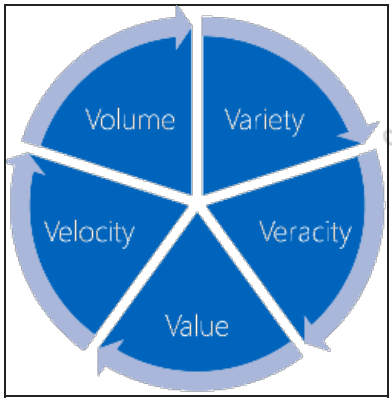

Figura 1. Las 5 uves del Big Data. Fuente: GBI Data Science

Volumen

Se refiere al gran volumen de información que se maneja (se consideran magnitudes del orden de petabytes o exabytes de información). En este sentido, los datos que se necesitan gestionar aumentan con un crecimiento exponencial, lo que obliga a realizar ampliaciones continuas de los almacenamientos de información.

Velocidad

Se refiere a la enorme velocidad a la que se generan, recogen y se procesan los datos de información. En este sentido, es necesario poder almacenar y procesar en tiempo real millones de datos que se generan en instantes de segundos. Muchos de estos datos proceden de fuentes de información, tales como sensores, redes sociales, sistemas de control ambientales y otros dispositivos de captación de información. Obsérvese que es la velocidad de procesamiento la que permite obtener un beneficio de la explotación, antes de que el dato quede obsoleto.

Variedad

Esta característica se refiere a la necesidad de poder gestionar información que procede de fuentes de información muy heterogéneas en la que los datos tienen diferentes niveles de estructuración, diferentes tipos de datos, etc. Así, para poder agregar todos los datos y gestionarlos como una unidad, se necesitan almacenamientos de información que sean lo suficientemente flexibles como para albergar datos heterogéneos. En este sentido, las bases de datos relacionales no constituyen una solución adecuada, dado que el tipo de dato esperado es homogéneo, regular y con una estructura previamente definida en el esquema de la base de datos.

Valor

Este aspecto se refiera a la capacidad de poder explotar la información almacenada con el objetivo de aprovecharla con diferentes fines, tales como obtener una rentabilidad económica, optimizar la producción, mejorar la imagen corporativa, acercarse mejor a las necesidades y preferencias de los clientes o predecir cómo de bien puede desarrollarse una campaña publicitaria o de ventas de un determinado producto. Esta característica es básica para que a una empresa o entidad le pueda interesar invertir en procesar tales cantidades de información.

Veracidad

Esta última característica se refiere a la necesidad de procesar adecuadamente los grandes volúmenes de información, de manera que la información que se obtenga sea verídica, y permita tomar decisiones adecuadas a partir de los resultados de su procesamiento. Este aspecto junto al anterior constituye las razones de peso que dan sentido al procesamiento masivo de la información.

El Big Data tiene aparejado un conjunto de requerimientos o características que lo identifican:

- Necesidad de una arquitectura de datos descentralizada y flexible.
- Necesidad de utilizar modelos de computación distribuida que permitan gestionar y procesar el tipo de datos propios de estos ambientes (datos semiestructurados o sin estructura). Por este motivo, se requerirá llevar a cabo tareas de procesamiento muy intensivas de una manera masiva.
- Necesidad de analizar grandes cantidades de datos en tiempo real en entornos de ejecución distribuidos.

Soluciones tecnológicas

Por otra parte, se puede afirmar que el Big Data no solo se explica con las características antes comentadas, también es cierto que se trata de un conjunto de nuevas tecnologías que han aparecido como respuesta a las necesidades antes citadas. En este sentido, existe una fragmentación de herramientas y soluciones tecnológicas que dan solución a algunos de los problemas concretos y específicos que se han comentado. De una manera esquemática, se puede decir que han surgido soluciones para algunas de las siguientes áreas:

Nuevos sistemas de persistencia de la información para las actuales necesidades de almacenamiento y procesamiento de datos, como las denominadas bases de datos NoSQL.

Lenguajes de programación específicos para realizar análisis de datos y procesar la información. Se requieren estructuras de datos que faciliten la manipulación de los datos y las operaciones sobre los mismos. Algunos ejemplos de estos lenguajes son el lenguaje R, un lenguaje de programación específico para el análisis de datos, o las librerías científicas de lenguaje de programación Python.

Sistemas de procesamiento masivo para poder trabajar con enormes cantidades de datos en tiempo real. Estos sistemas requieren poder funcionar en paralelo para obtener la suficiente potencia de cálculo y utilizan nuevos paradigmas de computación diferentes a aquellos tales como el algoritmo map-reduce. Algunos ejemplos de estos sistemas de procesamiento son Spark o Hadoop.

Herramientas de visualización de datos que tienen como objetivo ayudar a interpretar los resultados de los datos procesados mediante representaciones visuales que muestran de manera intuitiva y simple los datos. De esta manera, los analistas reciben ayuda en la tarea de tomar decisiones y pueden aprovechar así la información obtenida de su procesamiento. Algunos ejemplos de estas herramientas son Tableau o Qlik.

Algoritmos de procesamiento de la información. En este sentido, se han desarrollado algoritmos y procedimientos de análisis de la información originados en el área de la estadística, así como en el área de la inteligencia artificial, los cuales funcionan de una manera muy eficiente cuando se utilizan sobre una gran cantidad de datos. Por ejemplo, un área en auge es la denominada Machine Learning o los algoritmos de aprendizaje profundo que se están empezando a utilizar y aplicar en diferentes ámbitos.

Objetivos

En cualquier caso, para poder aprovechar los beneficios que puede aportar el Big Data en las tecnologías emergentes, resulta esencial conocer las características de las fuentes de información que van a dar los datos, conocer la naturaleza de los datos que se van a obtener, conocer el tipo de preguntas a las que se quieren responder y conocer las herramientas necesarias más adecuadas para poder responder a estas preguntas. Es por ello que la elección de la herramienta adecuada constituye un paso crítico en la aplicación del Big Data a un problema y dominio concreto.

Otra característica distintiva del Big Data se refiere a los tipos de análisis y procesamientos de la información que se desean realizar. En general, se trata de modelos de procesamiento predictivos que tienen como objetivo responder a preguntas sobre el comportamiento futuro de un proceso, de un individuo o bien de un grupo humano, a partir de los comportamientos conocidos del pasado y de otra serie de datos complementarios disponibles relacionados. Así, en este tipo de modelos se busca alcanzar objetivos como los siguientes:

Predecir determinados comportamientos a partir de los datos que ha generado un individuo

Para ello, se evalúa la probabilidad que tiene un individuo de mostrar un comportamiento específico en el futuro, tomando como base los comportamientos anteriores, así como otros datos adyacentes.

Buscar regularidades en la información

En este sentido, se procesa la información con el objetivo de encontrar patrones repetitivos que permitan discriminar la información. Estos patrones pueden servir para responder a preguntas que se planteen sobre el comportamiento de un individuo, grupo humano o institución.

Determinar un riesgo u oportunidad

Para ello, se realizan cálculos en tiempo real sobre los datos recopilados con el objetivo de poder evaluar un determinado riesgo u oportunidad, que permita orientar la toma de una decisión adecuada. Para ello son importantes algunos factores tales como la velocidad en la que se procesa, la cantidad de los datos que son procesados y la calidad de los datos en cuanto a su momento de generación y procesamiento.

Descubrir relaciones entre los datos con el objetivo de poder clasificar a los individuos en grupos

Este tipo de análisis suelen ser muy útiles en ámbitos empresariales para distinguir segmentos de negocio, así, por ejemplo, no es lo mismo el tipo de producto que se le puede ofrecer a una persona joven que a una persona anciana o adulta. De esta forma, se puede conocer información común de cada grupo y tomar decisiones específicas para cada segmento.

Identificar relaciones entre diferentes individuos

Esto permite predecir las consecuencias que puede tener una decisión sobre un conjunto de individuos, así como planificar diferentes tipos de acciones y efectos sobre los individuos relacionados, o bien poder deducir información de forma implícita a partir de las relaciones existentes. Un ejemplo típico es la búsqueda de relaciones e información a partir de las personas que se encuentran en contacto a través de una red social.

Describir la relación que puede existir entre todos los elementos que hay que tener en cuenta para poder tomar una decisión

Poder obtener la decisión óptima de entre todas las posibles, a partir de la información de la que se dispone, y conocer las variables y valores que determinan la propia decisión. Todo ello con la finalidad de predecir los resultados mediante el análisis de muchas variables. Este tipo de análisis tiene una aplicación especialmente importante en la toma de decisiones de una empresa donde existen muchas variables a tener en cuenta, donde existe un riesgo económico y se esperan obtener unos resultados determinados.

A modo de resumen, se puede decir que el fenómeno del Big Data ha supuesto un cambio tecnológico que ha traído consigo la aparición de un conjunto de nuevas tecnologías, algoritmos, lenguajes de programación y paradigmas de computación. Y todo este proceso ha sido consecuencia de las nuevas necesidades de procesamiento de la información y de las oportunidades de negocio que ofrece el poder disponer de enormes cantidades de datos para que puedan ser procesados y explotados.

IV. Las bases de datos relacionales

La mayoría de los sistemas informáticos requiere almacenar información de una forma persistente para poder realizar después procesamientos sobre la misma. En las últimas décadas, el mecanismo de persistencia más extendido han sido las bases de datos relacionales. Este tipo de base de datos se fundamenta en un modelo formal denominado modelo relacional, el cual utiliza como unidad de almacenamiento tablas. Una tabla es una estructura de información formada por columnas, que representan los campos de información de los que se desea almacenar información, y filas, que son la información propiamente dicha de cada columna. Un conjunto de columnas representa la información que se desea almacenar de una relación abstracta, como un estudiante, una empresa, un empleado, y cada fila representa una instancia concreta de la relación abstracta.

Por ejemplo, en una empresa podría interesar conocer el CIF, el nombre de la empresa, ámbito y número de empleados, lo que constituiría la relación abstracta. Sin embargo, una fila o instancia de la relación abstracta sería: “4343434F”, “Almacenes El gato”, “Papelería”, “10”.

Las bases de datos relacionales presentan un conjunto de características que las hacen muy eficientes en el procesamiento de la información almacenada:

Concurrencia

Este aspecto se refiere a las situaciones en las que un conjunto de datos almacenados necesita ser procesado por más de una aplicación a la vez. Las operaciones de procesamiento conllevan tareas secuenciales de acceso, modificación, eliminación o inserción de datos. El orden en el que se llevan a cabo estas tareas puede ser importante en algunos casos, pues puede cambiar el estado final de los datos, ya que dependen del orden de ejecución de las tareas, y, por tanto, la consistencia de la información. Estas situaciones hacen necesaria la existencia de algún mecanismo que permita coordinar de manera adecuada las aplicaciones que van a acceder a los datos. Se podría realizar de manera manual e implementarlo directamente en el código de las aplicaciones. Sin embargo, esta solución se complica por el número de problemas que surgen de una forma incremental a medida que aumenta el número de aplicaciones que intervienen. En este sentido, una característica fundamental de las bases de datos relacionales es la disponibilidad de un mecanismo ya implementado en las mismas para afrontar este problema. Se trata del uso del concepto de transacción como unidad de procesamiento. Así, las transacciones permiten gestionar al programador de una manera eficiente y transparente la concurrencia de acceso y el procesamiento que se puede dar en un sistema de bases de datos relacional.

Integración

Este aspecto se refiere a aquellas situaciones en las que varias aplicaciones informáticas comparten datos, por ejemplo, que una de las aplicaciones use los datos que son generados por otras aplicaciones. En estos casos, es necesario coordinar a las aplicaciones que colaboran, de manera que no se produzcan inconsistencias en los datos intercambiados y que se encuentren sincronizadas, de tal forma que la aplicación consumidora sepa cuándo puede recuperar datos, y la aplicación productora sepa cuándo puede añadir nuevos datos. El almacenamiento de los datos en un base de datos relacional es una solución muy interesante, pues esta puede actuar de mediadora entre las aplicaciones implicadas y además puede garantizar que las necesidades de sincronización y consistencia de la información se cumplirán, gracias nuevamente al concepto de las transacciones ACID y al sistema de control de concurrencia implementado en las bases de datos relacionales. Una de las consecuencias de usar el modelo ACID es garantizar el funcionamiento adecuado en estas situaciones.

Modelo estándar

Otro aspecto básico que ofrecen las bases de datos relacionales se refiere a los elementos estándar en los que se fundamenta. Por un lado, se encuentra el modelo relacional. Es un concepto formal en el que se basan estas bases de datos. Y, por otro lado, el lenguaje de bases de datos relacionales SQL. Cualquiera que quiera diseñar, crear o gestionar una base de datos relacional utilizará la misma terminología y conceptos, al ser estándar. Asimismo, en el caso de SQL, aunque existen diferentes implementaciones del lenguaje con algunas diferencias, las construcciones fundamentales y básicas del lenguaje son las mismas en todos los dialectos de SQL. Este factor de estandarización ha constituido una de las razones que han impulsado su utilización y expansión, pues permite trabajar a grupos de desarrollo desconocidos con un lenguaje y terminología comunes.

V. Limitaciones de Las bases de datos relacionales

Aunque en la sección anterior se han presentado las principales ventajas de las bases de datos relacionales, también hay que conocer sus limitaciones, entre las cuales destacan dos, por su impacto en el coste y la eficiencia, que son el problema de la impedancia y el problema de la distribución de datos.

Problema de impedancia

El primer problema hace referencia a la diferencia que presentan las estructuras de datos que son utilizadas en el ámbito de las bases de datos relacionales y las que se utilizan en los programas que explotan los datos de estos sistemas de almacenamiento. En el caso de las bases de datos relacionales, los datos almacenados se corresponden con datos simples, tales como números, cadenas o booleanos. No admiten ningún tipo de dato estructurado complejo, como podría ser una lista o registro. Asimismo, la unidad de almacenamiento y de intercambio de información son las filas de las tablas que sirven de almacenamiento. Sin embargo, en los lenguajes de programación se manipulan datos con una mayor riqueza y complejidad estructural, tales como pilas, colas, listas, etc. Esto supone un problema de comunicación de la información entre las bases de datos y los programas, ya que obliga a implementar un proceso de traducción entre ambos contextos. Así, cada vez que se recupera información de una base de datos relacional para usarla en un programa, es necesario descomponerla y transformarla en las estructuras de datos que se están usando en el mismo. Y, de igual manera, cuando un programa necesita almacenar información en una base de datos relacional, requiere otro proceso de transformación de la información almacenada en las estructuras de datos gestionadas por el programa en datos simples y agrupadas en secuencias que constituyen filas, que es lo que admite almacenar una base de datos relacional. Estas transformaciones constituyen un coste computacional adicional a los procesamientos de información que se realizan en los programas, que pueden llegar a tener un impacto importante, tanto en líneas de código como en tiempo de ejecución, lo que dependerá de la cantidad de datos y transformaciones que sean necesarias llevar a cabo. Este problema se ha denominado problema de la impedancia, que consiste en la diferente naturaleza que tienen los datos gestionados por una base de datos relacional y los que manejan los lenguajes de programación. Para aliviar este problema, se han creado algunas soluciones, como frameworks de las bases de datos orientadas a objetos o frameworks que mapean la información almacenada en la base de datos en un modelo orientado a objetos, como por ejemplo Hibernate. Sin embargo, estas soluciones no son del todo eficaces, pues, aunque resuelven en parte el problema de la impedancia, introducen otros problemas, como una reducción del rendimiento de la base de datos debido, entre otras razones, a la implementación de operaciones y consultas que obvian la existencia de una base de datos por debajo del modelo orientado a objetos.

Problema de la distribución de datos

El segundo problema hace referencia a la ejecución distribuida de las bases de datos relacionales. En la actualidad se necesita procesar ingentes cantidades de datos que crecen de manera exponencial y con una velocidad de generación muy rápida, dado que en muchos casos proceden de sensores o aparatos de control de constantes vitales, de fenómenos de la naturaleza, etc. Para ello, hacen falta máquinas con una capacidad de procesamiento importante. Para cumplir con estos requerimientos existen dos alternativas. Por una parte, se encuentran las soluciones de escalado vertical y, por otra parte, las soluciones de escalado horizontal.

Escalado vertical

El primer tipo de solución consiste en disponer de una máquina con altas prestaciones que cubra las necesidades de procesamiento requeridas. Desde el punto de vista de las bases de datos relacionales, es una solución ideal, dado que estas están diseñadas para ejecutarse en ambientes centralizados sobre una sola máquina. Sin embargo, desde el punto de vista económico y estratégico, se trata de una mala solución, ya que el crecimiento exponencial de los datos y su velocidad de generación hace que, en un tiempo relativamente corto, las máquinas se queden obsoletas y pequeñas con respecto a las necesidades de procesamiento requeridas, y es necesario comprar una nueva máquina. Asimismo, hay que tener en cuenta la debilidad estratégica de una solución centralizada, dado que un fallo en la máquina que contiene el sistema de almacenamiento tendrá como consecuencia una pérdida de la información y, por tanto, de todas las aplicaciones que exploten los datos almacenados, aunque también es cierto que normalmente existen copias de seguridad que se realizan periódicamente para que el impacto de un suceso de esta índole tenga unos efectos limitados.

Escalado horizontal

El segundo tipo de solución consiste en utilizar un conjunto de máquinas que trabajen colaborativamente en paralelo en el procesamiento de la información, denominado clúster de máquinas. De esta forma, el trabajo en conjunto de todas ellas permite alcanzar la capacidad de procesamiento requerido. Además, cuando se necesita incrementar la capacidad de procesamiento en un momento dado, basta aumentar el número de máquinas que forman el clúster. Asimismo, el tipo de máquinas utilizadas en los clústeres suelen ser de peor calidad en cuanto a prestaciones y más baratas que las máquinas que se utilizan en una solución de escalado vertical. Un requisito en este tipo de soluciones es que los datos que se están gestionando se encuentren distribuidos entre las máquinas del clúster. Este requisito, en determinadas condiciones, constituye una ventaja con respecto a una solución vertical, puesto que, si alguna máquina del clúster falla, el sistema no tiene porqué dejar de funcionar, dado que el resto de las máquinas seguirá funcionando y, si además se han replicado los datos en varias máquinas, bastará redireccionar el sistema a la máquina replicada. Sin embargo, esta característica entra en conflicto con el ámbito natural de ejecución de una base de datos relacional, que es centralizado en una única máquina. Así, en general, las bases de datos relacionales no están diseñadas ni preparadas para su ejecución en ambientes distribuidos. Se plantean problemas como saber la forma de descomponer las tablas de una base de datos relacional y decidir qué tablas se almacenan en una máquina y cuáles en otra, o cómo ejecutar las consultas sobre las tablas, en caso de estar distribuidas, pues habría que saber dónde se encuentra cada tabla. También se producen otros problemas acerca del tipo de consultas que son posibles realizar en un ambiente distribuido, la integridad referencial, la gestión de transacciones o el control de la concurrencia. Por otro lado, existe un factor económico que también hay que tener en cuenta, si se distribuye una base de datos relacional, ya que este tipo de ejecuciones requeriría la ejecución de varias instancias distintas de las bases de datos, lo cual incrementaría su coste económico. Para solventar estas limitaciones, han surgido soluciones dentro del ámbito de las bases de datos relacionales que han tratado de añadir algunos de los requisitos necesarios para su ejecución distribuida. Estas soluciones se denominan genéricamente bases de datos NewSQL. Mantienen las características principales de una base de datos relacional, así como el uso del lenguaje estándar SQL. Sin embargo, ninguna de ellas ha conseguido una implantación suficientemente sólida como para convertirse en una solución distribuida del modelo relacional. Así que las bases de datos relacionales se siguen utilizando para los ámbitos para las que fueron creadas.

VI. Bases de datos NoSQL

Ante las limitaciones presentes en las bases de datos relacionales para cubrir las necesidades surgidas en el ámbito del Big Data, algunas empresas comenzaron a desarrollar sistemas de persistencias alternativos que encajaban mejor con estos requerimientos. Entre estas empresas cabe destacar Amazon y Google, las cuales desarrollaron las que se consideran las primeras bases de datos NoSQL: las Big Tables, de Google, y Dynamo, de Amazon. Estas bases de datos tienen como característica común el hecho de que pueden gestionar y procesar enormes cantidades de datos mediante un sistema distribuido. A partir de ese momento, surgieron otras bases de datos que buscaban, de igual forma, dar solución a los problemas y limitaciones que las bases de datos relacionales no eran capaces de cubrir.

Características

Las bases de datos NoSQL comparten algunas características:

No utilizan el lenguaje SQL para realizar consultas, aunque todas disponen de lenguajes de consultas con un fin similar. Incluso hay algunas bases de datos que utilizan lenguajes con una sintaxis muy similar, como es el caso de la base de datos Cassandra con el lenguaje CQL.

La mayoría de las bases de datos NoSQL incluye entre sus características la posibilidad de ejecutarse en entornos distribuidos, dando así respuesta a las necesidades de procesamiento distribuido. Esta característica influye en los modelos de datos subyacentes, en la forma de gestionar la consistencia de los datos o la concurrencia. Es cierto que no todas las bases de datos NoSQL cumplen con esta característica. Este es el caso de las bases de datos orientadas a grafos, que no están diseñadas para su ejecución en un entorno distribuido.

No usan esquemas de estructuración de la información previamente prefijados. En el mundo relacional, lo primero que se debe hacer al diseñar una tabla es definir cómo se va a estructurar la información y qué tipo de datos se van a utilizar, de manera que la información que se almacene cumpla con esta definición (no admitiendo datos que no lo cumplan). Sin embargo, en el mundo de las bases de datos NoSQL no se utilizan esquemas, simplemente se añade libremente la información que se desea almacenar. Esta información puede tener diferentes estructuras, tan complejas como se quiera, ser heterogéneas y con diversos tipos de datos.

En la mayoría de los casos, las bases de datos NoSQL surgieron y se mantienen como proyectos de código abierto.

Clasificación

A pesar de compartir muchas características, existen otras que permiten diferenciarlas y usarlas para realizar una clasificación de las bases de datos NoSQL. No existe una única clasificación, sin embargo, un elemento que sí permite realizar una diferenciación es el modelo de datos subyacente.

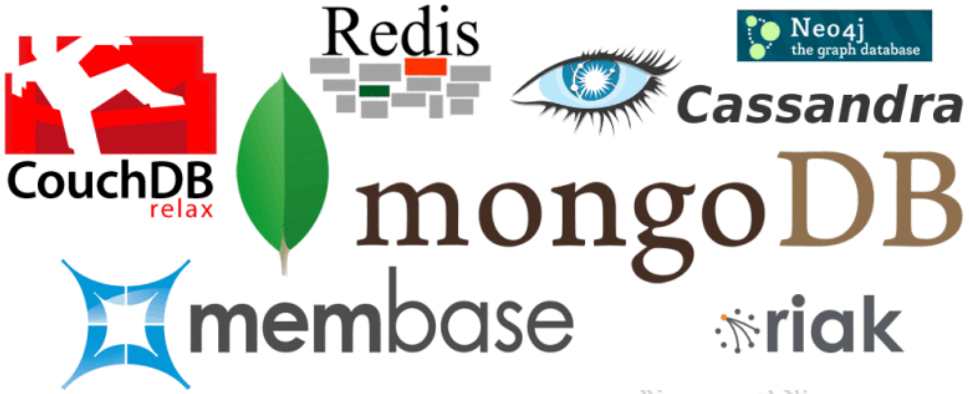

Figura 2. Familias de bases de datos NoSQL.
Fuente: elaboración propia a partir de los logos oficiales.

Por lo que se puede hablar de las siguientes familias (véase la figura 2):

- Bases de datos clave-valor: Riak, Redis, Dynamo, Voldemort.

- Bases de datos orientadas a documento: MongoDB, CouchDB.
- Bases de datos basadas en columnas: Cassandra, Hypertable, HBase, SimpleDB.
- Bases de datos de grafos: Neo4J, Infinite Graph.

Hay que destacar que esta clasificación es artificial, pues muchas de las bases de datos que están clasificadas en diferentes familias, sin embargo, comparten características, por lo que igual de correcto habría sido clasificarlas en una familia u otra. Por ejemplo, MongoDB es una base de datos tipo documental, sin embargo, una de las formas de acceder a la información es mediante la clave de los campos de información, por lo que sería correcto clasificarla como base de datos clave-valor.

Características

Una de las características de las bases de datos NoSQL, que supone un mayor cambio con respecto a las bases de datos relacionales, es la falta de necesidad de definir un esquema de la estructura de la información que se va a almacenar en la misma. El origen de esta característica se encuentra en las condiciones de procesamiento de la información que se producen en el ámbito del Big Data. En este contexto, el tipo de datos generados pueden ser semiestructurados o no estructurados, se requiere agregar conjuntamente datos que son heterogéneos, en muchos casos, los datos que se necesita almacenar pueden tener errores o puede haber campos de información vacíos, los datos se generan a una gran velocidad y en cantidades enormes. Estas condiciones son incompatibles con el tipo de información para el que está diseñada una base de datos relacional, dado que se esperan datos que respondan a una estructura de la información previamente fijada, se agregarán datos homogéneos, las cantidades de datos esperadas no son tan enormes y su inclusión en las tablas se realiza, en general, de una manera planificada. Es cierto que se podrían utilizar las bases de datos relacionales para estas situaciones, sin embargo, en este caso, se necesitarían tablas con muchos campos para poder guardar cualquier tipo de información y las tablas resultantes después de almacenar la información tendrían muchos campos vacíos, concretamente las filas de datos heterogéneos. Por ello no es una buena idea. Esta disonancia con el modelo relacional dio lugar a esta característica que se encuentra presente en las bases de datos NoSQL y que representa la necesidad de disponer de una flexibilidad sin límites para poder almacenar lo que sea necesario, ya que no puede predecir cómo será la información que se deberá gestionar. Sin embargo, esta característica tiene precisamente la ventaja de poder almacenar cualquier elemento de información, sin estar sujeto a ningún tipo de restricciones y puede ser adaptada a diferentes situaciones, lo cual es muy útil en el contexto del Big Data (que es por naturaleza dinámico), tal como se ha argumentado anteriormente.

Desventajas

Pero también presentan algunas desventajas. La principal es el traspaso de responsabilidades, que antes recaían en los sistemas de gestión de bases de datos, hacia el programador. Así, algunas operaciones que se hacían de manera transparente, como mantener la consistencia de los datos cuando se actualizan o se borran, asegurar que los datos introducidos son de los tipos esperados, gestionar la concurrencia del acceso a los datos, así como otras, ahora deben ser realizadas directamente desde los programas que procesan la información. Esto tiene también como problema que, para acceder a los datos, es necesario que el programador conozca qué tipo de información hay y cómo se encuentra estructurada en la base de datos. Asimismo, esta situación tiene otra consecuencia directa: la representación implícita del esquema de la información en el código de los programas que procesan la información. Hay que tener en cuenta que este esquema, en el mundo relacional, es gestionado automáticamente por el sistema de gestión de bases de datos, mientras que, en el mundo NoSQL, debe ser gestionado manualmente por el programador, con las implicaciones negativas que esto supone. Algunas consecuencias de esta situación son, por ejemplo, que cambios en el contenido de la base de datos producirán cambios en el código, es decir, que se crea una fuerte dependencia entre el código y sistema de persistencia. Esta dependencia con respecto al código redunda negativamente en la calidad del código, ya que disminuye el grado de reutilización de los códigos de programación, pues están particularizados para una situación determinada. Además, cuando se desea conocer qué información se tiene almacenada y cómo se encuentra relacionada es necesario consultar el código de la aplicación. El borrado consistente es otro de los grandes problemas que se producen, ya que puede ocurrir que la ejecución de una operación de esta naturaleza esté borrando información relacionada, entre la cual existe una dependencia, con lo que se merma la consistencia de la información. De igual forma puede ocurrir que borrados sucesivos dejen información aislada a la que no se puede acceder (basura) y, por tanto, sea necesario realizar una limpieza de la base de datos para eliminar estos elementos de información que ya no son útiles. Otras consecuencias son la imposibilidad de crear procedimientos programados para comprobar restricciones de la información que se almacena.

Razones

Por último, hay que comentar algunas de las razones que han empujado hacia la implantación y uso de este tipo de bases de datos NoSQL. En primer lugar, han supuesto una solución al problema de la impedancia. La posibilidad de almacenar datos con cualquier tipo de estructura hace innecesarias las transformaciones de información para adecuar la información gestionada por las aplicaciones que procesan la información de la base de datos y las estructuras de datos que utilizan las bases de datos para almacenarla. Como ya se comentó, esto suponía un coste a tener en cuenta, puesto que parte del código de la aplicación había que dedicarlo a esta tarea. Otro factor de éxito se debe a la posibilidad de procesar grandes cantidades de datos de una forma más barata y eficaz, ya que las bases de datos pueden ejecutarse en soluciones distribuidas de tipo clúster de máquinas que permiten escalar los sistemas mediante la adición de nuevas máquinas al clúster.

VII. Bases de datos orientadas hacia agregados

Las familias de bases de datos NoSQL documentales, clave-valor y orientadas a columnas están orientadas hacia agregados. En este contexto, un agregado representa una estructura de datos compleja que mantiene un conjunto de datos relacionados entre sí. Esta estructura se puede corresponder con estructuras de datos típicas de los lenguajes de programación, tales como listas, registros de datos anidados, etc. Los agregados son tratados como una unidad no descomponible a efectos de su procesamiento o de la gestión de la consistencia. Así, las operaciones que se realizan sobre un agregado se consideran atómicas y, desde el punto de vista operativo, las operaciones toman agregados y dan como resultados agregados.

Esta organización de la información conlleva algunas ventajas, como la gestión de los clústeres, pues se utilizan los agregados como unidad de replicación y distribución de los datos, y se manipula la información desde los programas, puesto que, al ser estructuras similares a las usadas en los lenguajes de programación, son más fáciles de manipular y gestionar desde los propios programas.

En relación con la gestión de la consistencia de la información, hay que observar que, aunque este tipo de bases de datos no admiten transacciones al estilo ACID sobre un conjunto de agregados, sin embargo, tal como se ha comentado más arriba, las operaciones sobre agregados individuales se realizan de forma atómica, de un agregado cada vez. En este sentido, si se quisiera realizar operaciones atómicas sobre un conjunto de agregados, sería necesario implementarlo desde los programas que los gestionan.

Desde un punto de vista semántico, cuando se tiene un elemento de información complejo, si se usa una base de datos relacional, es necesario descomponer esa información en elementos de información más simples con tipos básicos, para poder ser almacenados. Y, de igual forma, para devolver la información, habrá que reconstruir las relaciones existentes en la información. Sin embargo, en un modelo orientado hacia agregados, no será necesario este proceso de descomposición de la información, dado que lo que se gestiona directamente son los agregados, es decir, se recuperan agregados, se procesan agregados y se devuelven agregados. No obstante, hay que tener en cuenta que el uso de agregados dificulta la posibilidad de definir agregados que puedan ser utilizados en diferentes contextos.

Otra característica común a todas las bases de datos orientadas hacia agregados son las técnicas de diseño, ya que, en todos los casos, el diseño se centra en definir la estructura de datos más adecuada para representar la información que se quiere almacenar. Esta modelización se puede llevar a cabo usando, por ejemplo, la sintaxis de un documento JSON o bien utilizando una representación gráfica que usa rectángulos con campos de información que representan agregados o subagregados.

Por último, hay que comentar que existen varios tipos de agregados: aquellos a los que se puede acceder a la información almacenada y aquellos a los que no se puede acceder, pues se mantiene opaca.

VIII. Modelos de distribución en las bases de datos NoSQL

En las bases de datos NoSQL se pueden encontrar diferentes modelos para gestionar de forma distribuida los datos:

Servidor único

El primer modelo de distribución y más simple consiste en ejecutar la base de datos en un único servidor. Aunque la mayoría de las bases de datos NoSQL están pensadas para ser ejecutadas en un clúster, puede que en ciertas situaciones sea adecuado ejecutarlas en un único servidor, como ocurre en las bases de datos NoSQL de tipo grafo o si el uso de los datos tiene como objetivo realizar procesamientos sobre los agregados de datos en las bases de datos documentales o clave-valor.

Sharding

En algunas ocasiones las bases de datos están muy ocupadas porque hay muchos usuarios que están accediendo a diferentes partes del conjunto de datos. En estas circunstancias, se puede pensar en realizar un escalado horizontal, poniendo las diferentes partes de los datos en distintos servidores. Esta técnica recibe el nombre de sharding. En el caso ideal, habría diferentes usuarios que consultarían diferentes nodos-servidor. De esta forma, al interactuar cada usuario con un servidor diferente, se podrían obtener respuestas rápidas desde cada servidor, y la carga de trabajo se equilibraría muy bien entre los servidores (por ejemplo, si se tienen 10 servidores, cada uno de ellos solo tendría que manejar el 10% de la carga de trabajo). Con el fin de implementarlo, es necesario garantizar que los datos a los que se va a acceder de forma conjunta se agrupen en el mismo nodo, y que estas agrupaciones estén dispuestas en los nodos para proporcionar el mejor acceso a los datos. El primer problema que se plantea es cómo se pueden aglutinar los datos de forma que el usuario recupere sus datos desde un único servidor. Para conseguirlo, hay que diseñar los agregados de datos de forma que combinen los datos a los que se suele acceder conjuntamente, de manera que el agregado de datos se convierte en la unidad de distribución. Con respecto a cómo organizar los datos en los nodos, se deben tener en cuenta determinados factores como los que se exponen a continuación:

1. Si se sabe que la mayoría de los accesos a ciertos agregados se realiza en función de una ubicación física, entonces lo ideal será colocar los datos cerca de donde se está accediendo.
2. Otro factor a tener en cuenta es mantener la carga de trabajo. Se pueden organizar los agregados para distribuirlos uniformemente a través de los nodos, consiguiendo cantidades iguales de carga. Esta organización podría variar con el tiempo si, por ejemplo, se observa que determinados datos tienden a ser más visitados en ciertos días, para lo cual se pueden definir reglas específicas del dominio para ser usadas en estos casos.
3. En algunos casos puede ser útil colocar los agregados de datos juntos, si se espera que sean leídos en secuencia.

El sharding puede ser implementado, en cuanto a la lógica de la aplicación, dividiendo los datos en los fragmentos que sean necesarios. Sin embargo, esto complica el modelo de programación, dado que el código debe garantizar que las consultas se distribuyan a través de los diferentes fragmentos, y cualquier reequilibrio del sharding requerirá cambios en el código de la aplicación y migraciones de los datos. Para solventar este problema, muchas bases de datos NoSQL ofrecen autosharding, en donde es la propia base de datos la que tiene la responsabilidad de asignar los datos a diferentes fragmentos, y asegura, así, que el acceso a los datos se lleve a cabo en el fragmento correcto. La técnica del sharding es particularmente valiosa con respecto al rendimiento, lo mejora, tanto en lectura como en escritura. Por ejemplo, la replicación puede mejorar el rendimiento en cuanto a la lectura, pero mejora poco en aplicaciones donde se realizan muchas escrituras. En este sentido, el sharding proporciona un camino para escalar horizontalmente las escrituras. Con respecto a la recuperación ante fallos, el sharding no consigue mejoras importantes cuando se utiliza en solitario. Así, aunque los datos estén en diferentes nodos, si se produce un fallo en un nodo, la imposibilidad de acceso a los datos a través del fragmento correspondiente es la misma que si se usara una solución basada en un único servidor. El único beneficio con respecto a este aspecto es que únicamente los usuarios de los datos de ese fragmento serán los que no puedan acceder a sus datos. Sin embargo, no es una buena solución que una parte de la base de datos no sea accesible. Obsérvese que si se tiene un único servidor es más fácil pagar el coste y el esfuerzo de mantener el servidor en funcionamiento que con los clústeres, dado que en este caso existe una tendencia a usar máquinas menos fiables, por lo que aumenta la probabilidad de fallos en los nodos. Es por ello que el sharding tiende a decrementar la capacidad de recuperación ante fallos. Cuando se utiliza sharding, hay que tener en cuenta el diseño original de la base de datos. Si originalmente se diseñó para usar sharding, entonces lo mejor es ejecutarlas directamente sobre un clúster, y si fueron diseñadas para ejecutarse en un único servidor, entonces será mejor empezar a ejecutarlas en un único servidor, y en caso de que se den problemas con la carga, se usará sharding.

Replicación maestro-esclavo

La distribución maestro-esclavo consiste en replicar los datos a través de múltiples nodos, de manera que hay un nodo que actúa como nodo primario o maestro, el cual representa la fuente autorizada de los datos y es responsable de procesar cualquier modificación de los mismos. El resto de nodos son esclavos o secundarios y existe un proceso que los sincroniza con el nodo primario. Las principales ventajas de este modelo son:

1. Es muy útil para escalar cuando se tienen que realizar lecturas intensivas sobre los datos. Así, se puede escalar horizontalmente para mantener muchas solicitudes de lectura, añadiendo más nodos secundarios y asegurando que todas las peticiones sean dirigidas a los nodos secundarios. La única limitación se encuentra en la capacidad del nodo primario para procesar las actualizaciones y trasladarlas a los nodos esclavos. Hay que tener en cuenta que no es un buen esquema en situaciones donde exista un tráfico grande de escritura, aunque la descarga del tráfico de lectura ayude al manejo de la carga de escritura.
2. Otra ventaja es la capacidad de recuperación ante fallos. Si el nodo primario falla, entonces, al menos, los nodos clientes pueden manejar las peticiones de lectura. Esta situación es útil cuando la mayoría de los accesos son lecturas. Cuando falla el nodo primario, no es posible realizar escrituras hasta que se recupere dicho nodo o hasta que se elija un nuevo nodo primario de entre los secundarios (esta situación es rápida en este caso, ya que los nodos secundarios son réplicas del nodo primario). Obsérvese que esta posibilidad de reemplazar el nodo primario por uno secundario no solo es útil cuando se quiere escalar, también puede serlo en otras situaciones. Por ejemplo, se puede diseñar una solución de un único servidor con una gran capacidad de recuperación ante fallos, de forma que todo el tráfico de lectura y escritura se dirija al nodo primario, mientras que los nodos secundarios actuarán como copias de seguridad.

El nodo primario puede ser elegido manualmente o automáticamente. La elección manual significa que, cuando se configura el clúster de nodos, se elige un nodo como primario. En la elección automática, se crea el clúster de nodos y son los propios nodos los que eligen entre ellos al nodo primario. Esta opción es más simple y además permite que el clúster pueda elegir a un nuevo nodo primario cuando se produzca un fallo, reduciendo el tiempo de inactividad. Con el objetivo de conseguir una recuperación ante fallos en las lecturas, es necesario asegurar que los caminos de las lecturas y las escrituras sean diferentes, de manera que se pueda manejar un fallo en la escritura y aún se puedan realizar lecturas. Esto significa que hay que realizar las lecturas y escrituras en conexiones separadas a la base de datos. La principal desventaja de este modelo es la inconsistencia. Existe el peligro de que diferentes clientes lean diferentes nodos secundarios y que los datos sean diferentes, debido a que determinados cambios no se han propagado aún a los nodos secundarios. En el peor de los casos, un cliente no podrá leer una escritura que acaba de realizar. Incluso esto puede ser un problema en una configuración de un solo servidor con nodos que son copias de seguridad, pues, si el nodo primario falla, algunas actualizaciones se perderán, ya que no pasarán a los nodos secundarios.

Replicación peer-to-peer

El modelo de replicación maestro-esclavo ayuda en la escalabilidad de lectura, pero no en la escalabilidad de escritura. Además, proporciona la recuperación ante fallos de un nodo secundario, pero no para el nodo primario. Esencialmente, el nodo primario sigue siendo un cuello de botella y un único punto de fallos. La replicación peer-to-peer ataca este tipo de problemas eliminando la figura del nodo primario. Todas las réplicas tienen la misma importancia y pueden aceptar escrituras, y, si alguna de las réplicas falla, no impide el acceso a la base de datos. Además, es posible agregar nuevos nodos para mejorar el rendimiento.

El mayor problema que presenta este modelo es la inconsistencia. Cuando se puede escribir en dos lugares diferentes, existe el riesgo de que dos personas intenten actualizar el mismo registro al mismo tiempo, produciéndose un conflicto de escritura-escritura. Obsérvese que la inconsistencia en la lectura llevará a problemas transitorios, sin embargo, las inconsistencias en las escrituras son para siempre. Para evitar estos problemas tenemos dos aproximaciones:

1. Cada vez que se escriben datos, hay que coordinar las réplicas para evitar conflictos. Esto ofrece una garantía como la del modelo del nodo primario a costa del tráfico de red para coordinar las escrituras. Téngase en cuenta que no se necesita que todas las réplicas lleguen a un acuerdo, solo la mayoría, para que puedan sobrevivir aun perdiendo una minoría de los nodos replicados. En esta aproximación se valora más la consistencia que la disponibilidad.
2. Otra opción es considerar una escritura inconsistente. Existen situaciones en las que se puede seguir una política de fusionar escrituras inconsistentes. Se obtiene el beneficio del máximo rendimiento de la escritura sobre cualquier réplica. En esta aproximación se valora más la disponibilidad que la consistencia.

Combinando sharding y replicación

La replicación y el sharding son estrategias que pueden combinarse. Si se usan ambas técnicas, se tendrán múltiples nodos primarios, pero para cada ítem de datos solo habrá un nodo primario. Dependiendo de la configuración, se puede elegir que un nodo sea primario para algunos datos y secundario para otros, o bien tener nodos primarios o secundarios dedicados a determinados trabajos.

El uso de la replicación peer-to-peer y el sharding es una estrategia común en las bases de datos NoSQL de tipo columna. En este contexto, se podrían tener decenas o cientos de nodos en un clúster con datos fragmentados sobre ellos. Un buen punto de partida para la duplicación peer-to-peer consiste en tener un factor de replicación de 3, de forma que cada fragmento está presente en 3 nodos y, si un nodo falla, los fragmentos de dicho nodo serán reconstruidos sobre los otros nodos.

IX. El teorema cap

El teorema CAP (Consistency, Availability, Partition tolerance) formaliza algunas propiedades que se cumplen en un sistema distribuido ejecutado en un clúster de máquinas. Establece que, dado un sistema en las condiciones anteriores, solo se podrán cumplir a la vez dos de las siguientes posibles propiedades: consistencia de los datos gestionados, disponibilidad de acceso a los datos almacenados en las máquinas que forman el clúster y tolerancia al particionamiento del sistema.

Se puede analizar el teorema en varios casos:

Si el sistema está formado por una única máquina

En este caso, si la máquina funciona correctamente, la información estará disponible y se podrá garantizar la consistencia de la información almacenada. Además, no es posible que se produzca un particionamiento del sistema.

Si el sistema está formado por más de una máquina

En este caso, si se produjera un particionamiento del sistema, se interpreta que se mantiene la disponibilidad, siempre que, cuando se realice una petición a una máquina que esté en funcionamiento, se reciba una respuesta. Así pues, las particiones tienen como consecuencia una compensación entre la consistencia y la disponibilidad del sistema, es decir, se mantendrán algunas inconsistencias del sistema a cambio de incrementar la disponibilidad del sistema. Obsérvese que este equilibrio entre consistencia y disponibilidad está directamente relacionado con la latencia de un sistema. Así, cuantas más maquinas existen en un sistema, mejor es la consistencia, pero peor es la disponibilidad, ya que aumenta la latencia del sistema.

Quorum de escritura y Factor de replicación

En relación con la consistencia, hay que observar que cuantas más máquinas participan en una petición, mayor es la posibilidad de evitar una inconsistencia, lo que lleva a preguntarse cuántas máquinas son necesarias para conseguir una fuerte consistencia.

Supónganse algunos datos replicados sobre 3 nodos. No es necesario que todas las máquinas reconozcan una escritura para asegurar una fuerte consistencia, solo es necesario sobre 2 de ellos, es decir, una mayoría. Si se tienen escrituras conflictivas, solo uno puede conseguir la mayoría. Esto se conoce como "**quorum de escritura**" y se expresa en la siguiente ecuación: W > N/2, lo que significa que el número de máquinas participantes en la escritura debe ser mayor que la mitad del número de máquinas que participan en la replicación(N). El número de réplicas es a menudo conocido por el **factor de replicación**.

Quorum de lectura

Existe también la noción de "**quorum de lectura**", es decir, cuántas máquinas han de contactar para asegurarse de que se tiene el dato más actualizado. Este caso es más complicado, pues depende de cuántas máquinas necesitan confirmar una escritura.

Considérese un factor de replicación de 3. Si todas las escrituras necesitan dos máquinas para confirmar W=2, entonces se necesita contactar con al menos dos máquinas para asegurarse de que se consigue el dato más actualizado. Pero si las escrituras son solo confirmadas por un único nodo (W=1), entonces se necesitará contactar con las tres máquinas para asegurar que se tiene la última actualización. En este caso, puesto que no se tiene quorum de escritura, podría existir un conflicto de actualización, aunque contactando con suficientes lectores, se podría estar seguro de detectarlo. De esta forma, se pueden conseguir lecturas fuertemente consistentes, incluso si no se tiene una fuerte consistencia en las escrituras.

Esta relación entre el número de máquinas que deben contactar para una lectura (R), aquellas que confirman una escritura (W) y el factor de replicación puede ser capturada en una inecuación que dice que se puede tener una consistencia de lectura fuerte, si R+W>N.

Estas inecuaciones están escritas para un modelo de distribución peer-to-peer. Ahora bien, si se tiene un modelo de distribución maestro-esclavo, entonces solo hay que escribir a un nodo maestro para evitar conflictos escritura-escritura, y en cuanto a la lectura solo hay que hacerlo desde el nodo maestro para evitar conflictos lectura-escritura.

Obsérvese que es fácil confundir el número de máquinas en el clúster con el factor de replicación, y estos valores, a menudo, son diferentes. De hecho, en general, se sugiere que un factor de replicación de 3 es suficiente, dado que permite que un simple nodo falle, mientras que aún se mantiene el quorum para las lecturas y escrituras. Si se tiene rebalanceo automático, no se tardará mucho en crear una tercera réplica, y las posibilidades de perder la segunda réplica antes de ser reemplazada son muy pequeñas.

El número de máquinas que participan en una operación puede variar con la operación. Cuando se escribe, podría ser necesario un quorum para algunos tipos de actualizaciones, pero no para otras, dependerá de cuánto se valore la consistencia y disponibilidad. De forma similar, una lectura necesita velocidad, pero puede tolerar falta de actualización si contacta con menos máquinas.

Con frecuencia es necesario tomar en cuenta ambos aspectos. Si se necesita rapidez, lecturas fuertemente consistentes, entonces podría ser necesario que las escrituras fueran reconocidas por todas las máquinas, permitiendo que las lecturas contacten solo con una máquina (N=3, W=3, R=1). Eso significaría que las escrituras son lentas, puesto que tienen que contactar con las tres máquinas y no serían tolerantes a la pérdida de una máquina. Pero en algunas circunstancias puede que sea una desventaja hacerlo.

X. Resumen

En esta unidad se ha realizado una introducción a las bases de datos NoSQL. En primer lugar, se ha introducido el concepto de Big Data y sus implicaciones con respecto a la gestión, almacenamiento y procesamiento de datos, así como las herramientas tecnológicas que han aparecido como consecuencia del Big Data. Esto ha servido de introducción para, a continuación, describir la utilidad y ventajas que ofrecen las bases de datos relacionales y cómo estas han sido el mecanismo de persistencia de datos que ha imperado en las últimas décadas en los sistemas de persistencia. Se ha comentado su sistema de control de la concurrencia y su capacidad para garantizar la consistencia de la información mediante el uso de transacciones ACID. Asimismo, la existencia del lenguaje de consultas SQL supone un elemento de estandarización que ha ofrecido grandes ventajas para su expansión.

A continuación, se han expuesto las limitaciones que presentan las bases de datos con respecto a las necesidades de persistencia y procesamiento de datos. En particular, se ha destacado su limitación para poder ser ejecutadas en entornos distribuidos, así como para almacenar tipos de datos que no son estructurados o la necesidad de agregar datos. Esto ha generado la necesidad de la aparición de las bases de datos NoSQL.

Después se han introducido las bases de datos NoSQL como una respuesta a las limitaciones de las bases de datos NoSQL. Todas ellas comparten un conjunto de propiedades, tales como no utilizar como lenguaje de consultas SQL, estar diseñadas, en la mayoría de los casos, para ser ejecutadas en ambientes distribuidos y que son proyectos de código abierto. Entre todas las propiedades, se ha prestado especial atención a una de ellas: la no necesidad de usar un esquema predefinido para establecer la estructura de los datos que se van a almacenar. Se han mostrado las consecuencias que se derivan de esta característica, tales como la flexibilidad para almacenar datos de cualquier tipo y adaptarse fácilmente a diferentes situaciones. Sin embargo, también presenta como desventaja la necesidad de utilizar un esquema implícito en el código, y la dependencia que se genera entre el código y los cambios que se producen en las bases de datos. Esencialmente, se produce un traslado de responsabilidades del sistema de gestión de bases de datos hacia el programador. También se han introducido las diferentes familias de bases de datos NoSQL. En particular, hay un conjunto de estas familias que se agrupan en las denominadas bases de datos orientadas hacia agregados, las cuales utilizan como unidad de almacenamiento una entidad denominada agregado, que representa un dato con una estructura compleja.

Seguidamente, se han comentado otras características de las bases de datos, tales como el teorema CAP, que establece un conjunto de propiedades acerca del comportamiento de los sistemas distribuidos con respecto a las propiedades de la consistencia, la disponibilidad y el particionamiento del sistema. Asimismo, se han comentado los diferentes modelos de distribución de datos que se pueden definir en las bases de datos NoSQL, los cuales dependen de los roles que pueden ir tomando los nodos o máquinas que forman el sistema distribuido, desde la existencia de una máquina que toma el rol de maestro, y el resto son máquinas esclavas, hasta sistemas donde todas las máquinas tienen las mismas responsabilidades y ninguna de ellas destaca sobre las demás.

Ejercicios

Ejercicio

Se desea diseñar una aplicación que implemente un buscador de ofertas de trabajo. Cada usuario puede configurar qué portales de ofertas de empleo servirán de fuente para recuperar las ofertas y las características de las ofertas que deben recuperarse. La aplicación se conectará a los portales de empleo y generará entradas asociadas a cada usuario. Una entrada será esencialmente el nombre del portal web donde se ha encontrado la oferta, el contenido principal de la oferta, las características que cumple la oferta, con respecto a los requisitos impuestos por el usuario, y un enlace a la misma (la aplicación guardará una copia de la oferta para asegurarse de que no sea eliminada del portal de empleo), de forma que si el usuario desea consultar la oferta completa puede pulsar sobre el enlace. Obsérvese que las entidades que deberán gestionarse, en cuanto al número de portales de empleo, requisitos de búsqueda, ofertas y usuarios, van a ser enormes.

Se pide realizar un análisis acerca del modelo de datos y la arquitectura de distribución y replicación de los datos más adecuado para los requisitos de la aplicación descrita. En el caso del modelo de datos, se pide discutir la conveniencia de un modelo relacional o uno NoSQL y, en este último caso, qué modelo sería el más adecuado.

Solución

En primer lugar, el modelo de datos no debería ser relacional dado que, tal como se describe en el problema, el sistema requiere gestionar una cantidad enorme de datos y tendrá que soportar una elevada concurrencia de acceso por muchos usuarios a las ofertas recuperadas. Esta situación hace recomendable utilizar una fragmentación horizontal de los datos. Por tanto, la mejor opción sería utilizar una base de datos NoSQL, que además de facilitar la fragmentación horizontal de los datos, ofrece alta disponibilidad y soporta elevados niveles de concurrencia. De los posibles modelos de bases de datos NoSQL, encajarían mejor los modelos de tipo clave-valor, orientados a documentos u orientados a columnas. Un modelo orientado a grafos no sería una buena solución, puesto que en el sistema propuesto apenas existen relaciones.

De acuerdo con las condiciones planteadas, la mejor forma de distribuir los datos sería en función de los lugares donde viven los usuarios registrados. De esta forma, se maximiza la localidad de los datos y se asegura un acceso rápido a los mismos. Esta decisión se debe a que normalmente los usuarios buscan las ofertas de viajes en portales web de los países o regiones donde viven. Por tanto, se realizará una fragmentación horizontal que cumpla las propiedades de completitud y reconstrucción, de acuerdo con los lugares en los que exista un número elevado de usuarios registrados en la aplicación. Con el objetivo de asegurar cargas equilibradas de trabajo en todas las particiones, aquellos lugares que no tengan un número no suficiente de usuarios registrados se almacenarán en el mismo nodo. Por último, se realizarán redistribuciones de la fragmentación cuando se produzcan cambios en el número de usuarios registrados en las diferentes particiones.

Respecto a la replicación, se replicará al menos una vez con el objetivo de asegurar el servicio en caso de caída de un nodo. Además, teniendo en cuenta que se está utilizando una arquitectura de distribución basada en la localización, el lugar óptimo para almacenar las réplicas serán los nodos geográficos más cercanos, así se minimizará el tiempo de acceso a los datos. Por otra parte, de entre los posibles modelos de replicación, se utilizaría un modelo maestro-esclavo puesto que permite garantizar un alto rendimiento cuando hay un elevado número de peticiones a la base de datos.

Recursos

Bibliografía

- ***Getting Started with NoSQL*** : Vaish, G. Packt Publishing Ltd; 2013
- ***Nosql Database: New Era of Databases for Big Data Analytics-Classification, Characteristics and Comparison.*** : Moniruzzaman, A. B. M. y Hossain, S. A. ArXiv preprint arXiv: 1307.0191. 2014.
- ***NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence*** : Pramod, J. S. y Fowler, M. Addison-Wesley Professional; 2012
- ***Professional NoSQL.*** : Tiwari, S. John Wiley & Sons; 2011.
- ***Seven Databases in Seven Weeks: A Guide to Modern Databases and the NoSQL Movement***: Redmond, E. y Wilson, J. R. Pragmatic Bookshelf; 2012.
- **Relational vs. NoSQL Databases: A Survey**: Mohamed, M. A.; Altrafi, O. G. e Ismail, M. O. *International Journal of Computer and Information Technology*(ISSN: 2279–0764). Volumen 3; 2014

Glosario.

- **Base de datos NoSQL**: es una base de datos que no está diseñada utilizando como base el modelo relacional. Entre otras características, se destaca que no usa el lenguaje SQL como lenguaje de consultas.

- **Base de datos orientada hacia agregados**: es un conjunto de bases de datos que se caracteriza porque utilizan como unidad de almacenamiento un elemento de información denominado agregado. Un agregado representa un conjunto de datos que presentan una estructura de la información compleja.

- **Base de datos relacional**: es una base de datos que está diseñada utilizando como base el modelo relacional. Su principal característica es que organiza la información usando como unidad de almacenamiento las tablas. Asimismo, utiliza SQL como lenguaje de consultas.

- **Big Data**: es un fenómeno que implica un conjunto de herramientas, tecnologías y técnicas que tienen como objetivo procesar enormes cantidades de datos de todo tipo con el objetivo de explotar la información contenida y obtener algún tipo de ventaja.

- **Consistencia**: se refiere al mecanismo que asegura que los datos que se recuperan de una base de datos son válidos y se encuentran actualizados.

- **Disponibilidad**: se refiere a la forma de asegurar que los datos almacenados en una base de datos se encuentren accesibles en un momento dado.

- **Escalado**: es la forma en que se puede aumentar la capacidad de procesamiento de un sistema. Hay dos tipos: escalado vertical y escalado horizontal. El primer caso consiste en adquirir una máquina con mejores prestaciones que la anterior, mientras que el segundo caso consiste en crear un clúster de máquinas que colaboran entre sí, de manera que se suman las potencias de procesamiento de todas las máquinas.

- **Esquema de una base de datos**: consiste en una definición de los tipos de datos que se van a almacenar. Se especifican las restricciones, condiciones o tipo que deben cumplir. Además, el esquema se define previamente a la introducción de los datos.

- **Esquema implícito**: es la representación de la estructura de la información que se encuentra almacenada en una base de datos NoSQL, dentro del código del programa que la gestiona, debido a la no existencia de un esquema previamente prefijado en la base de datos.

- **Latencia**: se refiere al retardo que se produce en un sistema en el envío de la información.

- **Lenguaje SQL**: es un lenguaje declarativo que se utiliza en las bases de datos relacionales para realizar consultas.

- **Problema de la impedancia**: este problema se refiere a las diferentes estructuras de datos que se usan en las bases de datos relacionales y en los lenguajes de programación, de manera que hacen necesario realizar transformaciones de la información cuando se quieren intercambiar datos.

- **Propiedades ACID**: es un conjunto de características que aseguran que las transacciones de información se realizan de manera consistente, aislada, atómica y durable.

- **Replicación**: consiste en realizar una copia de los datos en más de una máquina de un sistema distribuido con el objetivo de garantizar que no se pierden los datos en el caso de la caída de alguna de las máquinas.

- **Replicación maestro-esclavo**: es una forma de replicación de los datos a través de múltiples nodos, de manera que hay un nodo que actúa como nodo primario o maestro, el cual representa la fuente autorizada de los datos y es responsable de procesar cualquier modificación de los mismos.

- **Replicación peer-peer**: es una forma de replicación que se caracteriza porque todas las réplicas tienen la misma importancia y pueden aceptar escrituras y, si alguna de las réplicas falla, no impide el acceso a la base de datos.

- **Sharding**: es una organización de la información que consiste en distribuir los datos entre un conjunto de máquinas, de manera que en las máquinas se pueden definir roles, como que una máquina actúe de servidor y el resto de clientes o que todas las máquinas actúen con los mismos roles.

- **Teorema CAP**: es un teorema que establece un conjunto de propiedades acerca de los sistemas distribuidos con respecto a la disponibilidad, consistencia de los datos o el particionamiento del sistema.

- **V´s (o uves) del Big Data**: se refiere a las características principales que cumple la información en el contexto del Big Data, las cuales son: velocidad, valor, volumen, variedad y veracidad.