UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf
Escola Tècnica Superior d'Enginyeria Informàtica

Escuela Técnica Superior de Ingeniería Informática
Universitat Politècnica de València

Estudio comparativo de BBDD relacionales y NoSQL en un entorno industrial

Proyecto Final de Carrera

Ingeniería Informática

Autor: Vicente Jesús Bas Abad

Director: Francisco Daniel Muñoz Escoí

18 de septiembre de 2015

Resumen

El presente proyecto final de carrera pretende demostrar si un sistema de base de datos no relacional (NoSQL), es capaz de superar el rendimiento de un sistema relacional (SQL) en una aplicación industrial. Esta aplicación desarrollada por Ensai Solutions S.L para la empresa Aguas de Valencia S.A, de nombre Regedate, se encarga de recopilar diariamente todos los datos de los equipos remotos situados en pozos, depósitos y contadores.

Estos datos son guardados en la base de datos, para posteriormente ser utilizados por los usuarios para generar distintos tipos de informes utilizados en la toma de decisiones a largo plazo.

Esta aplicación “big data” presenta un problema de rendimiento a la hora de mostrar los datos al usuario. Este problema de rendimiento, viene dado por la relación entre la forma en la que el sistema gestor de base de datos almacena la información y la forma en que se muestra al usuario. Mensualmente, Regedate crea una nueva tabla, donde llega a almacenar 100.000.000 de entradas. Si además hay que combinar y ordenar varias tablas, el alto coste es obvio.

Para solucionar esta limitación, buscaremos un sistema gestor de base de datos no relacional (NoSQL), que sea capaz de permitirnos almacenar la información estructurada del modo más cercano posible a la forma en que se muestra al usuario. Consiguiendo así evitar el sobre coste computacional que tiene actualmente la aplicación para estructurar, juntar y ordenar los resultados de cada consulta.

El primer punto a salvar será la elección del sistema gestor de bases de datos. Con el fin de conseguir una mayor flexibilidad a la hora de almacenar la información compararemos las diferentes opciones en el mercado y evaluaremos parámetros buscando el sistema que mejor se adapte a las necesidades del proyecto. Para este análisis, estudiaremos los diferentes tipos de bases de datos no relacionales y preseleccionaremos un pequeño grupo, sobre las que evaluaremos:

- Rendimiento
- Teorema CAP
- Características ACID
- Integración con .NET (plataforma sobre la que se construye Regedate)
- Tipo de licencia
- Etc

Una vez evaluadas las diferentes opciones de bases de datos y seleccionada una de ellas. El siguiente punto comprenderá la fase de estudio donde se mostrarán alternativas a la estructura de almacenamiento actual. Buscaremos diferentes opciones que se ajusten a la estructura en que la aplicación muestra los datos. De este modo, evitaremos JOINS y transposiciones sobre el sistema gestor de base de datos. Finalmente, evaluaremos mediante mediciones de tiempo estas alternativas y las compararemos con la versión existente. Así podremos concluir si realmente es interesante y ofrece un incremento de rendimiento a la aplicación esta nueva implementación.

Palabras clave: SQL, NoSQL, Couchbase, comparación, base, de, datos, resultados, análisis, estudio

Agradecimientos

Desde aquí quisiera dar las gracias a la empresa Aguas de Valencia, por poder formar parte de su plantilla y llevar a cabo este proyecto. Dentro de Aguas de Valencia, dar las gracias a todos aquellos que de una u otra forma se han visto involucrados, en especial a Jorge Bayo Arribas co-director del proyecto.

Agradecer de igual modo a Francisco Daniel Muñoz Escoí, director del proyecto en la universidad por su alta implicación.

Tabla de contenidos

Introducción

El presente proyecto final de carrera pretende buscar soluciones que mejoren el rendimiento en un sistema de minado de datos industrial. Esta aplicación “big data” presenta un problema de rendimiento a la hora de mostrar los datos al usuario. Este problema de rendimiento viene dado por la relación entre la forma en la que el sistema gestor de base de datos almacena la información y la forma en que se muestra al usuario.

Para solucionar esta limitación, buscaremos un sistema gestor de base de datos que sea capaz de permitirnos almacenar la información estructurada del modo más cercano posible a la forma en que se muestra al usuario. Consiguiendo así evitar el sobre coste computacional que tiene actualmente la aplicación para estructurar los resultados de cada consulta.

El primer punto a salvar será la elección del sistema gestor de bases de datos. Compararemos las diferentes opciones en el mercado y evaluaremos parámetros buscando el sistema que mejor se adapte a las necesidades del proyecto. Una vez evaluadas las diferentes opciones de bases de datos, el siguiente punto comprenderá la fase de estudio donde se mostrarán alternativas a la estructura de almacenamiento actual. Finalmente, evaluaremos mediante mediciones de tiempo estas alternativas y las compararemos con la versión existente. Así podremos concluir si realmente es interesante y ofrece un incremento de rendimiento a la aplicación esta nueva implementación.

Motivación

El actual proyecto nace de la necesidad de buscar opciones de almacenamiento en un sistema “big data”, tratando de solucionar posibles problemas de rendimiento a la hora de generar informes para los usuarios.

También partimos de la búsqueda de respuesta a la pregunta: ¿Puede una empresa pequeña utilizar bases de datos no relacionales de forma óptima? Existen grandes empresas como Google o Facebook, que han conseguido adaptar este tipo de bases de datos a sus necesidades. En el caso de Facebook, es obvio que para mostrar un “muro” les resulta mucho más cómodo que la base de datos devuelva un documento con toda la información. Mientras

que en el caso de Google, más similar al nuestro, han desarrollado una base de datos específica, conocida como “BigTable”, totalmente adaptada a sus necesidades de minería de datos. Pero volviendo a la pregunta planteada, ¿Seremos capaces de adaptar los sistemas no relacionales que actualmente se encuentran en el mercado ofreciendo un mejor rendimiento del que se puede conseguir con una base de datos tradicional? ¿O por el contrario, este auge de las bases de datos no relacionales es producto del bombo que han recibido de los mismos usuarios/desarrolladores?

Entorno

Se ha desarrollado bajo un marco de convenio de prácticas con el grupo Aguas de Valencia. Esta empresa se encuentra en el momento de implantación de un sistema de minado de datos que recoge valores de diferentes señales en pozos, depósitos, contadores, bombas, variadores, etc. Actualmente, se está utilizando una solución tecnológica clásica, con bases de datos relacionales. Dependiendo de las necesidades de los usuarios, este tipo de bases de datos puede suponer un cuello de botella a la hora de ejecutar la aplicación.

Objetivo

Haciendo uso de bases de datos no relacionales, buscaremos la forma de incrementar el rendimiento de la aplicación en el momento de mostrar datos al usuario sin la necesidad de modificar el hardware del que se dispone actualmente.

Resumen

- **Capítulo 1**: Introducción detallada al proyecto.
- **Capítulo 2**: Descripción de la aplicación de minado de datos y la funcionalidad de la base de datos.
- **Capítulo 3**: Tecnologías
- **Capítulo 4**: Comparativa de bases de datos no relacionales.
- **Capítulo 5**: Importación de datos a Couchbase.
- **Capítulo 6**: Emulando Regedate.
- **Capítulo 7**: Resultados.
- **Capítulo 8**: Conclusiones.

¿Qué es Regedate?

Regedate, desarrollado por Ensai Solutions S.L, es un sistema de adquisición y explotación de información industrial. Su funcionalidad básica es la de recopilar información en sistemas de telemando y ofrecer al usuario, la posibilidad de obtenerla de forma estructurada según sus necesidades.

Está dividido en dos bloques:

- Instalación industrial. Encargada de la recolección de datos. En esta parte se obtienen datos de los sistemas SCADA, se consolidan y son enviados al bloque superior.
- Sistema central. Permite la administración del sistema, la integración de los datos recolectados en el bloque inferior y la generación de vistas, tablas e informes por los usuarios. A esta parte la conoceremos como explotación.

El objetivo final de Regedate, es la generación de informes en ficheros formato Excel. Estos ficheros pueden ser generados por cualquier usuario de la aplicación, mostrando información de las variables en periodos horarios, diarios o mensuales. Su función principal, es servir como ayuda en la toma de decisiones a largo plazo.

¿Como funciona Regedate?

Cuando un nuevo telemando es integrado en el ecosistema Regedate, existen varios pasos a realizar para poder generar informes. Primero seleccionaremos las variables a almacenar. El número de variables en los telemandos es bastante dispar, existen casos como el de Calpe, donde llega a las 4500 variables, ya que necesita recoger señales de un elevado número de pozos y depósitos. En el otro extremo, encontramos el caso de Benifayó que no supera las 400. Una vez seleccionadas las variables necesarias para el informe, estableceremos el periodo de muestreo de estas y ajustaremos el formato del fichero Excel a nuestras necesidades.

Integración del SCADA

Para la integración del SCADA en Regedate, se instalan tres servicios en el equipo. Trabajando conjuntamente son capaces de recopilar los datos del telemando y enviarlos al servidor central de Regedate.

Configuración del agente

Con los servicios instalados, el siguiente paso es la configuración del agente. En él, seleccionaremos las variables que serán guardadas por Regedate. En este punto es mejor pecar por exceso que por defecto, las variables que no sean seleccionadas, no tendrán ningún dato registrado, y si se diese el caso de que en algún momento las necesitásemos, solo podríamos trabajar con datos desde el instante de tiempo en que solicitásemos a Regedate su indexación.

A estos datos o variables, se les pueden establecer alias para ser identificados fácilmente en pasos posteriores, solucionando de este modo, el problema de la ambigüedad en el nombre de algunas señales. Regedate, también permite establecer límites inferiores y superiores a estos valores. Si un valor supera el límite superior, el sistema asignará a dicha variable el valor del límite superior. Del mismo modo, si rebasa el límite inferior, adoptará el valor establecido como límite inferior.

En este paso, existen diferencias a la hora de escoger cómo llegan los datos entre los diferentes tipos de telemando. Mientras que los de tipo Siemens son capaces de recoger datos de modo casi instantáneo, los Kerwin de mejoras, recogen los datos una vez finalizado el día.

Creación de vistas

En Regedate, las vistas son agrupaciones de un conjunto de variables. Estas vistas son creadas por un usuario, que a su vez, puede compartirlas con otros usuarios. Pueden contener datos de uno o varios telemandos. Estos datos pueden ser del tipo bruto o estadístico. Además ofrece la posibilidad de definir fórmulas para obtener tipos de datos combinados dentro de la misma vista.

- Dato bruto: Son los datos tal cual leídos en el SCADA; no se aplica ningún tipo de operación. Estos datos tienen un límite temporal de 400 días. Con ellos no se pueden generar informes de larga duración.

- Datos estadísticos: Generados por Regedate para un intervalo de tiempo y con una operación de consolidación definida por el usuario. Esta operación puede ser del tipo media, máximo, mínimo o suma. No tienen ninguna restricción temporal. En Regedate se utilizan para generar informes con periodos superiores al año.

  En una misma vista, no existe la posibilidad de combinar los diferentes tipos de datos.

Regedate ofrece la posibilidad de visualizar estas vistas en formato tabla dentro de la misma aplicación.

Tablas personalizadas

Otra fuente de información son las tablas personalizadas. Estas permiten al usuario introducir datos en formato CSV, que posteriormente podrán ser utilizados en informes. También pueden ser compartidas.

Gráficas

Regedate cuenta con la posibilidad de generar series de gráficas. Estas permiten seleccionar variables de una vista, establecer el periodo de muestreo, y seleccionar el intervalo de tiempo. Tras ello obtendremos la representación gráfica en un modelo de líneas de los valores de la o las variables.

Informes

Objetivo final de Regedate. Un informe es la combinación de vistas y o tablas de usuario, con al menos uno de estos elementos. Cada una de estas vistas o tablas se debe definir para un intervalo de tiempo dado. Una misma vista, puede añadirse al informe repetidas veces, en intervalos diferentes.

Una vez los orígenes de datos de un informe están definidos, este puede ser ejecutado. Al ejecutar un informe, Regedate devuelve un fichero Excel con una hoja por cada elemento del informe. En el eje vertical de esta hoja, aparece la fecha y la hora en la que se tomó el dato o en la que se ha calculado. Mientras que en el eje horizontal, aparece cada una de las variables que compone dicha vista.

Con el fichero Excel que se obtiene de la ejecución del informe cabe la posibilidad de realizar un maquetado a modo de plantilla. Si esta plantilla es subida a Regedate, a partir de ese momento, cada vez que se ejecute el informe, aparecerá con la plantilla previamente definida.

Al igual que las vistas y las tablas, estos informes puede ser compartidos con otros usuarios, tanto en modo lectura como en modo escritura.

Aplicación NoSQL a Regedate

Para el almacenamiento de datos, Regedate utiliza una tabla en la que guarda todos los valores adquiridos de los diferentes sensores y sondas. Esta tabla contiene 5 columnas:

- DriverId, clave foránea que referencia la tabla con la lista de los drivers de adquisición Regedate.
- TagId, clave foránea que referencia la tabla con la lista de los tags o variables.

- TS, timestamp. Almacena la fecha y la hora en la que se tomó el dato representado en esa fila.
- Value, valor del dato.
- IsValid, indica si el dato es válido. Se establece a 1 para datos correctos, en caso contrario a 0.

La clave primaria de la tabla, está formada por la combinación de los campos DriverId, TagId y TS.

Internamente, Regedate secciona esta tabla en intervalos mensuales. Cuando un mes termina, deja de escribir registros en la tabla actual. En ese momento, crea una nueva tabla donde guardará los registros del próximo mes. Para evitar las operaciones del tipo JOIN, la aplicación realiza un SELECT sobre cada una de estas tablas y sobre la tabla donde se guardan los nombres de las variables. Con los datos en memoria, Regedate realiza las operaciones necesarias para agruparlos y transponer filas por columnas, dejando de este modo los datos en el formato que los muestra la aplicación.

Tabla rawdata

TS	DriverId	TagId	Value	IsValid
31/12/14 23:58	42	906	1,97	1
31/12/14 23:58	45	464	11534,7	1
31/12/14 23:59	42	906	1,97	1
31/12/14 23:59	43	170	2,2	1
31/12/14 23:59	46	132	6452,50	1
31/12/14 23:59	40	38	0	1
31/12/14 23:59	45	491	1,8	1

+

Tabla Tags

DriverId	TagId	Name
42	906	NIVEL DEPOSITO [Deposito Bugarra]
45	464	Factor_potencia_III_CIRCUTOR_ONTINYEN
43	170	NIVEL DEPOSITO [Deposito Casinos]
46	132	Caudal Pueblo l/min_CAUDAL_ONTINYENT
40	38	02_Valor_Centrifuga_3_Caudal_Lodos
45	491	Tiempo bomba horas_HORAS_BOMBA_CI

Tabla objetivo a mostrar en las vistas

	NIVEL DEPOSIT	Factor_potencia_II	NIVEL DEPOSIT	Caudal Pueblo l/r	02_Valor_Centrifuga_	Tiempo bomba horas_
31/12/14 23:58	1,97	11534,7				
31/12/14 23:59	1,97		2,2	6452,50	0	1,8

El objetivo de esta transposición, es facilitar al usuario la visualización de datos mostrándolos de un modo amigable. Para ello, Regedate realiza consultas select sobre la tabla que contiene los nombres de las variables y sobre las tablas mensuales de datos según el intervalo temporal definido. Estos selects, obtienen los datos para la tabla con el DriverId, el TagId, el nombre de cada una de de las variables y todos sus valores en el intervalo definido. Posteriormente y con los datos en memoria, se procesa el datatable (forma en la que .Net guarda la consulta) resultante para transformar los datos y formatearlos como la tabla objetivo.

Estas operaciones, pese a ser mucho más rápidas en memoria que si las realizase la base de datos, siguen suponiendo un alto coste computacional por el gran número de registros que puede llegar a contener una vista.

Durante el primer periodo de implantación de Regedate, se han dado casos en el que un usuario ha solicitado informes con un total de 400 variables. Estas guardan un valor cada minuto, de modo que la cantidad de registros en un informe anual, asciende a un total de 210.240.000. Para evitar este alto coste computacional, buscaremos un sistema de almacenamiento en el que los valores puedan quedar guardados en el formato utilizado por la aplicación.

En este punto, entran en juego las bases de datos NoSQL del tipo almacén de documentos. Gracias a su flexibilidad estructural, exploraremos las posibilidades para guardar estos registros de modo que tengan un formato amigable para la aplicación y podamos evitar estas transformaciones. Testaremos las diferencias de rendimiento real entre el sistema con una base de datos relacional y otra del tipo NoSQL. Evaluando las ventajas o inconvenientes de cambiar el sistema gestor de bases de datos.

Tecnologías

SQL Server

Gestor de base de datos de Microsoft, basado en el modelo relacional. Utiliza como lenguajes de consulta T-SQL (Transact-SQL) y ANSI SQL. SQL Server, reúne en un solo producto, el motor de base de datos relacionales y las herramientas de gestión. Entre sus principales características destacan:

- Soporte de transacciones.
- Escalabilidad, estabilidad y seguridad.
- Soporte para procedimientos almacenados.
- Entorno gráfico de administración.
- Modo cliente servidor.
- Permite administrar información de otros servidores de datos.

T-SQL

Lenguaje SQL propiedad de Microsoft y Sybase. Desarrollado originalmente por IBM, T-SQL busca expandir el estándar SQL para incluir en este la programación procedural, variables locales y cambios a las sentencias DELETE y UPDATE. Gracias a estos cambios permite definir tareas que no serían posibles sobre SQL.

Couchbase

Servidor de bases de datos NoSQL de alto rendimiento, capaz de escalar en hardware para soportar grandes conjuntos de datos con un alto número de lecturas y escrituras manteniendo baja latencia y alta consistencia.

Tres posibles implementaciones.

- **Base de datos documental**. Acceso a documentos con la ventaja de una caché integrada para alto rendimiento.
- **Clave valor**. Alto rendimiento para lecturas y escrituras mientras mantiene disponibilidad y persistencia.
- **Caché distribuida**. Provee escalabilidad y accesos con baja latencia a altas cantidades de datos.

Características.

- Escalabilidad
- Disponibilidad
- Rendimiento
- Replicación
- Fácil administración mediante API y consola
- Integración Big Data

La arquitectura de Couchbase está centrada en memoria para aumentar la escalabilidad y el rendimiento. Los datos se escriben en memoria principal (RAM) para una baja latencia en lecturas y escrituras. De este modo, evita operaciones innecesarias sobre disco.

Couchbase Server almacena los datos en documentos JSON. Un documento, no es más que un objeto JSON con un número de campos definidos. También es capaz de almacenar datos en pares clave-valor utilizando la clave como lo haría SQL.

En este tipo de bases de datos, no necesitamos establecer relaciones entre tablas como se haría en una base de datos relacional. Cada documento JSON puede tener su propia serie de claves, y cada documento tiene su propio esquema implícito que está representado en la forma de organizar la información. Estas claves no necesitan tener un tipo de datos predefinidos, puede contener diferentes tipos durante el tiempo incluso ser diferentes entre objetos similares.

N1QL

Lenguaje de consultas en Couchbase, permite abstracciones similares a SQL consiguiendo una rápida curva de aprendizaje. Soporta joins, expresiones, subconsultas,

lenguaje de manipulación de datos (DML) y otras características que permiten crear aplicaciones rápidamente.

Al igual que en SQL, las consultas se realizan con la sentencia SELECT pudiendo realizar subconsultas anidadas sobre los distintos documentos JSON que componen la base de datos. Para realizar consultas sobre varios documentos al mismo tiempo, de forma homóloga a SQL, se puede utilizar JOIN. Si se necesita una consulta con condiciones, se dispone de la cláusula WHERE.

N1QL proporciona rutas para acceder a datos anidados. Separando los nombres de los documentos/subdocumentos por puntos se puede acceder a la ubicación lógica de estos. Como ejemplo, la forma de obtener la calle del pedido de un cliente sería “orders.billTo.street”. De este modo obtendríamos el valor de la calle dentro de un determinado documento. Si además un elemento está compuesto por varios arrays, se puede acceder a cada uno de estos con “orders.items[x].productId”, donde “x” es el subdocumento que queremos alcanzar, y productId, el identificador del producto.

También permite utilizar las sentencias GROUP BY, ORDER BY, LIMIT y OFFSET, así como un amplio conjunto de funciones para transformar los resultados según sea necesario.

Map/reduce

Mapreduce es un modelo de computación paralela preparado para trabajar con grandes cantidades de datos. Sirve para abordar procesos que se puedan disgregar en funciones map() y reduce().

map()

Función encargada del mapeo, toma como entrada un par de datos (clave, valor) y devuelve una lista de pares de datos. Esta operación es realizada en paralelo para cada par de datos.

reduce()

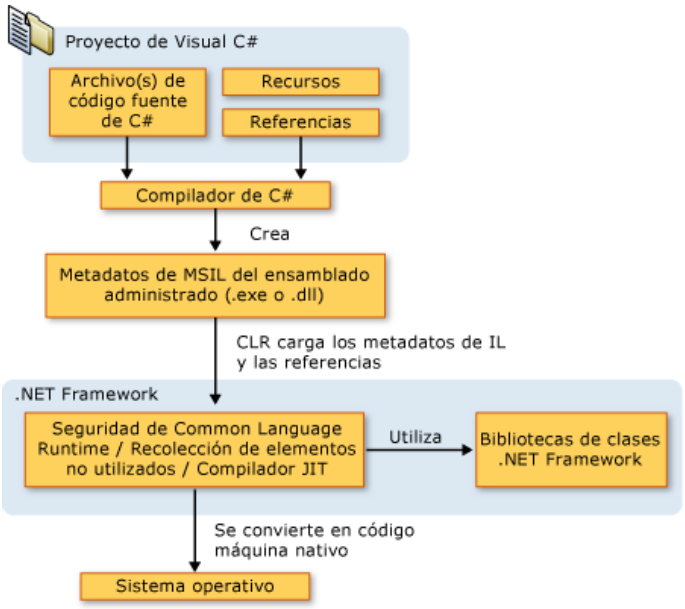
Función que recibe como entrada la lista de pares de claves generados en la función map(). Produce una colección de valores.

.Net C#

Lenguaje orientado a objetos, permite compilar aplicaciones que se ejecutan sobre .Net Framework. Permite crear aplicaciones escritorio para Windows, servicios web XML, componentes distribuidos, aplicaciones cliente-servidor, aplicaciones de bases de datos y más...

Proporciona un editor de código avanzado, diseñadores de interfaz de usuario, depurador integrado y numerosas herramientas más para facilitar el desarrollo de aplicaciones.

Sintaxis basada en símbolos de llave, al estilo C++ o Java. Simplifica las complejidades de C++ y proporciona características eficientes tales como tipos de valor que admiten valores NULL, enumeraciones, delegados, expresiones lambda y acceso directo a memoria. Permite el uso de expresiones Language-Integrated Query (LINQ) que convierten una consulta fuertemente tipada en una construcción del lenguaje.



Como lenguaje orientado a objetos admite los conceptos de encapsulación, herencia y polimorfismo. Todas las variables y métodos incluidos en el método Main se encapsulan dentro de las definiciones de una clase. Una clase puede heredar directamente de una clase primera y además puede implementar cualquier número de interfaces.

C# facilita el desarrollo de componentes de software a través de varias construcciones de lenguajes innovadoras, entre las que se incluyen:

- Firmas de métodos encapsulados denominados delegados, que habilitan notificaciones de eventos con seguridad de tipos.
- Propiedades, que actúan como descriptores de acceso para variables de miembro privadas.
- Atributos, que proporcionan metadatos declarativos sobre tipos en tiempo de ejecución.
- Comentarios en línea de documentación XML.
- Language-Integrated Query (LINQ) que proporciona funcionalidades de consulta integradas en una gran variedad de origines de datos.

Arquitectura .NET Framework

Los programas C# se ejecutan en .NET Frameork un componente que forma parte de Windows e incluye un sistema de ejecución virtual denominado Common Language Runtime (CLR) y un conjunto unificado de bibliotecas de clases.

El código C# se compila en un lenguaje intermedio (IL) conforme con la especificación CLI. Este código intermedio se guarda en disco en un fichero ejecutable. Al iniciar este ejecutable, el código se carga en CLR. Si se cumplen las directivas de seguridad, CLR realiza una compilación Just In Time (JIT) para convertir el código intermedio en instrucciones máquina nativas.

La interoperabilidad del lenguaje es una de las características más importantes de .NET Framework. Al estar compilado como código intermedio que cumple la especificación común de tipos (CTS), puede interactuar con el código generado en las versiones .NET de cualquiera de los más de 20 lenguajes conformes con CTS.

Estudio comparativo almacenes NoSQL.

¿Qué es NoSQL?

Son bases de datos que almacenan información sin una estructura predefinida ni relaciones. Los modelos de almacenamiento más destacados son: clave-valor (similar a una tabla hash), almacenamiento en documentos, grafos y columnas. La elección de un tipo u otro vendrá dado por las necesidades de la aplicación a la hora de almacenar la información.

Este tipo de base de datos, ofrece un alto rendimiento. Están preparadas para escalar de forma horizontal (añadiendo más nodos al sistema). Normalmente, la información utilizada con más frecuencia la almacenan en memoria, dejando el disco para obtener persistencia.

Gracias a estas características, son capaces de ofrecer grandes volúmenes de datos de forma mucho más rápida de la que lo harían los tradicionales modelos relacionales.

Por contra, no suelen soportar operaciones del tipo JOIN ni garantizan las propiedades ACID (atomicidad, consistencia, aislamiento y durabilidad) por completo.

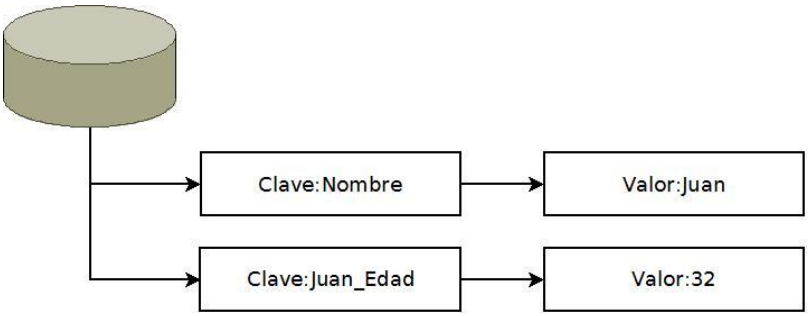
Tipos de almacenes NoSQL

Clave-Valor

Las bases de datos clave-valor funcionan de modo similar a una tabla con dos elementos. Cada elemento está identificado por una clave única y por un valor asociada a esta. La clave, suele estar representada por una cadena de texto, mientras el valor, puede ser un tipo del lenguaje de programación (string, integer, array) o un objeto.

Las bases de datos Clave-Valor presentan un alto rendimiento a escrituras y lecturas. Ofrecen una gran escalabilidad gracias a su facilidad para el particionado, pudiendo dividir la información de acuerdo a la clave.

Entre los primeros ejemplos de este tipo de bases de datos encontramos BigTable de Google y SimpleDB de Amazon. Las bases de datos posteriores suelen estar inspiradas en estas.

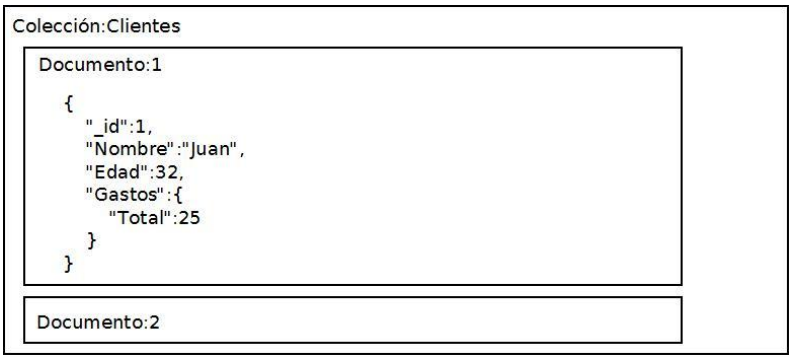


Almacenes de documentos

Las bases de datos orientadas a documentos, almacenan la información como un documento estructurado, generalmente JSON o XML. Cada uno de estos documentos, debe tener una clave única que lo identifique en el contexto de la base de datos. Contienen un conjunto de datos estructurados de modo similar al Clave-Valor, y al mismo tiempo, pueden anidar otros sub documentos. Estos documentos se organizan en colecciones de documentos. Una colección puede considerarse de modo análogo a una tabla en un modelo relacional, mientras que el documento sería una entrada de la tabla.

Dependiendo del tipo de implementación en la aplicación servidor, son capaces de hacer operaciones con los datos. Permitien consultas avanzadas, accesos a datos concretos dentro de un documento o colección. Aun así, son incapaces de realizar JOINS.

Como representantes de este tipo de almacenes de información destacan MongoDB, CouchDB o ArangoDB.

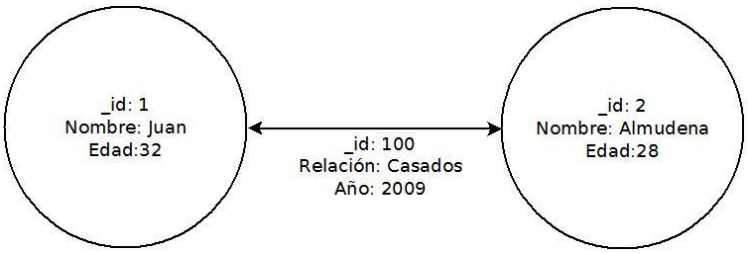


Grafos

Este tipo de bases de datos, guardan la información como grafos donde la importancia recae sobre las relaciones entre nodos. La información se representa como nodos del grafo, y las relaciones vienen dadas por las aristas del mismo. Las relaciones tienen significado direccional y pueden contener atributos, de modo que se crean patrones de búsqueda entre los nodos.

Gracias a esta estructura, la navegación entre relaciones es mucho más rápida que en un modelo relacional.

Entre las principales bases de datos orientadas a grafos cabe destacar Neo4J, Infinite Graph o OrientDB.

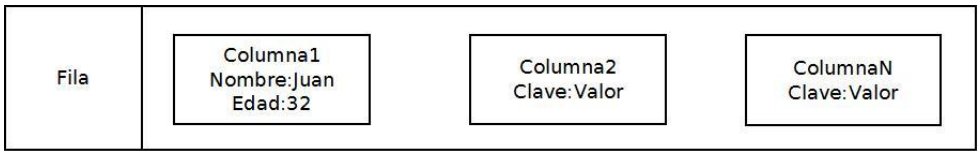


Orientadas a Columnas

Son bases de datos donde los valores se guardan en columnas en lugar de filas. Funcionan almacenando una clave a la que se le asocia una súper columna con toda la información.

Con estos cambios se consigue una alta velocidad de lectura, pero una muy baja eficiencia en escrituras. Son soluciones adecuadas para aplicaciones con muy pocas escrituras. Están pensadas para realizar consultas y agregaciones sobre grandes cantidades de datos.

Entre las bases de datos de este tipo destacan Cassandra y Hbase.



Análisis de bases de datos.

El análisis de las bases de datos consta de ocho puntos. Cada uno de estos puntos aporta 1 punto a la nota final, excepto el benchmark que aporta 2 puntos. Los 2 puntos serán para el motor NoSQL con mejor rendimiento y 0 para el peor. Se debe tener en cuenta, que estas

pruebas se han realizado en local sobre un procesador i5 M520@2'40GHz con 4GB y sistema operativo Windows 7 32bits. Por lo que en otros entornos podrían variar en gran medida.

Este análisis nos ayudará a establecer cual es la base de datos idónea para nuestra aplicación. En ella, vamos a necesitar un almacén de datos con índices de lectura concurrentes muy elevados. También será necesario que se pueda integrar fácilmente en .Net y que ofrezca al programador herramientas para hacer mas cómodo su trabajo. Por ello, vamos a evaluar una serie de parámetros, que finalmente puntuaremos en una tabla de la que podremos deducir cual de estos almacenes es el mejor para nuestro caso.

Veamos a continuación cada uno de los parámetros que hemos decidido evaluar.

- Benchmark, se realizarán pruebas de rendimiento sobre el motor de la base de datos. Estas pruebas consisten en la realización de lecturas (Get) y escrituras (Insert) de arrays de 1.000 , 10.000 y 100.000 elementos (enteros entre 0 y 99). Para las pruebas de lectura, el número de lecturas concurrentes variará entre 1, 10 y 100. Estas operaciones se ejecutan en paralelo, y se mide el tiempo en milisegundos, desde que empieza la primera hasta que acaba la última. Se marcarán en verde los mejores tiempos durante las pruebas, en rojo los peores.
- Interfaz REST, se considerará como punto a favor que la base de datos disponga de una Interfaz web. Facilitando así la comunicación entre esta y la aplicación.
- ACID, el sistema gestor debe ofrecer transacciones atómicas, consistentes, aisladas y durables.
- Integración .Net, al ser este el lenguaje sobre el que se realizará el proyecto final. Buscamos una base de datos con un driver que facilite la ejecución de operaciones en la base de de datos desde la aplicación.
- Interfaz de administración web, es interesante disponer de una interfaz web donde poder revisar el estado de las colecciones/objetos o modificar la configuración de la base de datos.
- Cluster y replicación, parte importante para conseguir escalabilidad y tolerancia a fallos en bases de datos NoSQL.
- Clientes, repasaremos qué empresas utilizan cada una de las bases de datos. Factor a tener en cuenta pues muestra la envergadura del proyecto.
- Licencia, se analizará el tipo de licencia y si tienen algún coste.
- Tipo de almacenamiento NoSQL, de documentos, clave-valor, grafo o columnas.

ArangoDB

- Benchmark:

Número de elementos	1000				10000				100000			
Operación	Insert	Get	Get	Get	Insert	Get	Get	Get	Insert	Get	Get	Get
Operaciones concurrentes	1	1	10	100	1	1	10	100	1	1	10	100
Tiempo 1	31	25	82	561	73	42	218	1697	235	120	693	6268
Tiempo 2	24	24	64	481	80	35	200	1699	224	132	632	6102
Tiempo 3	21	24	76	564	66	34	188	1737	275	132	692	6365
Tiempo 4	20	23	78	491	100	65	187	1704	246	153	727	6263
Media	24	24	75	524	80	44	198	1709	245	134	686	6249

- Interfaz REST: Sí. En el panel de administración incluye una API visual para comprender el funcionamiento de la interfaz. Permite interacción con la base de datos y con la información almacenada.
- ACID: Implementa transacciones ACID cuando se ejecuta en un solo nodo. Con múltiples nodos facilita la elección de dos de las tres propiedades CAP deseadas.
- Integración .Net: Existe un driver desarrollado por la comunidad que ataca a la base de datos a través de la api HTTP REST.
- Tipo: Clave-valor, grafo, almacén de documentos.
- Administración web: Completo portal de administración WEB con gestor de colecciones y documentos, acceso a grafos, shell javascript, logs y API.
- Cluster y replicación: Sharding, configuración mediante fichero .conf y web.
- Clientes: No se han encontrado clientes destacados.
- Licencia: Apache 2.0 license. Gratuito para uso comercial y no comercial.

Couchbase

- Benchmark:

Número de elementos	1000				10000				100000			
Operación	Insert	Get	Get	Get	Insert	Get	Get	Get	Insert	Get	Get	Get
Operaciones concurrentes	1	1	10	100	1	1	10	100	1	1	10	100
Tiempo 1	9	2	43	147	62	5	32	222	118	56	230	1836
Tiempo 2	13	2	54	152	60	2	16	190	84	31	252	1830
Tiempo 3	6	2	51	185	17	4	31	200	93	53	247	1973
Tiempo 4	17	2	55	189	9	2	32	197	96	36	241	1911
Media	11	2	51	168	37	3	28	202	98	44	242	1887

- Interfaz REST: Sí. Base de datos y información.
- ACID: Soporta transacciones ACID a nivel de documento. Cuando existe un cluster de servidores, prioriza la consistencia y la tolerancia a particiones. En un sistema multicluster proporciona disponibilidad y tolerancia a particiones.
- Integración .Net: SDK en versión 2.0, permite trabajar con documentos de forma síncrona o asíncrona, gestionar la base de datos y los clusters.
- Tipo: Clave-valor, almacén de documentos.
- Administración web: Completa administración web, permite configurar nodos en la base de datos, el balanceo, establecer alertas ...
- Cluster y replicación: Soporta clustering y sharding. Balanceo y replicación.
- Clientes: Adobe, Adidas, AccuWeather, BMW, CISCO, eBay, HONDA, Intel, NAVTEQ...
- Licencia: Código Apache 2.0 license. Los binarios tienen dos tipos de licencia, “Community edition”, gratuita. “Enterprise edition” de pago.

CouchDB

- Benchmark:

Número de elementos	1000				10000				100000			
Operación	Insert	Get	Get	Get	Insert	Get	Get	Get	Insert	Get	Get	Get
Operaciones	1	1	10	100	1	1	10	100	1	1	10	100
Tiempo 1	31	28	136	790	72	61	293	2342	553	375	1514	14319
Tiempo 2	62	33	170	964	97	72	286	2846	523	349	1513	14179
Tiempo 3	63	35	102	804	77	69	298	3664	520	325	1611	14098
Tiempo 4	60	34	122	859	59	53	296	2465	532	314	1581	14474
Media	54	32	132	854	76	64	293	2829	532	341	1555	14267

- Interfaz REST: Sí. Configuración base de datos e información.
- ACID: Soporta transacciones ACID a nivel de documento. En cluster prioriza la disponibilidad y la tolerancia a particiones.
- Integración .Net: Drivers creados por la comunidad que atacan a la base de datos a través de la api HTTP.
- Tipo: Documental.
- Administración web: Permite inspeccionar las colecciones de datos, la configuración y la replicación.
- Cluster y replicación: Sharding y replicación en bases de datos remotas.
- Clientes: BBC, CERN...
- Licencia: Apache 2.0 license, gratuito.

MongoDB

- Benchmark:

Número de elementos	1000				10000				100000			
Operación	Insert	Get	Get	Get	Insert	Get	Get	Get	Insert	Get	Get	Get
Operaciones concurrentes	1	1	10	100	1	1	10	100	1	1	10	100
Tiempo 1	78	9	34	349	145	22	130	1019	440	105	860	8494
Tiempo 2	38	4	52	355	141	38	115	995	664	121	820	7788
Tiempo 3	56	5	53	368	77	18	112	1039	361	111	833	8150
Tiempo 4	39	12	36	356	132	21	113	1043	566	116	823	8190
Media	53	7	44	357	123	25	117	1024	507	113	834	8155

- Interfaz REST: Por defecto no, se puede montar una capa por encima de la base de datos que funcione como interfaz REST.
- ACID: Soporta transacciones ACID a nivel de documento. Consistencia eventual entre nodos secundarios, soporte a particiones y disponible. Existe un modo seguro de funcionamiento en el que se comporta consistente y con soporte a particiones.
- Integración .Net: Sí, driver basado en la versión 3.5 del Framework de .Net.
- Tipo: Almacén de documentos.
- Administración web: Por defecto no tiene administración web, existen aplicaciones de terceros como MMS que ofrecen una capa web para la administración de MongoDB.
- Cluster y replicación: Soporta sharding y replicación.
- Clientes: Bosch, Forbes, ADP, The Weather Channel...
- Licencia: GNU AGPL v3.0. De pago para uso comercial.

RavenDB

- Benchmark:

Número de elementos	1000				10000				100000			
Operación	Insert	Get	Get	Get	Insert	Get	Get	Get	Insert	Get	Get	Get
Operaciones	1	1	10	100	1	1	10	100	1	1	10	100
Tiempo 1	143	44	No funciona		278	189	No funciona		514	452	No funciona	
Tiempo 2	190	36			147	164			460	299		
Tiempo 3	76	38			237	99			453	338		
Tiempo 4	115	49			206	97			545	289		
Media	131	42			217	137			493	344		

- Interfaz REST: Sí. Información almacenada en la base de datos.
- ACID: Soporta transacciones ACID a nivel de documento. Con varios nodos, ofrece disponibilidad básica, consistencia eventual y tolerancia a fallos relajada.
- Integración .Net: Driver y más, RavenDB está implementado para trabajar principalmente con .Net.
- Tipo: Almacén de documentos.
- Administración web: Interfaz simple, permite ver las colecciones y poco más.
- Cluster y replicación: Sharding y replicación.
- Clientes: MSNBC, OpenVPN, Reffeed...
- Licencia: Gratuita únicamente para proyectos open source.

Resultados

	REST	ACID	.Net	Tipo	WEB	Replicación	Clientes	Licencia	Bench	Total
ArangoDB	1	1	0,5	1	1	1	0	1	1	7,5
Couchbase	1	1	1	1	1	1	1	0.5	2	9,5
CouchDB	1	1	0,5	0.5	1	1	1	1	0.5	7,5
MongoDB	0	1	1	0,5	0	1	1	0,5	1,5	6,5
RavenDB	1	1	1	0,5	0,5	1	1	0	0	6

A nivel global, Couchbase es la opción que mejor cumple los aspectos analizados. Por contra, tiene la penalización de la licencia de pago para uso comercial. Se puede usar en su versión community (gratuita), donde los fixes y las actualizaciones son un poco mas tardías. Existen otras alternativas recomendadas como ArangoDB, pero el ser un proyecto relativamente nuevo, peca en la inexistencia de grandes clientes o patrocinadores que confirmen un proyecto de envergadura.

MongoDB no dispone de grandes ayudas al desarrollador, pero en las pruebas ha demostrado un buen rendimiento, es un proyecto de gran envergadura. Con soporte de grandes empresas, y altas contribuciones por parte de la comunidad.

Otras opciones, en las que el rendimiento no ha sido muy elevado, son CouchDB y RavenDB. El primero, tiene unas buenas características, pero la integración con .Net no es gran cosa. Por contra, RavenDB, tiene una buena integración pero su tipo de licencia de pago es un gran punto en contra.

Importando datos a Couchbase

En el pasado capitulo, Couchbase resultó ser la base de datos elegida por el alto rendimiento, funcionalidades y la compatibilidad. A continuación, vamos a presentar las diferentes opciones para las estructuras de objetos que almacenará Couchbase. En definitiva, mostraremos cómo guardar la información SQL que gestiona Regedate, en una base de datos NoSQL.

Patrones para modelado de datos en NoSQL

En la implementación de un sistema de base de datos relacional, existen una serie de normas que llevan al desarrollador a un modelo basado en tablas y relaciones. Este modelo, pese a poder ser diseñado de múltiples formas siempre tendrá una estructura similar. Por su esquema flexible, esto no sucede en las bases de datos no relacionales. En éstas, es el desarrollador quien debe buscar un diseño que se adapte a su modelo de negocio y su estructura de datos. Lo interesante de las bases de datos NoSQL, es la multitud de formas en la que se puede plantear esta estructura. Existe un elevado número de posibilidades, desde la búsqueda de un modelo con relaciones, hasta un modelo con documentos anidados que repita información. Estos dos modelos, nos llevan a pensar en nuestro modelo de negocio y de datos. ¿Qué será mejor para nuestra aplicación? Un esquema en el que debamos procesar los datos (relaciones) o uno en el que la cantidad de datos de cada consulta sea elevada (documentos anidados) pero el procesado sea nulo.

A continuación analizaremos unos patrones o recomendaciones, no guías, que nos ayudarán a la hora de definir las diferentes estructuras de base de datos que tiene por objetivo este proyecto final de carrera. Buscaremos una estructura con un buen compromiso entre peso y procesado para poder comparar su rendimiento contra la base de datos SQL original de Regedate.

De forma opuesta a lo que se suele hacer en las bases de datos relacionales, en el caso que nos ocupa, no guiaremos la estructura por el tipo de datos disponibles, si no por los accesos

que se realizan sobre estos datos. Para modelar una base de datos no relacional se requiere un conocimiento de estructuras de datos y algoritmos mucho más profundo del que se pueda requerir en las bases de datos relacionales. Dependiendo de cómo funcione la aplicación planificaremos el acceso a datos.

Para empezar, consideremos un ejemplo en el que diferenciar los dos tipos de bases de datos. Pongamos como caso, un esquema de base de datos relacional con una tabla usuario, relacionada con las tablas direcciones, compras, rol, amigos, etc. Para una base de datos SQL, con una consulta obtenemos todos los datos para un usuario dado en cada una de las tablas nombradas.

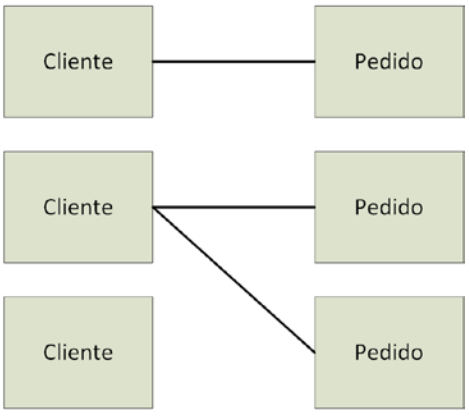
En cambio, en una base de datos NoSQL, en la que no dispongamos de un lenguaje de consultas específico de alto nivel (algunas bases de datos NoSQL ya están empezando a implementar lenguajes de consulta) y sigamos un patrón de diseño similar al de las bases de datos relacionales, consultar estos datos puede llevarnos a tener que lanzar una petición por cada documento (tabla). De este tipo de problemas nace la necesidad de buscar otros patrones para modelar los datos, un ejemplo de ello es la duplicidad de datos. De este modo, teniendo duplicadas dentro de la tabla usuarios cada una de las anteriores tablas se consigue que la velocidad de la base de datos sea muy superior al modelo relacional a costa del alto consumo de espacio.

Técnicas de modelado

Veamos antes de empezar, para contextualizar, dos tipos de relaciones en bases de datos que debemos comprender para entender algunas de las técnicas de modelado.

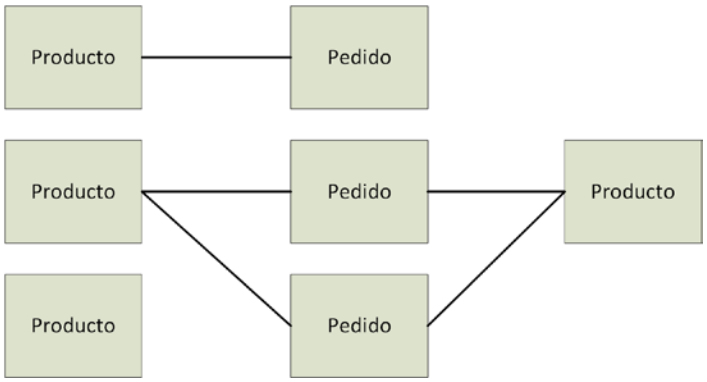
Relaciones “one-to-many”

Esta es una de las relaciones más habituales en las bases de datos, un posible ejemplo es el de un cliente que puede realizar múltiples compras. Se establece una relación de 1 a n elementos.



Relaciones “many-to-many”

Caso en el que se necesitan múltiples instancias en ambos lados de la relación. Ejemplo de un producto que puede estar en múltiples pedidos y donde diferentes pedidos contienen un producto.



Duplicidad de datos o desnormalización

Llamamos desnormalización a la técnica de modelado de datos basada en la duplicidad de estos en múltiples documentos o tablas, para conseguir optimizar el procesado en las consultas. Esta técnica no es específica de las bases de datos no relacionales, se puede usar del mismo modo en las bases de datos SQL. Esta técnica puede ser altamente útil en los siguientes casos.

Cuando la cantidad total de datos es muy superior a los datos obtenidos por cada consulta es interesante duplicar datos que se encuentren en diferentes documentos consiguiendo optimizar el número de consultas y la cantidad de datos por consulta.

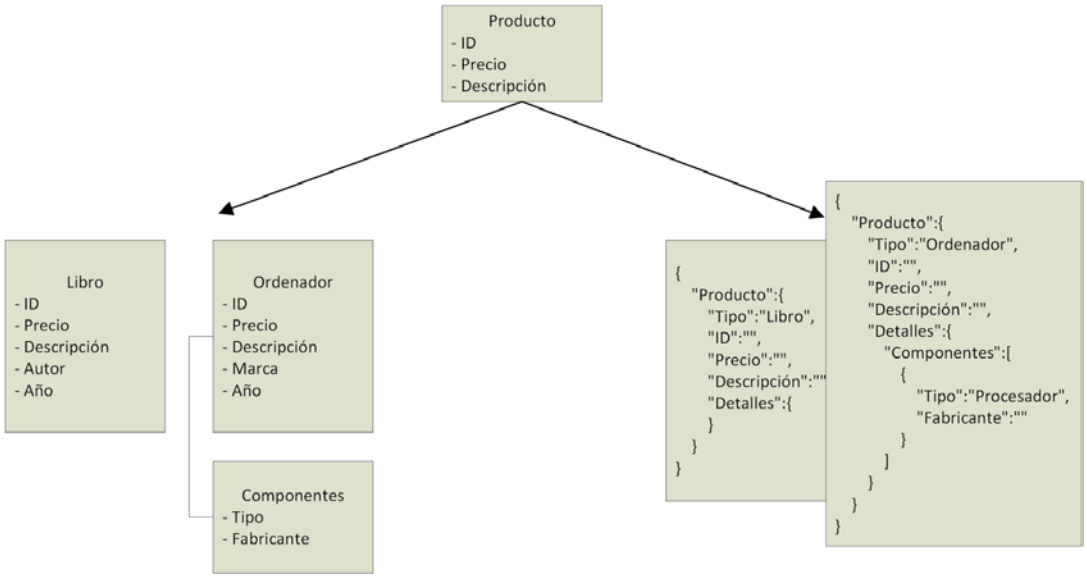
La dificultad del proceso es otro caso en el que resulta altamente interesante la duplicidad de datos. Si consideramos una consulta que accede a un alto número de tablas o documentos, para devolver una cantidad pequeña de datos, ésta puede ser muy costosa temporalmente, por ello de nuevo puede ser interesante, disponer de los datos de más difícil acceso duplicados de modo que las consultas sean más amigables y más fáciles de realizar.

Agregación

Basada en la flexibilidad del esquema NoSQL, permite formar documentos con estructuras anidadas, gracias a ello se minimizan las relaciones entre diferentes tablas o documentos, llevando a realizar menor cantidad de JOINS por la minimización de relaciones “one-to-many”. Además se reducen las diferencias entre el modelo de negocio y las entidades, facilitando la comprensión de la estructura de la base de datos.

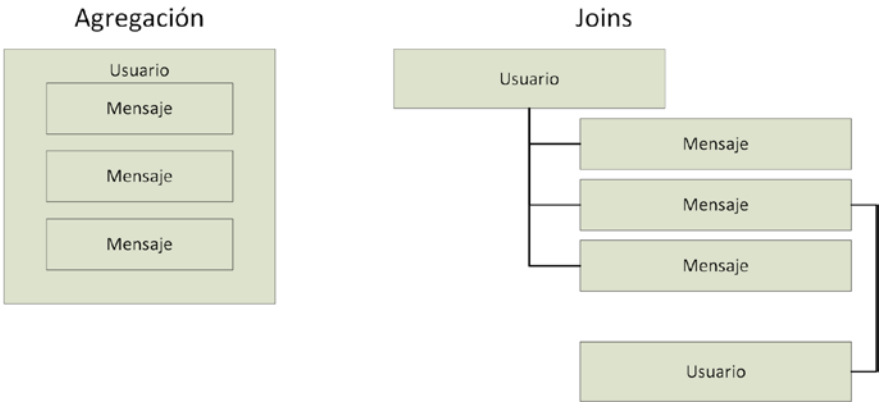
Veamos en el siguiente gráfico cómo utilizar agregación en un ejemplo. Partiendo de una tabla producto formada por un identificador, precio y descripción, puede surgir la necesidad

de guardar tipos específicos de productos como libro u ordenador. Alguno de estos atributos puede además contener relaciones “one-to-many” como en el caso del ordenador al ser un producto formado por varios componentes. Por ello, es posible que estas entidades no sean modeladas de una forma fija y estándar para todos los productos. Gracias al uso de la agregación cada producto puede tener un único documento sobre el que se anidarán subdocumentos consiguiendo adaptarnos al tipo de producto.



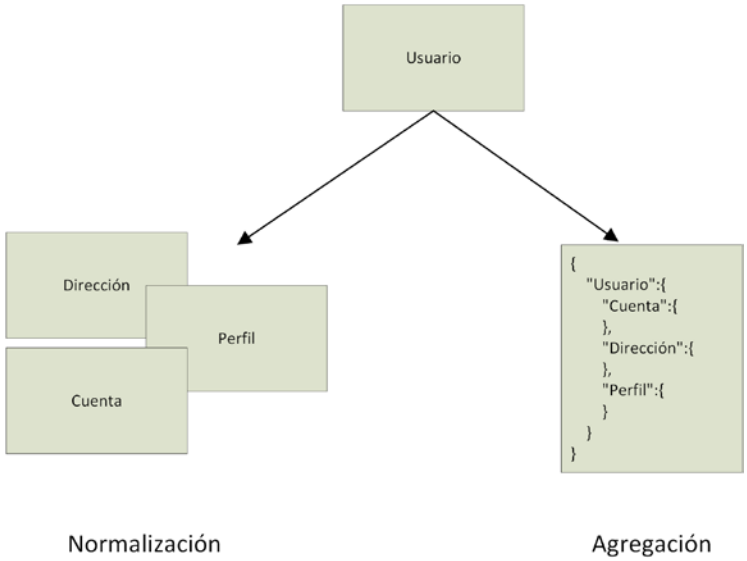
JOINS en la capa de aplicación

Las bases de datos NoSQL, por defecto y salvo algunas excepciones que disponen de un lenguaje de consultas estructurado no soportan la unión de tablas. Como consecuencia, estas uniones deben ser mínimas y en casos aislados, gestionándose en la aplicación que realiza la consulta. Un caso en el que usar uniones son las relaciones “many-to-many”. Como hemos visto anteriormente, utilizando agregación podemos reducir las uniones, ya que cada elemento contendrá todas o gran parte de sus relaciones. Un buen ejemplo para ello es el de compra/producto en un modelo tradicional, cada compra guarda enlaces a cada uno de los productos que se compraron, siguiendo la agregación directamente se guarda el producto.



Agregaciones atómicas

A fin de garantizar las propiedades ACID, la mayoría de sistemas NoSQL limitan el soporte a transacciones. Por ello se utilizan agregaciones, de este modo se limitan las actualizaciones sobre diferentes localizaciones, ya que los datos se guardan a modo de documento único.



Claves numeradas

Pese a los grandes beneficios de un sistema en el que no se usan claves y la complejidad de ordenar claves en entornos distribuidos, puede que ciertos tipos de aplicación necesiten claves ordenadas pese a que el modelo usado no lo proporcione por defecto.

Algunos almacenes NoSQL proveen contadores que permiten la generación de IDs secuenciales. Considerando como caso de ejemplo un servidor de correo electrónico, este puede almacenar los mensajes como IDusuario_IDmensaje, obteniendo así una clave compuesta. De este modo, si se conoce el último IDmensaje es posible desplazarse adelante o atrás dentro de mensajes de un mismo usuario con un coste muy reducido.

Otra ventaja de ello es la posibilidad de agrupar los mensajes en grupos diarios, permitiendo fácilmente desplazarse entre diferentes fechas.

Reducción dimensional

Esta técnica permite transformar un mapa de datos multidimensional en un almacén del tipo no multidimensional. Se emplea tradicionalmente en campos como la geografía donde los sistemas utilizan variaciones de Arboles N-arios para los índices. Estas estructuras, son caras de gestionar y difíciles de manipular cuando el volumen de datos es grande. Una alternativa aproximada es la transformación de la estructura 2D en una lista de entradas.

Índices de tablas

Los índices de tabla son una técnica que ofrece soporte a almacenes que por defecto no disponen de índices internos. Este tipo de índices son especialmente utilizados en almacenes del estilo BigTable. Se pretende crear una tabla especial, con claves que siguen el patrón de acceso. Por ejemplo, una tabla maestra que almacena las cuentas de usuario, la forma de acceder a cada uno de ellos es por ID de usuario. Si una consulta necesita recuperar todos los usuarios de una ciudad específica, puede ser apoyada por una tabla adicional en la que la ciudad es la clave. El problema de este tipo de índices de tablas es que al actualizar la tabla principal se tiene que actualizar la tabla secundaría, lo que puede llevar a problemas de consistencia.

UserID	Info
235	Ciudad: Madrid, email: jhon@doe.com
213	Ciudad: Valencia, email: juan@yahoo.es
254	Ciudad: Madrid, email: pedro@msn.es

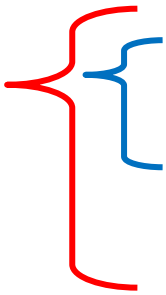
Ciudad	UserIDs
Madrid	235,254
Valencia	213

Claves compuestas

Esta es una técnica muy genérica, pero que puede aportar grandes beneficios cuando queremos utilizar una tabla con claves ordenadas. Una clave compuesta permite definir un índice multidimensional. Como ventajas, poder acceder por claves compuestas permite seleccionar documentos de forma muy rápida por los patrones de cada una de estas tablas.

SELECT Values WHERE provincia="MAD:*"

SELECT Values WHERE ciudad="MAD:Madrid*"



Provincia:Ciudad:Usuario	Valores
MAD:Madrid:235	Valor
MAD:Madrid:254	Valor
MAD:Alcobendas:211	Valor
MAD:Majadahonda:198	Valor
SEV:Sevilla:213	Valor
VAL:Valencia:233	Valor

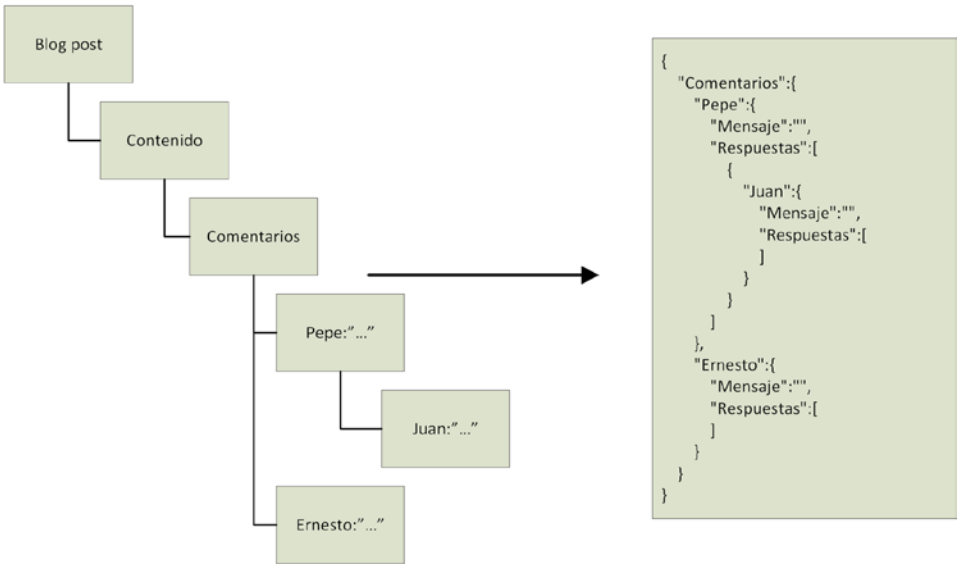
Agregación con claves compuestas

Las claves compuestas, además de usarse para indexar, se pueden utilizar para conseguir diferentes tipos de agrupaciones. Un buen ejemplo de ello, podría ser las veces que un usuario accede a una determinada sección de una tienda online y desde donde ha llegado (muy útil en el comercio online, desde Google, introduciendo directamente la url, anuncios…). Al usar además la agregación almacenaremos cualquier otro atributo en el documento permitiendo un rápido acceso.

UserID:EventID	
123:34512	Addwords
135:14623	Google.com
135:51235	Addwords
135:9844	Bing.com
234:14633	Google.com

Agregación en árbol

Los árboles pueden ser modelados como un simple documento. Esta técnica es muy eficiente cuando el árbol se accede una única vez (Ejemplo, todos los comentarios y respuestas dentro de una noticia de blog). Las búsquedas arbitrarias dentro del árbol pueden suponer un problema, pues habría que recorrerlo entero buscando el elemento. Del mismo modo, las actualizaciones sobre elementos del árbol pueden ser problemáticas, por la dificultad de encontrar el subdocumento correcto dentro de todos los niveles de anidación.

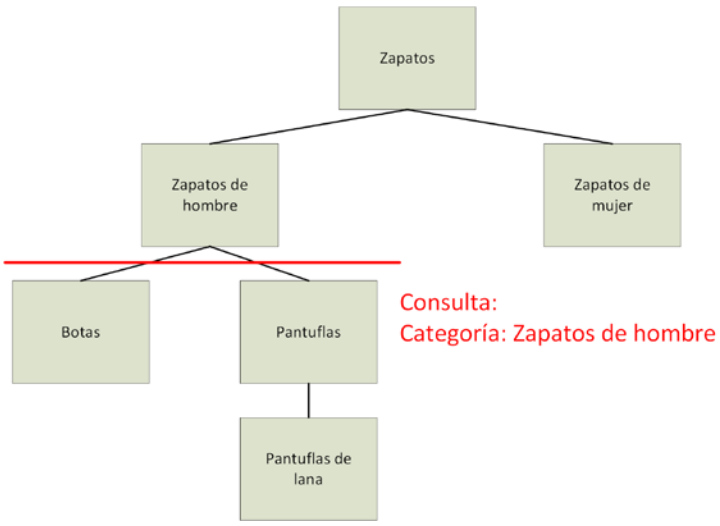


Listas adyacentes

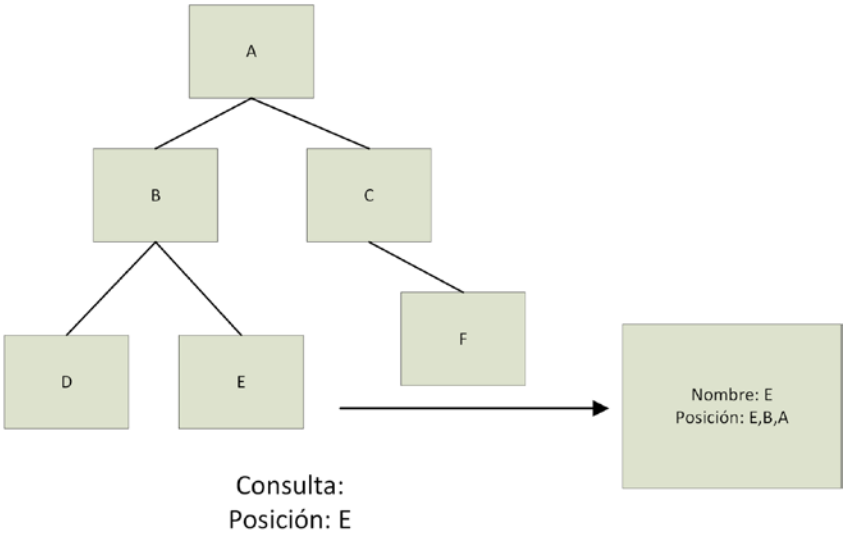
Las listas adyacentes son altamente utilizadas en modelados por grafos, cada nodo se modela como un registro independiente que contiene una lista con sus antecesores y descendientes. Esto permite realizar búsquedas por identificadores de sus padres o sus hijos. Al contrario que en el caso anterior, este es ineficiente para realizar búsquedas que necesiten descargar todo el árbol entero de un nodo dado.

Caminos

Técnica destinada a eliminar la necesidad de recorrer estructuras tipo árbol de forma completa. Como muchos de los modelos anteriormente nombrados, es un tipo más de desnormalziación. Basada en añadir a cada nodo identificadores de sus padres e hijos, haciendo así posible determinar todos los descendientes o predecesores de un nodo sin recorrerlo.

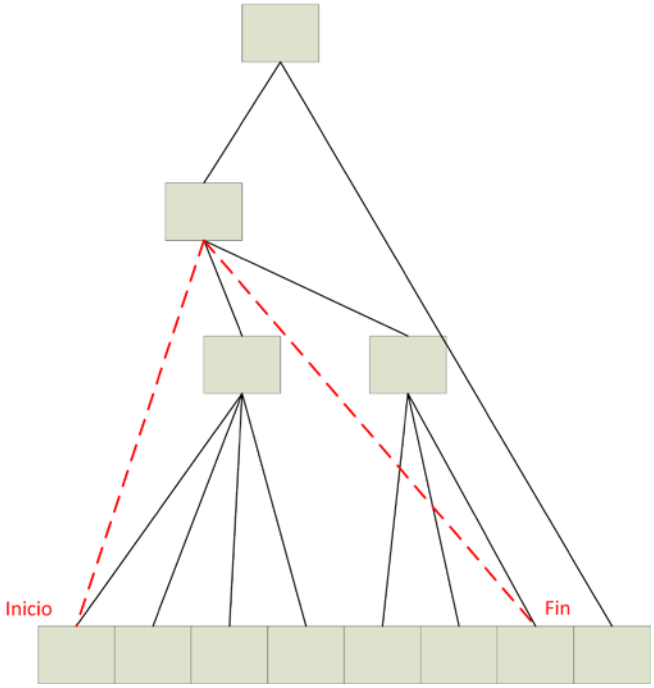


Técnica muy útil en los motores de búsqueda, ya que permite convertir estructuras en documentos planos. De esta forma, con una simple consulta se pueden conocer todos los productos o subcategorías de un documento. Almacenando las rutas como listas o cadenas de texto delimitadas por cierto carácter, podemos con una consulta o expresión sintáctica conocer cuál es el nodo que sigue cierto criterio de localización.



Sets anidados

Técnica estándar para el modelado de estructuras en árbol, ampliamente utilizada en bases de datos relacionales, pero aplicable a almacenes clave valor y a bases de datos documentales. Almacena hojas del árbol en forma de array en cada elemento que no sea hoja utilizando índices de comienzo y de fin.



Muy eficiente para información que no cambia nunca pues permite conocer todas las hojas de un nodo de forma rápida, por el contrario, es ineficiente respecto a inserts y actualizaciones.

Patrones empleados en Regedate NoSQL

Cuando trabajamos con Couchbase, no podemos hablar de base de datos. Cada uno de los almacenes se conoce como bucket. En ellos se almacenarán los documentos que contendrán los datos. En las recomendaciones para desarrolladores que ofrece Couchbase instan en todo momento a usar el mínimo número posible de buckets, el objetivo es siempre empezar con un único bucket. Si las necesidades de la aplicación son altas se pueden utilizar hasta 5, pero nunca se recomienda superar los 10. En el caso de este proyecto, toda la información se almacenará dentro de un solo bucket.

Pese a tener una grandísima cantidad de datos, trabajamos solo con una pequeña parte del sistema de información Regedate. Por ello, dentro del bucket Couchbase se almacenará un único tipo de documento que guardará:

- DriverId: Valor entero con el que se identifica cada Driver.
- TagId: Valor entero con el que se identifica cada Tag dentro de un Driver.
- Name: Nombre del Tag (Ejemplo Contador_salida_rebombeo_VOLHORA_CATADAU_REB)
- TimeStamp: Valor que almacena el instante en el que fue tomado un dato.
- Value: Valor del dato.

Desnormalización

Considerando que el documento anteriormente citado funcionase dentro del actual entorno Regedate, podemos observar que estamos combinando datos de diferentes tablas. Los campos DriverId, TagId y Name se encuentran en las tablas Driver y Tag del esquema original Regedate, mientras que los campos TimeStamp y Value estaban en las tablas mensuales de valores. Obviamente, estas tablas ya contenían los campos DriverId y TagId para utilizarlos cómo referencia a las tablas Driver y Tag. No sucedía lo mismo con el campo Name que solo se encontraba en la tabla Tag y que de ahora en adelante duplicaremos. Con ello estamos primando la agilidad a la hora de obtener los datos contra el pequeño aumento que puede a suponer obtener siempre el nombre de la variable.

```
{
“DriverId”:234,
“TagID”:21,
“Name”:”Nivel deposito”,
“TimeStamp”:201412122300,
“Value”:5.2
}
```

Agregación

Siguiendo el camino de la desnormalización, los campos TimeStamp y Value se añadirán al documento cómo un subdocumento anidado de modo que lograremos almacenar todos los pares de tiempo-valor dentro de este subdocumento.

```
{
“DriverId”:234,
“TagID”:21,
“Name”:”Nivel deposito”,
“Data”:[
                {“TimeStamp”:201412122300,“Value”:5.2},
                {“TimeStamp”:201412122315,“Value”:5.4},
]
}
```

Claves compuestas

Este patrón no se ha utilizado durante el almacenamiento pero sí forma parte de la función map. Gracias al uso de las claves compuestas podremos generar una nueva clave en esta función. Esta nueva clave compuesta constará del DriverId, TagId y TimeStamp. Con ello, se podrán recuperar los datos para un DriverId y un TagId determinados en un periodo. Esto es debido a que Couchbase permite recuperar la información entre una "startkey" y una "endkey" dadas. Estas start y endkey hacen referencia al ID o Clave de cada componente.

```
{
“Name”:”Nivel deposito”,
{"ID":"234:21:201411181120",“Value”:5.2},
{"ID":"234:21:201412122315",“Value”:5.4}
}
```

Emulación de acceso a datos Regedate

Durante el desarrollo del proyecto final de carrera no se tuvo acceso al código fuente de Regedate, por lo que la parte de acceso a datos se tuvo que emular con ayuda de sus desarrolladores. Estos establecieron unas guías sobre cómo la aplicación realizaba el acceso a datos y cómo los procesaba. Para ser capaces de evaluar el rendimiento del sistema SQL y poder comparar con el nuevo sistema NoSQL se implementó una versión reducida de la capa de acceso a datos sobre .Net C#.

Como ya se ha comentado en anteriores capítulos, las tablas donde Regedate almacena los datos tienen una duración mensual. Cuando el mes termina se crea una nueva tabla y todos los datos se empiezan a guardar en ella. Esto es muy cómodo a la hora de realizar operaciones escritura, pero dificulta las opciones de lectura. Pongamos un ejemplo, si la aplicación necesitase acceder a los datos de una variable durante un periodo dos meses, sería tan sencillo como realizar una unión entre ambas bases de datos y lanzar la consulta para esa variable. El tiempo estimado de procesado para obtener los registros solicitados será relativamente bajo. Ahora bien ¿qué puede suceder en un futuro, si necesitásemos obtener todos los valores de varios años para 50 variables entre las centenas de millones de registros que puedan existir? Obviamente el coste computacional sería elevadísimo. Más aún, si para acceder a los datos de cada una de estas tablas mensuales, la aplicación tratase de hacer uniones entre ellas.

La primera solución que Regedate trata para evitar este tipo de problemas, es la generación automática de datos estadísticos. Esta buena práctica, genera cada cierto tiempo agrupaciones (media, suma, máximo…) para cada variable, de modo que si se necesita una petición para un intervalo de tiempo muy largo, se puede hacer sobre estos datos. Si aun así existe la necesidad de lanzar la consulta sobre el grueso de datos brutos, la aplicación administrará el acceso a estos de modo que sea lo más rápido posible.

Debido a la forma en que se almacenan los datos, Regedate necesitaría también que el sistema gestor de la base de datos, realizase una transposición entre filas y columnas. Al ser esta una operación muy costosa, se realizará en la capa de aplicación. Veamos a continuación un

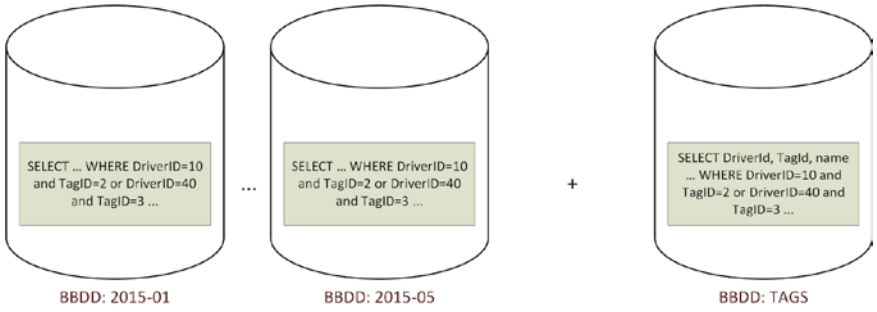
ejemplo del resultado de una consulta mediante la unión de las tablas y la transposición que se realizará sobre estos datos.

TimeStamp	Valor	Nombre variable
20141118112600	16.4	Contador_salida_rebombeo_VOLHORA_CATADAU_REB
20141118112600	103.4	Contador_salida_VOLHORA_ONTINYENT_DTORRATER
20141118112600	104.5	Contador_salida_rebombeo_VOLDIA_CATADAU_REB

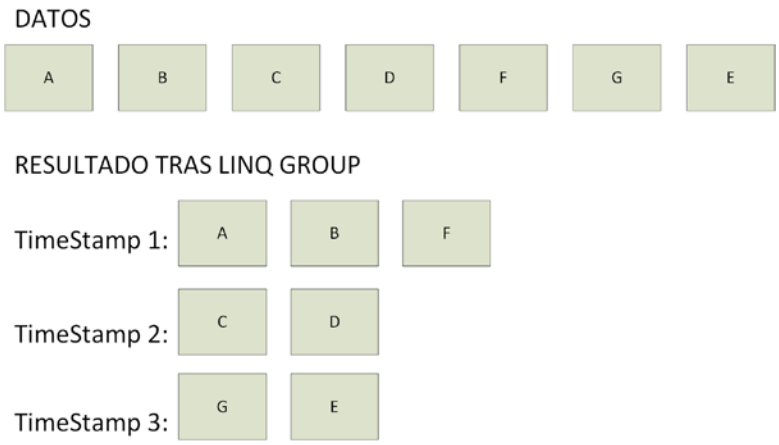
TimeStamp	Contador_salida rebombeo_VOLHORA_CATADAU_REB	Contador_salida_VOLHORA_ONTINYENT_DTORRATER	Contador_salida_rebombeo_VOLDIA_CATADAU_REB
20141118112600	16.4	103.4	104.5

Durante el proceso de lectura, siempre se evitarán a toda costa los JOINS. Como hemos visto en el capítulo anterior, una buena forma de gestionar este tipo de acceso a datos es evitar las uniones entre tablas durante las consultas y gestionarlas en la parte de la aplicación. Este es el patrón que sigue Regedate, con lo que las uniones necesarias se realizan sobre los datos ya procesados en Memoria. De este modo los accesos a datos son mucho más rápidos. En resumen, lo que hace Regedate es una operación SELECT para todas las variables seleccionadas sobre cada una de las tablas implicadas en la consulta.

Con las tablas de datos brutos en memoria, se realiza una nueva consulta que nos devolverá el nombre de cada variable. Estos nombres de variables se encuentran en una tabla aparte, utilizada únicamente para almacenar cada uno de los nombres junto sud TagID y DriverID. Estos dos IDs juntos, forman la clave primaria.



A continuación, se prepara la transposición de filas a columnas, como ya vimos en el capítulo 1 necesitamos conseguir una tabla donde el eje de la Y sean los tiempos, y el eje de las X los nombres de cada una de las variables. El primera paso a realizar con el de conseguir este objetivo es agrupar todos los valores con la misma marca de tiempo (timestamp). Para ello se concatenan todas las tablas de datos brutos creando una nueva supertabla. Sobre ella, se aplica una función LINQ propia de .Net que es capaz de agrupar de una forma muy rápida grandes cantidades de datos.



Con esta agrupación concluida tendremos unas estructuras de tipo grupo, donde el índice será el tiempo y los valores serán cada uno de los valores que se han registrado en ese tiempo. Ahora solo nos faltará montar la nueva tabla. Para ello, se van creando columnas que tienen como nombre cada uno de los nombres de las variables. Finalmente con la ayuda de un diccionario auxiliar previamente definido, en un bucle creamos arrays por cada grupo de datos. Estos arrays tiene como longitud el total de columnas necesarias y, siguiendo el diccionario, lo rellenaremos de modo que cada valor caiga en la componente X,Y que le corresponde. Para finalizar, se añaden estos arrays como filas de la tabla y damos el proceso por concluido.

Resultados

En el presenta capítulo realizaremos una comparación entre los dos sistemas vistos hasta el momento y dictaremos en qué circunstancias es mejor el rendimiento de cada uno de ellos.

Veamos ahora la diferencia medida en tiempo (milisegundos) al consultar datos entre bases de datos SQL y Couchbase. Para las pruebas se ha utilizado un equipo i7-2670QM a 2'20ghz con 16gb de ram y un raid 0 de dos discos duros. Para estas pruebas realizaremos consultas sobre el total de los datos que se nos cedieron a modo de dump (copia a fichero) de las tablas. Estos datos son el equivalente a las tablas de dos meses; la primera de ellas para noviembre de 2014 y la segunda para diciembre de 2014. El sistema gestor de bases de datos SQL funciona sobre MySQL, con una estructura como la ya comentada en capítulos anteriores, donde cada tabla representa un mes diferente y una tabla extra almacena la relación entre nombres, driverId y tagId.

Las pruebas se dividirán en dos bloques principales,

- Bloque de pruebas SQL. Dividido a su vez en dos sub-bloques.
    - Selección de todos los datos de 1 mes dentro de una tabla mensual.
    - Selección de todos los datos de 2 meses divididos en dos tablas mensuales.
- Bloque de pruebas NoSQL. En este caso, por la nueva estructura definida de Couchbase las pruebas siempre se realizarán contra un único documento. Para estar en igualdad de condiciones a las pruebas SQL, existirán también dos sub-bloques.
    - En el primero se seleccionarán todos los elementos para un mes dentro del documento.
    - En el segundo se seleccionarán los elementos de dos meses.

Para la selección de elementos obtendremos todos los que se encuentren en la tabla mensual. Ésta es la opción escogida porque cuando se ejecute una consulta con altos requisitos de computación se lanzará contra un periodo de tiempo largo (ej, 1 año), con lo que la mayoría de consultas sobre cada tabla serán solicitando todos los datos de ella. Las dos únicas tablas en las que no sucederá esto son la primera y la última, donde el sistema gestor de base de datos cortará entre el primer día de la selección y el último del primer mes y el primer día del último mes y el ultimo día de la selección respectivamente.

Tabla SQL (MySQL)

Consulta 1 mes de datos

Veamos las consultas SQL para un periodo de tiempo de 1 mes. En este caso contaremos el tiempo para realizar la consulta de los n nombres de variable (de 1 a 50) a la tabla tags y de igual modo la consulta para recuperar los valores de las n variables.

Numero de variables	Periodo de tiempo (cantidad de datos)	Tablas	Tiempo
1	1 mes	1 + tags	992 ms
5	1 mes	1 + tags	2084 ms
10	1 mes	1 + tags	3376 ms
20	1 mes	1 + tags	6639 ms
50	1 mes	1 + tags	8301 ms

En esta tabla podemos observar como los resultados evolucionan de forma logarítmica, van creciendo a un ritmo lógico según la cantidad de datos. Cuando el número de variables es n veces mayor, el tiempo no se multiplica por n, aumenta bastante menos.

Consulta 2 meses de datos

Veamos las consultas SQL para un periodo de tiempo de 2 meses. Si comparamos esta tabla con la anterior los tiempos son algo más del doble. Este aumento de tiempo es obvio, ya que la cantidad de datos pasa de 1 mes a 2. Aun así mientras más va aumentando el número de variables, más se va alejando el tiempo del doble. El comportamiento es similar a la tabla anterior, pero con una penalización de tiempo por hacer dos consultas más la consulta a tags.

Como ya comentamos en episodios anteriores, para obtener los valores de n meses necesitamos n consultas a la base de datos. Al realizar n consultas somos capaces de evitar un JOIN que aumentaría en gran medida el tiempo de cada consulta. Pero como podemos comprobar en el gráfico actual se produce una pequeña penalización pudiendo definir la función del tiempo como "t=NumeroTablas*t1 + var", donde t1 es el tiempo de consultar una tabla y var es una variable de tiempo que aumenta de modo proporcional al número de variables.

Número de variables	Periodo de tiempo (cantidad de datos)	Tablas	Tiempo
1	2 meses	2 + tags	1220 ms
5	2 meses	2 + tags	4229 ms
10	2 meses	2 + tags	8057 ms
20	2 meses	2 + tags	15118 ms
50	2 meses	2 + tags	19656 ms

Tabla NoSQL (Couchbase)

Consulta 1 mes de datos

La siguiente tabla representa el tiempo necesario para obtener 1 mes de datos en un documento Couchbase. Los tiempos son muy similares a los de obtener las variables para 1 mes en SQL. En este caso, la consulta es mucho más fácil de gestionar al no tener que realizar otra segunda consulta sobre la tabla tags.

Número de variables	Periodo de tiempo (cantidad de datos)	documentos	Tiempo
1	1 mes	1	858 ms
5	1 mes	1	1896 ms
10	1 mes	1	3875 ms
20	1 mes	1	6542 ms
50	1 mes	1	8215 ms

Consulta dos meses de datos

Éste es el caso donde se empieza a notar la mejoría por parte de Couchbase, una de las principales razones es el uso de un único documento para almacenar la información. En la tabla a continuación se puede observar como el tiempo para obtener el doble de información respecto

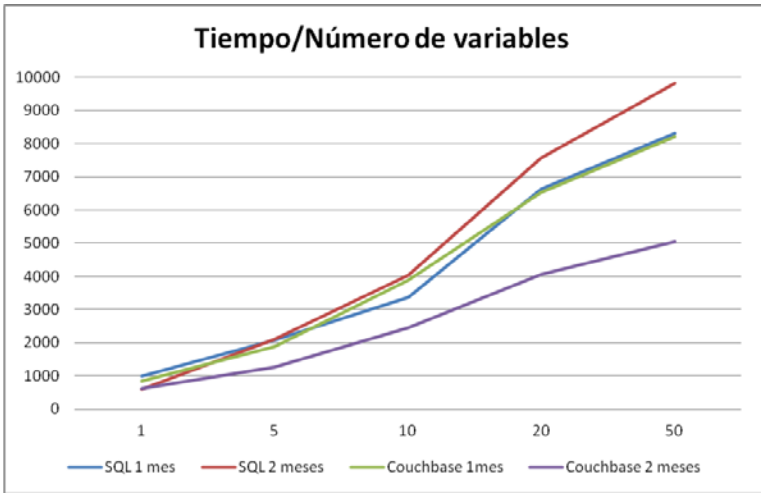
a la consulta anterior apenas aumenta en un 15%. Esto significa que el diseño de tablas utilizado en Couchbase va a dotar a la aplicación de una velocidad de acceso a datos significativa.

Número de variables	Periodo de tiempo (cantidad de datos)	documentos	Tiempo
1	2 meses	1	1254 ms
5	2 meses	1	2541 ms
10	2 meses	1	4895 ms
20	2 meses	1	8133 ms
50	2 meses	1	10087 ms

Gráfico comparativo

El siguiente gráfico muestra los valores normalizados a 1 mes para las anteriores cuatro consultas. Con el término normalizado se está haciendo referencia a que los tiempos obtenidos en las tablas bimensuales han sido divididos por dos. Esta normalización se ha podido llevar a cabo por que el número de elementos dentro de una variable es prácticamente idéntico para los dos meses, ya que todas las variables seleccionadas recibían un dato cada 15 minutos.

Con esta normalización podremos realizar una rápida comparación entre las diferentes opciones. De este modo, de un simple vistazo a la tabla podemos comprender que a mayor número de meses la consulta SQL empeorará su rendimiento, mientras que Couchbase lo mejora. Si la consulta se realiza dentro de un mismo mes, el rendimiento de ambos sistemas es muy parejo.



Conclusiones

Durante este proyecto final de carrera hemos comparado los distintos puntos a valorar en un sistema de bases de datos. El objetivo de éste, era encontrar un sistema gestor con el que mejorar el rendimiento del actual sistema SQL. En el capítulo 4 comparamos los mejores gestores NoSQL del mercado para buscar el que se adaptase a nuestros requisitos de forma más conveniente. En posteriores capítulos se ha analizado el formato de datos en el sistema NoSQL y el acceso a estos.

En estos capítulos, hemos buscado en todo momento la forma más rápida de guardar y acceder a los datos. Con este objetivo se trazó un plan para comparar diferentes sistemas gestores en la mayor igualdad de condiciones. Como resultado de este plan hemos podido deducir que las bases de datos NoSQL, con Couchbase como representante en este proyecto son idóneas para este tipo de almacenes. Los resultados sacan a relucir las carencias de los sistemas gestores SQL a la hora de gestionar múltiples tablas, mientras que ensalzan las características “bigdata” de Couchbase. Los sistemas NoSQL se han mostrado mucho más eficientes a la hora de gestionar grandes cantidades de datos. Mientras más grande era el número de variables y el periodo de selección de datos, mejor rendimiento ofrecían respecto a sus homólogos SQL.

Existen varios aspectos que por alcance no se han podido contemplar en este trabajo y que podrían ofrecen una visión más amplia y real a la hora de comparar los diferentes sistemas.

Como sabemos, en un entorno de producción actual para un proyecto de relativa envergadura nunca habrá un único servidor de bases de datos. Por ello, en proyectos como podría haber sido Regedate, pueden existir varios servidores, o incluso se pueden encender y apagar servidores en proporción de la demanda al sistema. De este modo, las peticiones no se gestionan en un único servidor como hemos visto en este proyecto, se gestionan sobre múltiples servidores. Al repartir el trabajo sobre estos diferentes servidores no solo se consigue agilizar los picos de trabajo. Se mejorará enormemente la obtención de datos. Y es aquí donde haremos especial hincapié en la funcionalidad map/reduce, ya que esta está expresamente preparada para este tipo de entornos y es capaz de multiplicar el rendimiento. Obviamente sobre SQL también existe la posibilidad de dividir el servidor en múltiples sub-servidores, pero estos de ninguna forma serán tan flexibles y rápidos como los servidores NoSQL.

Por los motivos citados y por la alta tasa de transmisión de datos, en este proyecto se ha considerado a los sistemas NoSQL y en concreto a Couchbase como mejor posicionados para este tipo de aplicaciones donde se manejen un alto número de datos con pocas relaciones.

Referencias

1. Bases de datos NoSQL. Elige la opción que mejor se adapte a tus necesidades. Disponible en: http://www.genbetadev.com/bases-de-datos/bases-de-datos-nosql-elige-la-opcion-que-mejor-se-adapte-a-tus-necesidades
2. NoSQL - Wikipedia, la enciclopedia libre. Diponible en: http://es.wikipedia.org/wiki/NoSQL
3. NoSQL Databases: An Overview | ThoughtWorks. Disponible en: http://www.thoughtworks.com/insights/blog/nosql-databases-overview
4. NoSQL Data Modeling Techniques. Diponible en: https://highlyscalable.wordpress.com/2012/03/01/nosql-data-modeling-techniques/
5. Apache CouchDB. Disponible en: http://couchdb.apache.org/
6. ArangoDB - the multi-model NoSQL DB. disponible en: https://www.arangodb.com/
7. MongoDB for GIANT Ideas | MongoDB. Disponible en: http://www.mongodb.org/
8. RavenDB - the open source NoSQL database for .NET. Disponible en: http://ravendb.net/
9. Couchbase. Disponible en: http://www.couchbase.com
10. couchbase community edition on more than 2 nodes - Stack Overflow. Disponible en: http://stackoverflow.com/questions/12685801/couchbase-community-edition-on-more-than-2-nodes/12693619#12693619
11. Couchbase 101 Q & A. Disponible en: http://blog.couchbase.com/couchbase-101-q-and-a