Universidad de Valladolid

E.T.S Ingeniería Informática

TRABAJO FIN DE GRADO

Grado en Ingeniería Informática de Sistemas

Administración en MongoDB

Autor:
Dña. Natalia Román Seneque

Tutora:
Dña. Carmen Hernández Díez

Dedicado a
mi familia,
especialmente
a mi abuelo

Resumen

MongoDB es uno de los principales sistemas de bases de datos no relacionales que encontramos en la actualidad, motivo por el cual se convierte en el protagonista de nuestra investigación.

El principal objetivo de este proyecto es el de conocer más en profundidad la administración de este tipo de bases de datos, conocer su comportamiento, y también su rendimiento cuando los utilizamos en diversos entornos. Todo ello se llevará a cabo a través de la realización de un conjunto de pruebas cuyos resultados serán utilizados posteriormente en la docencia, concretamente en la asignatura "Administración de bases de datos", de esta manera, el alumno tomará contacto también con sistemas no relacionales.

Nos centraremos aquí, por lo tanto, en escenarios y operaciones comunes que podemos encontrarnos cuando utilizamos este tipo de bases de datos, todo ello enfocado con sencillez y de manera concisa, por lo que es posible que eche en falta una mayor profundización a la hora de realizar pruebas o un mayor nivel de detalle en las realizadas.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción y Objetivos

Se busca a lo largo de este primer capítulo presentar una imagen general del proyecto, de manera qe el lector llegue a comprender los principales objetivos que se persiguen con su desarrollo. Junto con ello, se presentará el método que se ha utilizado para llevar a cabo el proyecto, así como una pequeña planificación temporal y las herramientas necesarias para finalizarlo con éxito. Además se presenta también una pequeña evaluación de posibles riesgos que pueden aparecer y a los que nos podemos enfrentar a lo largo de toda la vida de dicho proyecto y el plan de acción frente a cada uno de ellos.

1.1. Introducción y objetivos

Hemos de darnos cuenta de que el mundo actual gira entorno a millones de datos almacenados en diversos lugares del planeta. La expansión de Internet, la expansión de las redes sociales, el auge de la computación en la nube o la implantación de las tecnologías de la información en las empresas, son ejemplos de hechos que nos han llevado a crear lo que podemos llamar la ”revolución de los datos”.

Se ha de tener en cuenta que no basta solo con almacenar todos estos datos, sino que también es necesario llevar a cabo un correcto mantenimiento, un correcto tratamiento y una correcta adaptación para su posterior utilización, sin olvidarnos, por supuesto, de que es necesario dotarlos de la debida seguridad para mantener su integridad y confidencialidad. Nos podemos dar cuenta con todo ello, de la importancia que reside en las bases de datos y la importancia de elegir correctamente sus características a la hora de implantarla en un sistema. Pero, por supuesto, toda esa revolución ha hecho que también las necesidades hayan ido cambiando. Centrándonos tan solo en datos de Internet, en el año 1997 su tamaño era de aproximadamente **88 TB**, en la actualidad es difícil dar una cantidad aproximada, aunque se puede asegurar que su extensión supera el exabyte, esta gran expansión nos hace ver la cantidad de datos que hay por almacenar en el mundo. Junto con ello, debido a la gran importancia que ha adquirido la nube, es de vital importancia tener tiempos de respuesta muy pequeños para que todo funcione correctamente, teniendo en cuenta también que los datos en ella están descentralizados. Por supuesto, existen muchos más motivos, pero estos son algunos ejemplos que han hecho que sea necesario que las bases de datos empleadas para realizar esas tareas también evolucionaran, satisfaciendo así las necesidades de los usuarios. Surge de esta evolución y de estas nuevas necesidades, el objetivo de nuestro estudio, las bases de datos NoSQL *Not Only SQL*, las cuales llegan como un posible sustituto o alternativa a las bases de datos relacionales.

Es esa actualidad y esa importancia que han ido adquiriendo bases de datos NoSQL lo que nos lleva a la creación de esta investigación. El principal objetivo del desarrollo de este proyecto es el de familiarizarnos con las bases de datos de tipo NoSQL, concretamente con MongoDB, en la cual nos centraremos más detalladamente en capítulos posteriores, ya que el conocimiento previo es de tan solo bases de datos relaciones, principalmende de MySQL. Concretamente, se busca centrarnos en la parte de administración a través de la realización

de pruebas relacionadas con ella, las cuales nos ayudarán a comprender su funcionamiento y su comportamiento ante diversas situaciones. Todas estas pruebas aparecerán detalladas, junto con sus conclusiones correspondientes, en el Capítulo 4.

Junto con todo ello, y previamente, en el Capítulo 2 se busca conocer y estudiar más en profundidad las características de las bases de datos NoSQL en general, y los diversos tipos que existen, así, de esta manera nos situaremos en el contexto y comprenderemos mejor con qué estamos trabajando, de manera que dicho estudio nos guíe y ayude a la hora de crear y diseñar las pruebas anteriormente mencionadas.

Todos estos objetivos buscan conseguirse desde un punto de vista docente, por lo que las pruebas que se realizarán estarán enfocadas a conocer el comportamiento de las operaciones más comunes que se pueden realizar en las aulas.

1.2. Planificación

Se detalla a continuación la planificación temporal que se ha seguido para elaboración de este proyecto, así como los recursos que han sido necesarios para llevar a cabo cada fase.

Nombre de la tarea	Descripción	Duración de la tarea	Fecha de inicio	Fecha de fin
Plan de desarrollo del proyecto		44 días	lun 11/11/13	mar 24/12/13
Determinar el alcance del proyecto	Descripción de los propósitos y objetivos que se pretenden alcanzar con el desarrollo del proyecto, así como sus límites	4 días	lun 11/11/13	jue 14/11/13
Planificación del proyecto		8 días	vie 15/11/13	vie 22/11/13
Identificación de tareas	Identificación y descripción de cada una de las tareas necesarias para llevar a cabo el proyecto	3 días	vie 15/11/13	sáb 17/11/13
Determinar la duración de las tareas	Determinación de las relaciones entre las diferentes tareas	3 días	lun 18/11/13	mié 20/11/13

Sigue en la página siguiente.

Nombre de la tarea	Descripción	Duración de la tarea	Fecha de inicio	Fecha de fin
Identificación de los recursos	Identificación de los recursos necesarios para llevar a cabo el proyecto	2 días	vie 15/11/13	sáb 16/11/13
Afianzar recursos	Establecer los recursos que necesita cada actividad	2 días	jue 21/11/13	vie 22/11/13
Identificación y análisis de riesgos	Identificación y descripción de los riesgos a los que nos podemos enfrentar y descripción de la manera de solventarles	1 sem	sáb 23/11/13	mié 27/1/13
Documentación "Introducción y objetivos"	Elaboración de la documentación referente al plan de desarrollo del proyecto	25 días	jue 28/11/13	dom 22/12/13
Revisión del plan de desarrollo	Supervisión y aprobación del plan de desarrollo del proyecto	2 días	lun 23/12/13	mar 24/12/13
Entregable	Fecha límite para la elaboración del plan de desarrollo	0 días	mar 24/12/13	mar 24/12/13

Sigue en la página siguiente.

Nombre de la tarea	Descripción	Duración de la tarea	Fecha de inicio	Fecha de fin
Búsqueda de información NoSQL y MongoDB	Búsqueda de información relacionada tanto con bases de datos NoSQL con MongoDB	30 días	lun 17/02/14	mar 18/03/14
Adquisición e instalación de las herramientas	Adquisión de herramientas como Latex, MySQL o MongoDB	7 días	mié 19/03/14	mar 25/03/14
Síntesis y redacción de la información	Redacción de la documentación utilizando la información adquirida anteriormente	20 días	mié 26/03/14	lun 14/04/14
Determinación de las pruebas MySQL	Diseño de las pruebas que se van a realizar para comparar MySQL con MongoDB a partir de la información recolectada	7 días	mar 15/04/14	lun 21/04/14
Determinación de las pruebas MongoDB	Diseño de las pruebas que se van a realizar con MongoDB en base a la información recolectada	15 días	mar 22/04/14	mar 06/05/14

Sigue en la página siguiente.

Nombre de la tarea	Descripción	Duración de la tarea	Fecha de inicio	Fecha de fin
Realización y documentación de las pruebas de MySQL y MongoDB	Realizar y documentar las pruebas entre las dos bases de datos con sus conclusiones	7 días	mié 07/05/14	mar 13/05/14
Preparación del entorno de MongoDB	Preparar y configurar el entorno donde se van a realizar las pruebas de MongoDB	7 días	mié 14/05/14	mar 20/05/14
Realización y documentación de las pruebas de MongoDB	Realizar y documentar las pruebas de MongoDB con sus respectivas conclusiones	30 días	mié 21/05/14	jue 19/06/14
Síntesis de la documentación	Formato de la documentación y síntesis de toda ella	7 días	vie 20/06/14	jue 26/06/14
Revisión del proyecto		3 días	vie 27/06/14	dom 29/06/14
Entrega del proyecto		0 días	lun 30/03/14	lun 30/03/14

Tabla 1.1: Planificación temporal

Se muestra también el diagrama de Gantt correspondiente a dicha planificación.

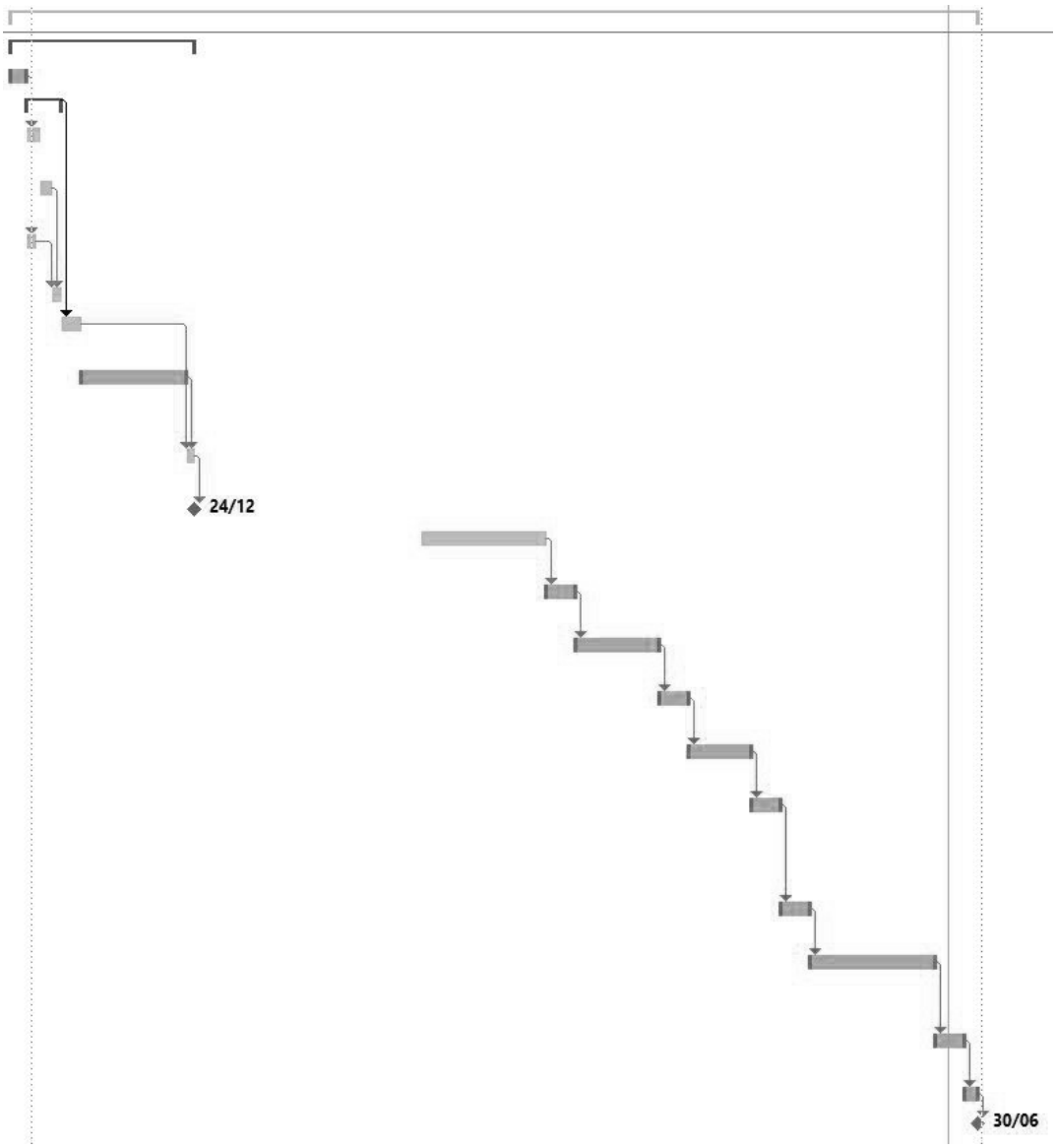

Figura 1.1: Diagrama de Gantt

1.3. Herramientas utilizadas

Para llevar a cabo este proyecto con éxito ha sido necesaria la utilización de una máquina Linux, concretamente con un sistema operativo Ubuntu 14.04 LTS que disponga de base de datos MongoDB, la versión de MongoDB que se ha utilizado ha sido la versión 2.6.1, cuya versión de shell es 2.4.9, y que disponga también de MySQL, concretamente ha sido utilizada la versión 5.1. La máquina dispone de un procesador Intel Core i7-3630QM a 2.40GHz, 8GB de memoria RAM y el procesador es de tipo x64.

En la elaboración de la planificación anterior se ha utilizado la herramienta de Microsoft Project 2013, la cual permite realizar un mejor seguimiento de la evolución temporal del proyecto, de manera que se podrá evaluar si hay algún problema en ella y si aparece algún riesgo relacionado con la planificación temporal.

Ha sido necesaria también la utilización de una herramienta capaz de crear los suficientes datos de prueba tanto para MySQL como para MongoDB. Para MySQL se ha utilizado **generateData**, la cual está disponible tanto como herramienta web como en versión de escritorio. Esta última ha sido la utilizada, ya que se necesitaban más de cien registros (el máximo en la versión gratuita online).

Por último, la realización de este documento se ha realizado utilizando la herramienta Texmaker 4.1 en un entorno Windows, la cual utiliza PDFLaTeX para la compilación del mismo.

1.4. Gestión de riesgos

La planificación presentada en la sección 1.2 es la que se intentará llevar a cabo para la realización de este proyecto, pero es necesaria la realización de un análisis de riesgos que, en caso de que ocurran no arruinen dicha planificación, y por lo tanto la correcta finalización del proyecto. En las tablas que se muestran a continuación aparecen identificados los riesgos que pueden afectar al normal desarrollo, junto con el plan de acción correspondiente para hacer que el impacto producido sea el menor posible, además aparece también la importancia de cada uno de ellos, ya que en caso de ocurrir simultáneamente es posible que se deban tratar de manera secuencial.

Número: 01	**Nombre del riesgo:** Indisposición durante el desarrollo del proyecto	**Fecha:** 02/2014
Descripción del riesgo		
A lo largo del desarrollo del proyecto, por ciertos motivos, no es posible realizar las tareas asignadas para ese periodo de tiempo		
Causas		
▪ Enfermedad ▪ Accidente laboral		
Consecuencias		
La/s tarea/s que se tuvieran que realizar en dicho periodo se realizarán posteriormente, aumentando la carga en días posteriores.		
Plan de acción		
Estrategia: Reservar el riesgo	**Descripción:** La solución será distribuir esas tareas entre el tiempo restante antes de la entrega del proyecto. De esta forma se consigue disminuir el impacto del riesgo.	
Impacto: Medio	**Probabilidad:** Baja	**Exposición:** 2 (*)

Tabla 1.2: Riesgo 01 - Indisposición durante el desarrollo del proyecto

<table>
<tr><td>Número: 02</td><td>Nombre del riesgo: Fallo de los equipos de desarrollo de pruebas y documentación</td><td>Fecha: 02/2014</td></tr>
<tr><td colspan="3">Descripción del riesgo</td></tr>
<tr><td colspan="3">Aparición de fallos de cualquier índole en cualquiera de los equipos destinados bien al desarrollo de las pruebas o la documentación.</td></tr>
<tr><td colspan="3">Causas</td></tr>
<tr><td colspan="3">▪ Fallos en el hardware de los equipos
▪ Infección por malware</td></tr>
<tr><td colspan="3">Consecuencias</td></tr>
<tr><td colspan="3">Implicará un retraso del proyecto, al tener que subsanar los problemas surgidos en las máquinas.</td></tr>
<tr><td colspan="3">Plan de acción:</td></tr>
<tr><td>Estrategia: Protección del riesgo / Aceptación del riesgo</td><td colspan="2">Descripción: El plan de acción de este riesgo consiste en disminuir la probabilidad de aparición del mismo a causa de malware, teniendo un antivirus actualizado y una correcta configuración en las mismas. En el caso de fallos hardware, se acepta ese riesgo debido a su probabilidad más baja respecto a la otra causa.</td></tr>
<tr><td>Impacto: Medio</td><td>Probabilidad: Baja</td><td>Exposición: 3 (*)</td></tr>
</table>

Tabla 1.3: Riesgo 02 - Fallo de los equipos de desarrollo de pruebas y documentación

Número: 03	**Nombre del riesgo:** Poca destreza con las herramientas	**Fecha:** 02/2014
Descripción del riesgo		
Aumento del tiempo necesario para la finalización del proyecto por no estar familiarizado con el entorno de MongoDB.		
Causas		
▪ Poca familiarización con las herramientas de desarrollo		
Consecuencias		
Es necesario invertir mayor cantidad de dinero en el proyecto, o mayor trabajo por día.		
Plan de acción		
Estrategia: Evitación del riesgo	**Descripción:** Realización de cursos y lectura de tutoriales fuera de la planificación temporal del proyecto o contar el aprendizaje en dicha planificación.	
Impacto: Alto	**Probabilidad:** Baja	**Exposición:** 4 (*)

Tabla 1.4: Riesgo 03 - Poca destreza con las herramientas

Número: 04	**Nombre del riesgo:** Estimaciones imprecisas	**Fecha:** 02/2014
Descripción del riesgo		
A causa de una mala planificación del proyecto, las estimaciones de usos de recursos y división temporal de las tareas, no son las adecuadas para el desarrollo correcto del proyecto.		
Causas		
▪ Demasiados hitos/tareas en la planificación ▪ Pocos hitos/tareas en la planificación		
Consecuencias		
La consecuencia es el fracaso del proyecto, si no se reacciona a las malas estimaciones a tiempo		
Plan de acción		
Estrategia: Protección del riesgo	**Descripción:** Para la protección de este riesgo, el plan de acción es llevar un seguimiento de la planificación para poder reaccionar a tiempo, redistribuyendo los recursos o añadiendo/eliminando ciertas tareas/hitos	
Impacto: Alto	**Probabilidad:**Medio	**Exposición:** 5 (*)

Tabla 1.5: Riesgo 04 - Estimaciones imprecisas

*La exposición de todos los riesgos ha sido calculada utilizando el ISO/IEC 27005:2008 mostrada en la siguiente ilustración.

		Likelihood of Incident Scenario				
		Very Low	Low	Medium	High	Very High
Business Impact	Very Low	0	1	2	3	4
	Low	1	2	3	4	5
	Medium	2	3	4	5	6
	High	3	4	5	6	7
	Very High	4	5	6	7	8

Figura 1.2: ISO/IEC 27005:2008

1.5. Resumen

Se pretende con el desarrollo de este proyecto realizar una investigación acerca de las nuevas tencologías NoSQL, de manera que se entienda mejor el comportamiento de las mismas. Para ello nos centraremos en el sistema gestor MongoDB, uno de los que ha ido adquiriendo mayor éxito en los últimos tiempos.

Este estudio se elabora a través de la realización de numerosas pruebas que se detallarán en el Capítulo 4, las cuales estarán enfocadas principalmente a la administración de bases de datos.

El principal objetivo el de aplicar los conocimientos que aquí se obtengan a la docencia, por lo que no se profundizará demasiado en algunos aspectos, el interés se centra en comportamientos comunes de las bases de datos y, especialmente de MongoDB.

La manera en la que se estructura este proyecto es la siguiente:

1. En el 'Capítulo 1 - Introducción y Objetivos', capítulo actual, se muestra una visión global del proyecto y los objetivos que se pretenden conseguir tras su realización, así como la manera en la que dicho proyecto se ha llevado a cabo y las herramientas necesarias para elaborarlo.

2. En el 'Capítulo 2 - Bases de datos NoSQL' nos centraremos en las bases de datos NoSQL. Se comenzará dando una visión global para comprender el entorno en el que se va a trabajar, y se finalizará dando una visión más detallada de nuestro objetivo, en este caso MongoDB.

3. El 'Capítulo 3 - Administración en MongoDB' está destinado a centrarnos ya en la administración en MongoDB. Es aquí donde nos encontramos una explicación detallada de en qué consistirán las pruebas y la realización de las mismas, tras lo cual se presentarán una serie de estadísticas y conclusiones obtenidas de los resultados de las mismas.

4. En el 'Capítulo 4 - Conclusiones generales' se realizará una valoración global de la investigación. Aquí se proporcionará una serie de conclusiones obtenidas a partir de los estudios realizados en el capítulo anterior, los cuales nos permitirán entender el comportamiento de la base de datos a estudiar.

Capítulo 2

Bases de datos NoSQL

A lo largo de este capítulo vamos a familiarizarnos con las bases de datos de tipo NoSQL. El principal objetivo será conocer y comprender el entorno con el que vamos a trabajar en el 'Capítulo 3 - Administración en MongoDB', saber qué motivaciones nos han llevado a la creación de este tipo de bases de datos, conocer también otros tipos NoSQL con el fin de proporcionar alternativas cuando se desarrolle una aplicación, e identificar diferencias entre ellas y las bases de datos relacionales, concretamente con el gestor MySQL, con el que estamos más familiarizados a la hora trabajar.

Tras este primer análisis nos centraremos también en MongoDB desde una perspectiva general, detallando sus características, sus ventajas e inconvenientes o las principales funciones que esta base de datos nos proporciona.

2.1. ¿Qué son las bases de datos NoSQL?

Como ya se comentaba inicialmente en la sección 1.1, las bases de datos NoSQL (*Not Only SQL*) se presentan como una alternativa a la bases de datos relacionales, pero es importante hacerse dos preguntas para comprender este entorno, ¿qué ha llevado a la aparición de estos tipos de sistemas de almacenamiento? y, ¿qué nos pueden proporcionar que las bases de tipo SQL no puedan?.

Para contestar a estas preguntas hemos de mirar hacia atrás y darnos cuenta del gran crecimiento y los grandes cambios que ha sufrido internet a lo largo de estos últimos años. En el Capítulo 1 se mencionó que en el año 1997 el tamaño de Internet era de unos 88 TB, mientras que en la actualidad ha llegado a los 5 EB, lo que supone unos 5.242.880 TB, es decir, en apenas veinte años internet es casi unas 60.000 veces más grande, en términos de información esto supone una mayor necesidad de almacenamiento y tratamiento de datos. Todo este crecimiento viene dado por el auge de la computación de la nube, la aparición de las redes sociales, los buscadores de internet, entre otros. Son todos estos elementos los que han hecho que las bases de datos relacionales vayan mostrando deficiencias, y por lo tanto, hacen que sea necesaria la aparición de una nueva generación que cubra con las necesidad actuales, las bases de datos NoSQL.

Estamos dando gran importancia al almacenamiento de los datos, pero no es la única característica a tener en cuenta cuando hablamos del desarrollo de las bases de datos, ya que ese nuevo tipo de aplicaciones que ha surgido, y el "nuevo" internet del que ahora disponemos no solo necesita de capacidad de almacenamiento, si no que también se ha de tener en cuenta el rendimiento, es necesario contar con un rendimiento alto, se debe proporcionar escalabilidad, ya que se espera que todo siga creciendo con el paso de los años y es importante que la adaptación a ese crecimiento se realice de manera sencilla, y por último, otra cualidad deseable es la de que los datos se almacenen de manera distribuida, no todo en el mismo lugar.

Todas estas cualidades son las deseables para las bases de datos actuales, por todo ello, para llegar a cumplir con esas necesidades, surge NoSQL. No obstante, no estamos hablando en ningún momento de que este tipo de bases puedan hacer que desaparezcan las relacionales, si no que en ámbitos como puede ser por ejemplo internet es más conveniente la utilización de sistemas NoSQL.

Podemos definir por lo tanto una base de datos NoSQL como un lugar donde se almacena información que 'rompe' con los métodos de gestión clásicos que presentaban las bases de datos relaciones. Dentro de este grupo encontramos diferentes tipos de bases de datos que se explicarán en la siguiente sección 2.3, por lo que no podemos hablar de un modelo de organización de datos único dentro de ellas, aunque en general dicha organización es menos rígida que cuando hablamos de SQL. Por lo tanto, centrándonos en la segunda pregunta las principales características, y comunes entre todos los tipos, que presentan las bases de datos NoSQL son aquellas que resuelven los problemas que acabamos de mencionar y que a continuación se listan. En la siguiente sección se describirán más a fondo estas características, por lo que aquí se explican de forma muy breve.

1. Estructura distribuida: Los datos normalmente se distribuyen utilizando mecanismos de tablas hash distribuidas (DHT).

2. Escalabilidad horizontal: La implementación se realiza en varios nodos en lugar de utilizar tan solo uno de grandes dimensiones.

3. Tolerancia a fallos y redundancia.

4. No generan cuellos de botella: ya que evita el problema de las bases de datos SQL, las cuales deben, en el mejor caso, transcribir la sentencia para que esta pueda ser ejecutada.

Pero, no todo es bueno, y es que como ya hemos dicho, NoSQL no es un reemplazo de SQL, si no una alternativa, ya que hay características necesarias de las cuales las segundas disponen pero no las primeras, como por ejemplo las características conocidas como **ACID**, una cualidad por la que las bases de datos de tipo relacional destacan y algo necesario en muchos escenarios. Es conveniente recordar las características que engloba el concepto de ACID:

1. ***A****tomicity* (Atomicidad): es quien nos asegura que la operación se ha realizado o no con éxito. Todas las instrucciones deben ejecutarse correctamente, si no es así ninguna deberá hacerlo.

2. ***C****onsistency* (Consistencia): es la propiedad que se encarga de asegurar que sólo se va a comenzar aquello que se puede finalizar.

3. ***I****solation* (Aislamiento): la ejecución de una instrucción no puede afectar a otras.

4. ***D****urability* (Durabilidad): cuando se realiza una operación, la durabilidad es la que nos asegura que persistirá, aún teniendo fallos en el sistema.

El objetivo principal de estas características es el de garantizar que las instrucciones se ejecuten como una transacción. Una lugar en el que estas transacciones pueden llegar a ser de vital importancia es por ejemplo en un banco, donde es necesario que se ejecuten todo el conjunto de instrucciones que empiecen, y si no es así dichas instrucciones deben deshacerse. En esta situación las bases de datos NoSQL no podrían garantizar la correcta finalización de la transacción.

2.2. Características de los sistemas NoSQL

A pesar de existir numerosos tipos de bases de datos NoSQL, quienes poseen cada una sus características propias que resuelven problemas en diferentes situaciones, los sistemas NoSQL cuentan también con una serie de características comunes, algunas de las cuales ya se venían presentando brevemente en la sección anterior. Comenzaremos profundizando en ellas.

En primer lugar presentan una *estructura distribuida*. Cuando hablamos de bases de datos distribuidas hablamos de un conjunto de bases que se encuentran lógicamente relacionadas pero se encuentran en sitios diferentes, pueden estar dentro de un área reducida o en largas distancias, interconectadas por una red de comunicaciones. La información, por lo tanto, se encuentra repartida entre todas esas bases o nodos. De manera lógica podemos verlas como un solo sistema, pero de manera física se pueden ver como varios sistemas. Así, el usuario podrá acceder a la información de cualquiera de esos nodos como si todos los datos se encontraran en el mismo lugar.

Hablabábamos en segundo lugar de la *escalabilidad horizontal*. La escalabilidad la podemos definir como la propiedad deseable que posee un sistema para reaccionar y afrontar el crecimiento sin perder calidad en los servicios que ofrece. Para situarnos, esta escalabilidad puede ser de dos tipos:

1. *Escalabilidad Horizontal:* Se dice cuando un sistema al agregar más nodos mejora el rendimiento.

2. *Escalabilidad Vertical:* Se dice cuando se añaden más recursos a un nodo del sistema y el sistema en conjunto mejora.

Como ya se ha dicho, la escalabilidad de la que hablamos cuando mencionamos los sistemas NoSQL es la horizontal, es decir, que la base de datos se formará con varios nodos, los cuales se podrán ir añadiendo de manera relativamente sencilla.

Otra de las principales características que encontramos en estos sistemas, es que *no cuentan con una estructura de datos definida*, si no que cada tipo NoSQL emplea la propia, al contrario que SQL, el cual estructuraba sus datos en tablas, lo que hace que las primeras sean *más flexibles* a la hora de almacenar datos.

Cuando hablamos de características de este tipo de sistemas encontramos el *teorema CAP*, debido que es un teorema relacionado con los sistemas distribuidos, característica ya mencionada. Este teorema nos dice que en estos

sistemas distribuidos es imposible garantizar simultáneamente las siguientes tres propiedades:

1. *Consistencia (Consistency):* la cual consiste en hacer que todos los nodos del sistema vean la misma información al mismo tiempo, es lo que conocemos como integridad de la información.

2. *Disponibilidad (Availability):* consiste en garantizar que el sistema se podrá utilizar a pesar de que uno de los nodos se haya caído.

3. *Tolerancia al particionamiento (Partition tolerance):* también conocido como tolerancia a fallos. Es la capacidad que tiene el sistema de seguir funcionando a pesar de que existan fallos parciales que dividan dicho sistema.

Para conocer mejor este concepto se presenta la siguiente ilustración, en la que aparecen los siguientes términos:

1. *AP:* Este tipo garantiza tanto la disponibilidad como la tolerancia a particiones, pero no la consistencia.

2. *CP:* Indica que garantiza la consistencia y la tolerancia.

3. *CA:* Encontramos aquí problemas con la tolerancia a particiones, ya que garantizan la consistencia y la disponibilidad.

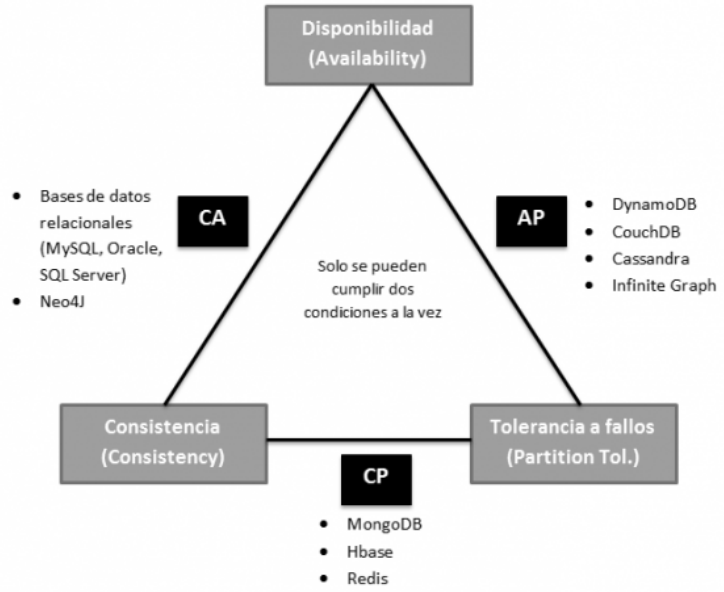

Figura 2.1: Teorema CAP

Todas estas características nos llevan por lo tanto a obtener una serie de ventajas e inconvenientes en este tipo de sistemas, algunas de los cuales ya se han ido mencionando.

En cuanto a las ventajas, dado que son sistemas distribuidos, cuentan con la alta velocidad de la que estos disponen, y de una alta capacidad de almacenamiento. Además, debido al bajo coste computacional que suponen, en comparación con sistemas de tipo SQL, hace que sea también más baratas económicamente, ya que necesitan de menos recursos para llevar a cabo su trabajo. Otra de sus ventajas es que responde muy bien tanto a la activación como desactivación de servicios, por lo que se puede adecuar a las necesidades que se tengan en el momento.

Por otro lado cuando hablamos de inconvenientes nos encontramos con que en ocasiones pueden sufrir pérdidas de datos e inconsistencia, ya que para adquirir velocidad solo vuelcan los datos a disco eventualmente, por lo que a veces los datos en memoria se pierde. Además, aun no están lo suficientemente maduros para incorporarlos en algunos ámbitos debido a su grado de novedad, algo que también afecta a la hora de la compatibilidad con otros sistemas.

2.3. Clasificación de las bases de datos NoSQL

Dentro de las bases de datos NoSQL encontramos diversos tipos según su implementación, los cuales disponen de diferentes características, lo que permite que puedan adaptarse a diferentes situaciones.

1. Bases de datos documentales.
2. Bases de datos orientadas a grafos.
3. Bases de datos clave/valor.
4. Bases de datos tabular.

A lo largo de esta sección se proporcionará una ligera visión de cada uno de ellos, en la que encontraremos su definición, la manera en la que almacenan los datos y algunas de sus ventajas, utilidades y lugares donde podremos encontrarlas.

2.3.1. Bases de datos documentales

Las bases de datos de tipo documental van a ser, como se verá más adelante, el principal objeto de nuestro estudio en este proyecto, centrándonos principalmente en uno de sus grandes ejemplos, ***MongoDB***.

Es uno de los modelos más utilizados dentro de NoSQL y motivo por el cual las bases NoSQL han adquirido tanto éxito, algo que también es uno de los principales motivos por el que el estudio se ha centrado en ellas. La forma en la que almacena la información es en colecciones de documentos. Para comprender la estructura de un documento se muestra a continuación un pequeño fragmento:

```
{
   Nombre:"Natalia",
   Apellidos:"Román Seneque"
   Dirección:"Palencia",
   Teléfono:[
      {Móvil:"612345678", Compañía:"Vodafone"},
      {Fijo:"987654321", Compañía:"Movistar"}
   ]
}
```

Los documentos pueden ser comparables a los registros o tuplas que poseen las bases de datos relaciones, aunque los documentos no se adaptan a ningún esquema, es decir, son *schema-less* (sin esquema). Cuando hablamos de este tipo de bases de datos podemos decir que son un tipo especial de las bases de datos clave-valor que se explicarán más adelante, con todas las ventajas que dicho tipo pueda proporcionar, pero con la excepción de que el valor se almacena de manera que el servidor pueda entenderlo y realizar operaciones sobre los datos. El formato en el que suele ser almacenado es JSON (*JavaScript Object Notation*), un formato ligero destinado al intercambio de datos, aunque se puede almacenar en otros tipos como puede ser XML (*Extensible Markup Language*). Otra de sus grandes características es que no solo nos permite realizar búsquedas del tipo clave-valor, si no que se pueden hacer búsquedas por el contenido de los documentos a través de consultas más complejas.

Algunas de las ventajas más destacables de las bases de datos documentales además de su alto rendimiento, es que no están sujetas a ningún esquema estricto como ya se ha mencionado, el escalado horizontal también goza de una

gran sencillez y además son de gran utilidad para la computación en la nube, algo a la orden del día.

Además de *MongoDB*, del cual aun no se proporcionarán más detalles, encontramos por ejemplo ***CouchDB*** desarrollado por Apache que pretende proporcionar facilidad de uso a sus clientes sin olvidar todas las ventajas que dan las bases de datos documentales. Algunos lugares donde podemos encontrarlo es en algunas aplicaciones de Android como *SpreadLyrics* o en algunas aplicaciones de Facebook como *Birthday Greetin Cards*, además empresas como la BBC lo utilizan para sus contenidos dinámicos. Aunque no todo es bueno, ya que Ubuntu, que utilizaba *CouchDB* para su servicio de *Ubuntu One*, en el año 2011 dejó de utilizarlo por problemas de escalabilidad. Junto con *CouchDB* encontramos también ***RavenDB*** de *Hibernating Rhinos* o ***IMB Lotus Domino***

2.3.2. Bases de datos orientadas a grafos

Las bases de datos orientadas a grafos como su propio nombre indica, almacenan la información en forma de nodos y relaciones entre ellos. Los nodos representan las entidades, y las relaciones vienen representadas como aristas entre dichos nodos, además, dichas relaciones tienen la capacidad de poseer atributos. Esta forma de almacenar la información va a permitir utilizar la teoría de grafos para recorrer la base de datos.

Un ejemplo de cómo se almacenan los datos en este tipo de bases de datos se muestra en la siguiente ilustración:

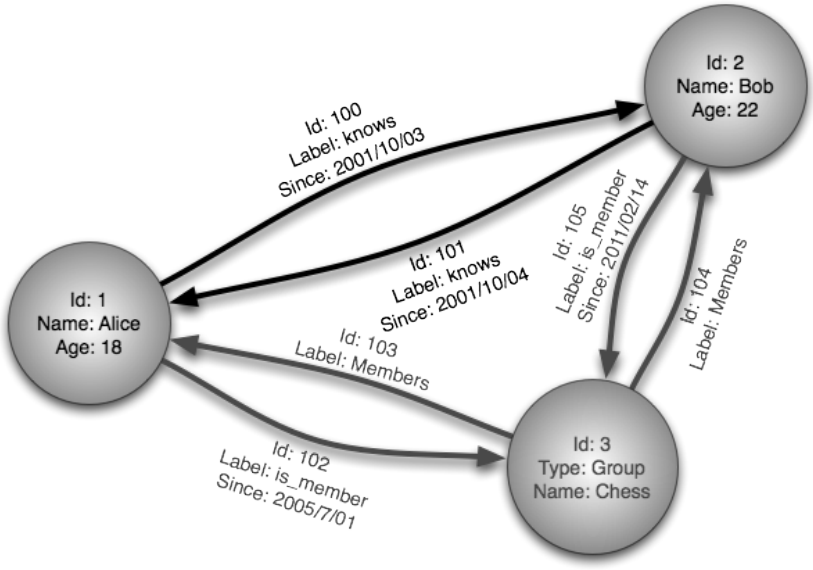

Figura 2.2: Representación de base de datos orientada a grafos

Las bases de datos de este tipo están diseñadas principalmente para el uso transaccional de sistemas, para optimizar el rendimiento y proporcionar tanto integridad en los datos como disponibilidad operacional. Sus tablas cuentan con tan solo una columna, y cada una de sus relaciones con dos.

En este caso no solo encontramos ventajas frente a la utilización de bases de datos relacionales, sino también frente a bases de datos NoSQL de otros tipos. Dichas ventajas las encontramos en diferentes características como por ejemplo

el **rendimiento**, ya que tiende a permanecer constante, la **flexibilidad**, es fácil adaptar el modelo de datos a las necesidades, o la **agilidad**, se da un mantenimiento progresivo.

Bases de datos orientadas a grafos son por ejemplo ***Neo4j***, una de las más famosas que cuenta con código abierto y es utilizado por grandes empresas como *Adobe*, *Cisco* o *hp* entre muchas otras, ***Infinite Graph***, quien colabora con empresas como *Vodafone* u *Oracle*, ***HyperGraphDB***, ***AllegroGraph***, etc.

2.3.3. Bases de datos clave-valor

Estamos ante uno de los tipos de bases de datos NoSQL más utilizados y más sencillos de utilizar y comprender, aunque su implementación puede llegar a ser verdaderamente compleja. La utilización de los pares clave y valor para obtener información son utilizados en otros ámbitos. En el caso de las bases de datos sigue la filosofía ya conocida (recordada en la Ilustración que se muestra a continuación, se obtiene cierta información a partir de una clave que es única. Este tipo de bases de datos tiene como fin principal el de almacenar información, no tiene gran utilidad para otras funciones como puede ser el análisis de los datos almacenados.

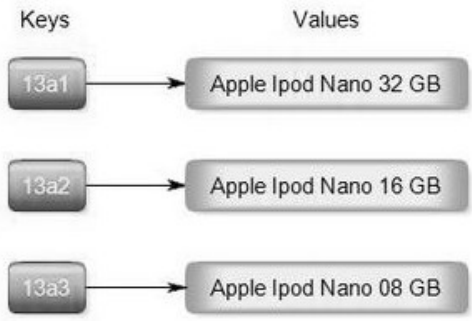

Figura 2.3: Representación de base de datos clave-valor

En la actualidad la escalabilidad horizontal va cobrando mayor protagonismo en las empresas debido a la gran cantidad de datos que maneja, algo ante lo que este tipo de bases de datos responde satisfactoriamente. Además, tiene la capacidad de manejar grandes cantidades de datos sin generar cuellos de

botella, lo que en ocasiones puede resultar crítico. Junto con ello, cuenta con la característica de proporcionar un escalamiento sencillo. Por supuesto, toda esa sencillez que se ha ido mencionando, la cual es un motivo por el cual destaca, hace que también haya que renunciar a cierta funcionalidad, como por ejemplo, la verificación de la integridad de datos, las claves foráneas o los *triggers*.

Cassandra es una de las bases de datos clave-valor más importantes, está desarrollada en Java por Apache, y entre sus clientes más famosos está la red social Twitter. Más ejemplos de este tipo son por ejemplo ***BigTable*** desarrollago por Google, ***Dynamo*** por Amazon o ***Voldemort*** por Linkedin.

2.3.4. Bases de datos tabular

Bases de datos conocidas también con el nombre de bases de datos orientadas a columnas. Son un tipo NoSQL muy similar a las bases de datos relacionales, pero en lugar de almacenar registros almacenan columnas de datos, las cuales son tratadas de manera individual. Se muestra en la ilustración de manera gráfica la forma en la que los datos se guardan, junto con ello aparece también el almacenamiento en filas, lo que nos permite ver más claramente las diferencias entre ambos.

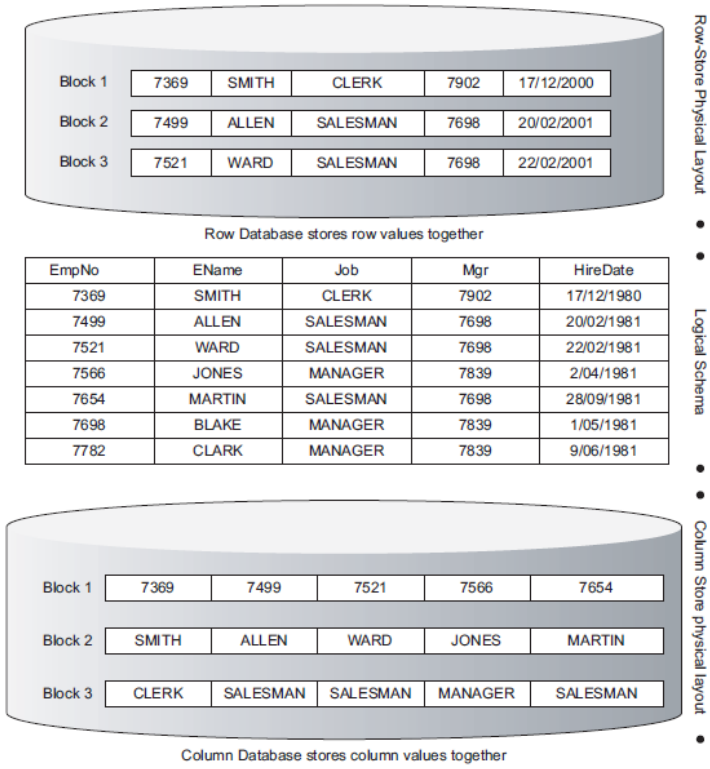

Figura 2.4: Representación base de datos orientada a columnas

Las bases de datos orientadas a columnas están diseñadas especialmente para tratar con grandes cantidades de datos, concretamente, destacan para la realización de consultas y agregaciones sobre sus datos de manera más rápida.

Además los valores de cada una de estas columnas son almacenados de manera contigua, lo que hace que los accesos también se realicen de forma más rápida.

En este tipo de bases de datos encontramos de nuevo la base de datos Cassandra, para que veamos la gran similitud entre el tipo clave-valor y las orientadas a columnas. Junto con ello destacan ***Bigtable*** de Google, ***Hypertable*** o ***HBase***.

2.4. Bases de datos SQL vs. Bases de datos NoSQL

Hasta el momento, tan solo se ha tenido experiencia en manejar bases de datos relacionales, especialmente se ha tenido contacto con MySQL. En esta sección se pretende hacer una comparación más exhaustiva entre las bases de datos de tipo NoSQL, las ya vistas, y las bases de datos relacionales (MySQL, Oracle, SqLite, etc.), con el fin de comprender mejor sus diferencias y comprender por qué las bases de datos relacionales no han sido suficiente para soportar la evolución de las nuevas tecnologías.

Ya se han ido mencionando a lo largo de este documento varias diferencias importantes entre ambos tipos de bases de datos, como puede ser por ejemplo la estructura de datos, no definida en las bases de datos no relacionales pero sí en las de tipo SQL, o también las características **ACID**, las cuales encontramos en el segundo tipo pero no en el primero, o el teorema **CAP**. Incluso la clasificación de ambos tipos es totalmente diferente, mientras que si hablamos de bases de datos relacionales diferenciamos entre tan solo código abierto y cerrado, en las no relacionales encontramos los ya mencionados, bases de datos orientadas a grafos, a documentos, etc. Pero no son las únicas diferencias destacables.

Se ha hablado también de la escalabilidad horizontal, característica propia de las bases de datos NoSQL, mientras que si pasamos a hablar del mundo relacional lo más común es encontrarnos con la estalabilidad vertical. Cuando trabajamos con bases de datos SQL estamos trabajando estrictamente con tan solo un nodo, mientras que con NoSQL la base de datos puede estar compuesta por varios nodos.

A pesar de que no se ha entrado en detalles sobre cómo se realizan las consultas sobre cada tipo de bases de datos, en general, podemos llamar al lenguaje que se utiliza en NoSQL un lenguaje no estructural **UnQL**, el cual depende de la base de datos donde le utilicemos, frente a este, encontramos por supuesto el lenguaje estructurado **SQL**. De igual manera, todos los soportes de tipo SQL son capaces de almacenar procedimientos, mientras que esto no ocurre en NoSQL.

Se ha de tener en cuenta también, que cuando hablamos de bases de datos no relacionales nos exponemos también a no tener respuestas correctas o completas, ya que para este tipo es más importante conseguir una respuesta rápida que una correcta.

La antigüedad de la que disponen las bases de datos relacionales frente al grado de novedad de las NoSQL hacen que encontremos también diferencias incluso a nivel de implementación y mantenimiento, ya que en ocasiones puede ser dificil incorporar un sistema no relacional a equipos ya existentes, mientras que una base de datos relacional es compatible con una gran cantidad de herramientas. Además el poco conocimiento de las mismas, puede llegar a hacer que su mantenimiento no sea sencillo.

A pesar de todas estas diferencias ambos tipos de bases de datos pueden ser capaces de trabajar juntos, algo que en muchas ocasiones es la mejor opción, ya que las características de las que no dispone un tipo las puede complementar el otro, obteniendo así grandes resultados. Por supuesto, siempre se ha de estudiar el escenario en el que se va a trabajar para tomar la mejor decisión.

Capítulo 3

MongoDB

Una vez que tenemos una ligera idea de en qué consisten los sistemas NoSQL y cual es su filosofía, vamos a centrarnos un poco más en nuestro objeto de estudio, MongoDB. En esta sección vamos a dar una visión global que recoge los principales conceptos que guardan relacionados con ella, consiguiendo así familiarizarnos con su funcionamiento y características antes de proceder al diseño y la realización de las pruebas.

También vamos a incluir en este capítulo, con el fin de completar la sección anterior 2.4, un pequeño conjunto de pruebas que enfrentan las características de MongoDB y MySQL, de manera que podamos comparar el donde venimos (MySQL) con el adonde vamos (MongoDB).

3.1. ¿Qué es MongoDB?

MongoDB es una base de datos NoSQL de tipo documental de código abierto. Es uno de los sistemas NoSQL que ha adquirido más éxito desde la aparición de los mismos, lo que ha influido en la decisión a la hora de seleccionar MongoDB como objecto del estudio, en dicho éxito han influido características como su esquema libre, su alto rendimiento, su alta disponibilidad y su escalado automático, de las cuales hablaremos de manera más detenida posteriormente.

Para que nos hagamos una idea de su importancia, podemos encontrar a grandes empresas y marcas conocidas utilizando sistemas de MongoDB ya sea de manera total para su aplicación o de manera parcial en pequeñas aplicaciones internas. Algunos ejemplos de ello son, ***MailBox***, producto de Dropbox, ***Expedia***, una de las grandes compañías de viaje online, ***Linkedin***, ***SAP*** o ***MTV*** entre otros muchos. Además, Mongo colabora con socios como IBM o Red Hat, grandes empresas dentro del mundo de las tecnologías de la información, entre otras muchas. Pero, centrémonos en comprender su funcionamiento para entender el por qué de este éxito.

Como base de datos documental, MonogoDB utiliza como registros, como unidad básica de almacenamiento, los documentos, los cuales guardan gran similitud con objetos JSON (*JavaScript Object Notation*), algo en lo que nos centraremos más adelante. Estos documentos son lo que sustituyen a lo que anteriormente conocíamos como fila en los sistemas SQL. La estructura que poseen estos documentos es del tipo clave:valor, es decir:

```
{
  clave : "valor",
  nombre : "Natalia",
  edad : 22,
  aficiones : ["leer", "escuchar música"]
}
```

Con este pequeño ejemplo podemos darnos cuenta de varias características que poseen los documentos, en primer lugar cada clave posee un valor y dichos valores tienen diferentes tipos de datos, las dos primeras son strings, la edad es un entero y las aficiones vienen en forma de array, a pesar de la variedad de tipos dentro de los valores, donde incluso podemos llegar a encontrarnos otros documentos o arrays de documentos, las claves siempre son de tipo string. En

segundo lugar, los pares clave valor se separan entre ellos por comas. Además, nos damos cuenta también de que cada clave debe ser única, siendo estas sensibles al uso de mayúsculas y minúsculas. Junto con todo ello, MongoDB asigna a cada documento un número de identificación (_id) único dentro de su colección.

Una colección es lo que conocemos como un conjunto de documentos. En comparación con los sistemas SQL una colección vendría a ser el equivalentes de las tablas. Las grandes ventajas que nos presentan las colecciones entre otras son por ejemplo, la creación automática de las mismas si estas no existen, y su gran flexibilidad, ya que los documentos que contienen no tienen por qué poseer el mismo formato. Otra característica más de estas colecciones es que a su vez se pueden agrupar en subcolecciones.

Por último, al igual que los documentos se agrupan en colecciones, estas colecciones se agrupan en bases de datos. En los sistemas SQL eran las tablas las que agrupábamos en la base de datos. La principal característica es que una instancia de MongoDB es capaz de albergar varias bases de datos.

Pero, todos estos documentos es necesario almacenarlos de alguna manera. Bien, para ello MongoDB utiliza BSON (*Binary JSON*) un formato de intercambio de datos, lo que le proporciona grandes características como flexibilidad, una alta velocidad de consulta o una fácil gestión y codificación. Lo que caracteriza a BSON es que almacena de forma más explícita información como las longitudes que poseen los campos, los índices de arrays, etc. Aunque, en realidad, con lo que realmente trabajaremos serña con JSON, ya mencionábamos anteriormente que los documentos de MongoDB tenían un formato muy similar a objetos de este tipo, por ejemplo:

```
Var ejemplo = {
      "clave" : "valor",
      "nombre" : "Natalia",
      "edad" : "22",
      "aficiones" : [
           "leer",
           "escuchar música"
      ]
};
```

Se puede observar claramente en el ejemplo la similitud con el formato que poseen los documentos de MongoDB.

3.2. Instalación de MongoDB

La instalación de MongoDB se caracteriza por ser un proceso muy sencillo. A pesar de que dicha instalación se ha llevado a cabo en entornos Windows como Linux, será en este último sistema operativo en el cual se desarrollarán las prácticas, por lo que es donde nos centraremos.

En primer lugar procedemos a descargar MongoDB desde su página oficial, disponible en el enlace [9]. En este caso se utilizará la versión 2.6.1 de 64 bit, la última versión disponible en el momento de la realización de este proyecto. En segundo lugar, para comenzar a utilizar nuestra base de datos basta con descomprimir nuestro fichero tgz y conectarnos a mongo. Es posible que en algunos casos haya que iniciar el servicio de mongodb, es decir, que tengamos que iniciar una instancia mongod, en el caso de nuestra versión no es necesario, ya que se inicia de manera automática.

3.3. Características

Se han ido mencionando ya una serie de caraterísticas a lo largo de todo el capítulo, pero profundicemos más en ellas.

En primer lugar, debido a que estamos ante una base de datos NoSQL, MongoDB es *schemaless*, es decir, sin esquema, esto permite almacenar documentos sin necesidad de que cumplan un esquema predefinido. En relación con los documentos, encontramos también la flexibilidad, la cual vienen dada por realizar el almacenamiento de los mismos a través de la utilización de JSON y BSON, ya que JSON nos proporciona un modelo de datos que evoluciona de manera más sencilla que los esquemas que nos presentan los sistemas relacionales.

En segundo lugar cuenta con una alta velocidad y con facilidad a la hora de aumentar la escalabilidad, esto se produce debido a que, como ya mencionamos en secciones anterior, estos sistemas no disponen de carga transaccional, lo que les lleva a adquirir estas características. Además, el escalado horizontal lleva a esta base de datos a poseer otra ventaja más, y es que la escalabilidad horizontal permite utilizar un conjunto de máquinas menos potentes, lo que ahorra dinero, o lo que permite incluir una mayor cantidad de ellas.

MongoDB, a pesar de ser un sistema NoSQL, ha conservado numerosas características que poseían ya los sistemas de bases de datos relacionales, como por ejemplo los índices secundarios, la ordenación, las consultas dinámicas, entre otras, lo que beneficia sobre todo a auqellos usuarios familiarizados con bases de datos relacionales.

A todo ello, algo relevante en cualquier herramienta que utilicemos, se le añade la facilidad de uso. MongoDB proporciona métodos sencillos tanto de instalación como de configuración, mantenimiento y uso. Para conseguirlo, intenta realizar ”lo correcto” de manera automática siempre que tenga posibilidad.

3.4. Operaciones de MongoDB

Es importante también conocer las operaciones que podemos realizar en este sistema y, por supuesto, como se realizan, para lo cual MongoDB nos proporciona una línea de comandos.

Las operaciones que trataremos aquí incluirá la inserción de documentos, la búsqueda, la modificación y la eliminación. Pero, para ejecutar estas instrucciones debemos conocer la notación de las consultas, la cual, de nuevo, es muy intuitiva. En primer lugar se indica la base de datos y la colección junto con la operación a realizar y entre paréntesis indicamos los criterios que debe cumplir, a lo cual se le puede añadir un modificador, veamos un ejemplo.

```
baseDatos.colección.operación({criterios}).modificador
```

```
db.usuarios.find({age : 22}).limit(3)
```

En el ejemlo indicamos que queremos buscar en usuarios, el cual se encuentra en la base de datos "db", aquellos usuarios que tienen una edad de 22 años, y además le indicamos que tan solo muestre 3 resultados.

Insertar

La inserción de documentos se puede realizar a través de tres métodos, tanto a través de la función *insert()*, como a través de la función *update()*, o utilizando *save()*.

El método *insert()* se encarga de añadir documentos a una colección. Por ejemplo, vemos a continuación la manera en la que se añade un nuevo usuario a la colección usuarios, a dicho usuario le vamos a proporcionar un _id concreto, aunque se le podría no indicar ninguno y MongoDB le proporcionaría uno único dentro de dicha colección.

```
db.usuarios.insert({_id : 5, nombre : "Natalia", edad : 22})
```

En segundo lugar, *update()*, utilizando su opción *upsert* crea un nuevo documento si el criterio de actualización de la consulta no ha sido encontrado. A continuación, aparece un ejemplo en el que se actualiza el campo dni de la colección de usuarios. En él cabe destacar la utilización de **upsert : true** mencionado anteriormente.

```
db.usuarios.update(
                    { nombre : "Natalia", edad : "22" },
                    { $set : { dni : "12345678A" } },
                    { upsert : true }
                  )
```

Por último, se hablaba también de *save()*, el cual, cuando se le proporciona un documento sin valor de _id lo que hace será una función *insert()* asignándole automáticamente un valor de _id.

Buscar

A la hora de realizar búsquedas de documentos dentro de una colección utilizamos la función *find()*, la cual nos devuelve un cursor hacia los elementos seleccionados. Esta selección puede ser desde un documento en concreto o puede incluir varios o incluso todos los documentos de una colección.

Si utilizamos tan solo la sentencia *db.usuarios.find()* obtendremos como resultado todos usuarios que pertenecen a dicha colección. Para seleccionar un campo en concreto deberemos insertar la condición correspondiente, por ejemplo:

```
db.usuarios.find({ edad : "22" })
```

Donde tendremos como resultado:

```
_id: 5,
nombre: "Natalia",
edad: 22
```

Otra característica de este método es que nos permite seleccionar los campos que queremos que se proyecten en la salida, para ello asignaremos un 0 a aquellos campos que no queremos que aparezcan y un 1 a los que sí. Mostramos en el siguiente ejemplo la manera en la que eliminaríamos el _id en la salida de la anterior consulta.

```
db.usuarios.find({edad : "22"}, {_id : 0})
```

Por último, *find()* cuenta también con la posibilidad de seleccionar tan solo un resultado a través de la instrucción *findOne()*.

Modificar

Ya hemos hablado en la inserción de documentos de los dos métodos que pueden ser utilizados para la modificación de documentos. Principalmente encontramos la función *update()*, por otro lado, alternativamente encontramos la de *save()*.

El método *update()* nos va a permitir actualizar tanto un documento como varios documentos a la vez gracias a su opción *multi:true*. La manera de utilizarlo es similar al resto de instrucciones como se ve en el ejemplo, en el cual vamos a actualizar la colección de usuarios añadiendo un campo que indique la ciudad donde vive.

```
db.usuarios.update({nombre : "Natalia", edad : 22},
{ciudad : "Palencia"})
```

Por lo tanto, si utilizaramos la función *find()* para buscar este documento tendría como salida:

```
_id: 5,
nombre: "Natalia",
edad: 22,
ciudad: "Palencia"
```

Por otro lado, la instrucción *save()* actúa como *update()* cuando se le proporciona un documento con una _id.

Eliminar

Para realizar el borrado permanente de documentos de la base de datos utilizamos el método *remove()*. Al igual que en los anteriores casos podemos desde eliminar un solo documento que reúna unos requisitos como eliminar todos los documentos de una colección, por lo que hay que tener especial cuidado cuando lo utilizamos.

Cuando deseamos borrar todos los documentos de la colección basta con no añadir ningún parámetro a dicho método, aunque en dicho caso puede ser aconsejable la utilización de *drop()*, el cual también eliminará los índices correspondientes.

Por otro lado, cuando indicamos requisitos del documento que queremos eliminar bien podemos hacer que se eliminen todos los que cumplan dicha condición de la manera que se muestra en el ejemplo:

```
db.usuarios.remove({edad : 22})
```

Como eliminar tan solo un documento que cumpla la condición de la siguiente manera:

```
db.usuarios.remove({edad : 22}, 1)
```

3.5. MongoDB vs MySQL

Ya se ha visto en el capítulo anterior una pequeña comparación de sistemas SQL con sistemas de bases de datos NoSQL, lo que se pretende en esta sección es ir más lejos y centrarnos tan solo en MySQL y MongoDB. El objetivo aquí es el de realizar una serie de pruebas utilizando estos dos tipos de bases de datos, de manera que nos proporcionen mayor información acerca del funcionamiento de las mismas. Además, a través del análisis de dichos datos se finalizará la sección aportando una serie de conclusiones.

Para realizar las mediciones de la manera más similar posible ha sido necesario tener en cuenta varias características:

1. **Documentos con los que se trabajan:** Es importante disponer en ambas bases de datos un fichero que disponga de la misma estructura y los mismos datos, para ello, como ya se mencionó anteriormente en 1.3, ha sido necesaria la uitlización de la herramienta ***generatedata***. Con esta herramienta se ha creado un fichero en formato '.csv' con 100.000 registros (cantidad que se ha considerado lo suficientemente grande para la realización de las pruebas), que nos permite poblar ambas bases de datos gracias a la ayuda de ***phpMyAdmin*** en MySQL y a ***mongoimport*** en MongoDB. Una vez pobladas en cada una de ellas se ha exportado los datos en su formato propio, es decir, se ha obtenido un fichero '.sql' y un fichero '.json' correspondientemente. Este fichero dispone de un campo numérico que nos servirá como clave en ambas bases, de esta manera se consigue homogeneidad, cuatro campos de tipo cadena y uno numérico.

2. **Configuración:** Aunque en el caso de MongoDB no ha sido necesario modificar la configuración que emplea, en el caso de MySQL ha sido necesario tener en cuenta a la hora de crear la tabla correspondiente que éste debe ser *MyISAM*, ya que es más similar a MongoDB.

3. **Toma de tiempos:** Es importante también elegir la herramienta con la que se tomarán los tiempos. Por ello, se ha decidido utilizar dos scripts en Python que realicen tanto la conexión con la base de datos correspondiente, como la ejecución de la sentencia correspondiente, esto lo que nos permite es utilizar la herramienta **time** de Linux para la ejecución de dicho script, y en su interior se utilizará la herramienta **timeit()** para medir tan solo la ejecución de la sentencia que nos interesa. En el caso de la inserción tan solo se requerirá la utilización de la primera de ellas.

Las pruebas incluyen las operaciones más comunes que se pueden llegar a realizar en una base de datos:

1. **Inserción de datos:** Inserción de los 100.000 datos en cada una de las bases de datos a través de su fichero correspondiente.

2. **Búsqueda de datos:** Se obtienen todos los datos que se encuentran en cada base de datos y, además, se obtiene también un conjunto de datos que cumpla algún criterio.

3. **Modificación de datos:** Modificación de algunos elementos que cumplan ciertos requisitos.

4. **Borrado de datos:** Se eliminan todos los datos de la tabla/colección correspondiente.

3.5.1. Inserción de datos

Una vez obtenidos los ficheros que contienen los datos de ambas bases procedemos a realizar la inserción en cada una de ellas y a tomar los tiempos correspondientes. Para realizar las mediciones en este caso se ha utilizado tan solo el comando **time** de Linux, ya que en MySQL se puede volcar el fichero al realizar la conexión y en MongoDB disponemos de la herramienta mongoimport como se mostrará a continuación.

En **MySQL** la inserción se ha realizado utilizando la sentencia:

```
mysql -uroot -ppass proyecto < benchmarkproyecto.sql
```

En el caso de **MongoDB** se ha utilizado la herramienta mongoimport de la cual se dispone:

```
mongoimport -d proyecto -c benchmarkproyecto
--file poblarMongoDB100Mil.json
```

Los resultados que se han obtenido en ambos casos son los que se muestran en la siguiente tabla:

Base de datos	Tiempo real	Tiempo de usuario	Tiempo del sistema
MySQL	6.738s	1.085s	0.910s
MongoDB	5.556s	0.646s	0.092s

Tabla 3.1: MySQL vs. MongoDB - Inserción

3.5.2. Búsqueda de datos

Para la búsqueda de datos se van a dar dos casos, en el primero de ellos se obtendrá el tiempo necesario para tener todos los datos que contiene dicha base, mientras que en el segundo se va a imponer la condición de que nos devuelva aquellos registros o documentos cuya edad es de 18.

En **MySQL** se han utilizado las siguientes sentencias, la primera de ellas para obtener todos los datos, mientras que la segunda presenta la condición correspondiente:

```
(1) SELECT * FROM benchmarkproyecto;
(2) SELECT * FROM benchmarkproyecto WHERE Edad = 18;
```

En el caso de **MongoDB** las consultas realizadas son:

```
(1) db.benchmarkproyecto.find()
(2) db.benchmarkproyecto.find({Edad:18})
```

Los resultados que se han obtenido en ambos casos son los que se muestran en la siguiente tabla:

Base de datos	Datos obtenidos	Tiempo real	Tiempo de usuario	Tiempo del sistema
MySQL	Todos	0.695s	0.071s	0.019s
	Edad = 18	0.764s	0.064s	0.016s
MongoDB	Todos	0.635s	0.078s	0.025s
	Edad = 18	0.668s	0.065s	0.019s

Tabla 3.2: MySQL vs. MongoDB - Búsqueda

3.5.3. Modificación de datos

Para este tipo de pruebas nos hemos decantado por utilizar un criterio de igualdad y modificar uno de los campos de tipo cadena. El número de registros a modificar es de 2.120, que es el número de resultados que coincide con el criterio 'Edad=18'.

Para la modificación en **MySQL** se utiliza la sigente sentencia update:

```
UPDATE benchmarkproyecto SET Fecha_de_nacimiento = '00/00/0000'
WHERE Edad = 18;
```

En el caso de **MongoDB** se ha utilizado la herramienta mongoimport de la cual dispone:

```
db.benchmarkproyecto.update({Edad:18},
{$set{Fecha_de_nacimiento:'00/00/000'}}, multi:true)
```

Los resultados que se han obtenido en ambos casos son los que se muestran en la siguiente tabla:

Base de datos	Tiempo real	Tiempo de usuario	Tiempo del sistema
MySQL	0.803s	0.084s	0.018s
MongoDB	0.674s	0.075s	0.025s

Tabla 3.3: MySQL vs. MongoDB - Modificación

3.5.4. Borrado de datos

Por último, la eliminación se realizará de todos los datos que se encuentran en la base de datos, es decir, de los 100.000 registros o documentos.

Se eliminan todos los registros en **MySQL**

```
DELETE FROM benchmarkproyecto;
```

En el caso de **MongoDB** se ha utilizado la herramienta mongoimport de la cual dispone:

```
db.benchmarkproyecto.remove()
```

Los resultados que se han obtenido en ambos casos son los que se muestran en la siguiente tabla:

Base de datos	Tiempo real	Tiempo de usuario	Tiempo del sistema
MySQL	0.073s	0.063s	0.009s
MongoDB	0.078s	0.066s	0.013s

Tabla 3.4: MySQL vs. MongoDB - Borrado

3.5.5. Conclusiones

Antes de analizar los datos obtenidos se va a realizar una nueva toma de tiempos de manera que se incluyan todas las operaciones anteriormente mencionadas, la búsqueda se realizará obteniendo todos los registros, es decir, sin utilizar condiciones. Los resultados obtenidos en este caso son:

Base de datos	Tiempo real	Tiempo de usuario	Tiempo del sistema
MySQL	1.096s	0.180s	0.030s
MongoDB	0.630s	0.048s	0.019s

Tabla 3.5: MySQL vs. MongoDB

Una vez obtenidos todos los tiempos se puede ver claramente la diferencia entre ambos sistemas, a continuación se presentan de manera gráfica los datos anteriormente obtenidos. Se muestran en estas gráficas tanto el tiempo de usuario como el tiempo real dedicado a operación, ya que son los datos más relevantes de los anteriormente obtenidos:

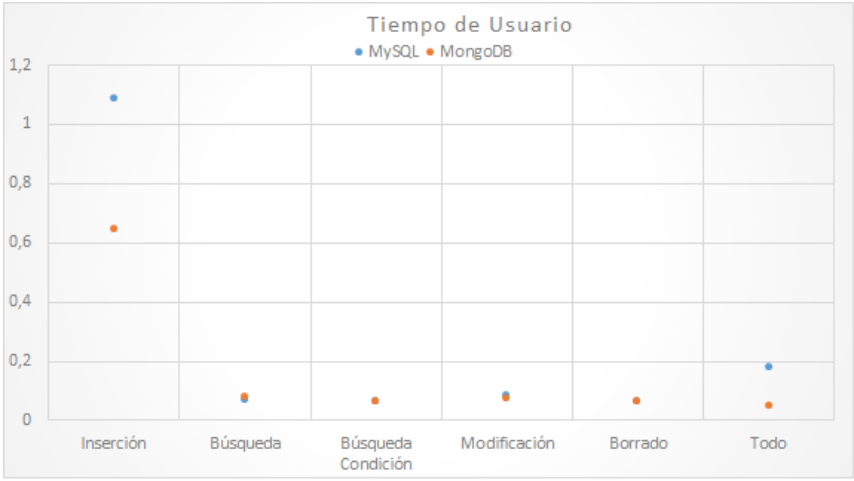

Figura 3.1: Tiempo de usuario MySQL vs. MongoDB

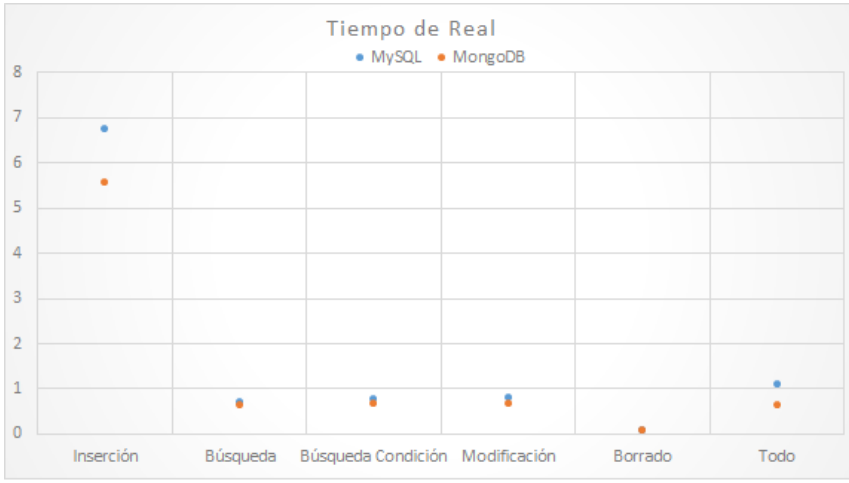

Figura 3.2: Tiempo real MySQL vs. MongoDB

Podemos ver claramente, que, a pesar de que en la mayor parte de las operaciones, véase, la búsqueda con y sin condiciones, la modificación y el borrado, los tiempos son muy similares entre ambas, la diferencia entre la inserción y la ejecución de todas las operaciones, precisamente las tareas con más carga de trabajo, es notable. Por lo que sin duda, estos datos nos llevan a pensar que el rendimiento de MongoDB es mayor que el de MySQL.

Pero, se ha de tener en cuenta que ningún sistema dispone de ninguna ventaja sobre el otro, es decir, ninguno de los dos ha sufrido modificaciones con el fin de optimizarlo, y, además, tenemos que pensar que se ha seleccionado el escenario más simple posible para cada uno de ellos, por lo que estos resultados nos sirven para hacernos una primera idea de ambos sistemas y sus diferencias, pero no para hacernos una idea definitiva de su rendimiento y funcionamiento, ya que a través de la utilización de distintas técnicas de optimización, o, tal vez, si nos encontráramos en otro tipo de escenario, bien podrían mantenerse estas diferencias o quizá MySQL podría pasar a ser el ganador.

Capítulo 4

Administración en MongoDB

A lo largo de este capítulo recorreremos la administración de las bases de datos desde una perspectiva global hacia una perspectiva más detallada de la administración en MongoDB, de manera que comprendamos mejor el contexto en el que se realizarán las pruebas. Será en este capítulo también donde se definirán las pruebas que se han realizado y se mostrarán los resultados obtenidos, los cuales serán posteriormente analizados con el fin de sacar diversas conclusiones y estadísticas, y entender así el comportamiento de este tipo de bases de datos ante las diversas situaciones presentadas.

4.1. Introducción a la administración

Una vez conocidos los conceptos con los que vamos a trabajar, las herramientas, las ventajas y desventajas que nos proporciona su utilización frente a otros tipos de bases de datos, etc. vamos a centrarnos en el principal objeto de nuestro estudio, la administración de las bases de datos, desde donde nos iremos centrando en cómo es la administración en MongoDB y en las características que posee a través de la realización de las pruebas presentadas en la siguiente sección.

La administración de las bases de datos es un proceso que debe estar siempre presente a la hora de trabajar con las mismas. Es el administrador de bases de datos (DBA) el que se encarga de realizar las tareas que abarca ese proceso, a este administrador podemos definirle como la persona que tiene la responsabilidad tanto de mantener como de operar las bases de datos del sistema en cuestión. Alguna de sus tareas concretas son la creación y la configuración de la base de datos, mantener la integridad y disponibilidad de los datos que esta contiene, garantizar su seguridad, diseñar planes de contingencia, entre otras muchas. Todos los tipos de bases de datos, bien sean relacionales o no relacionales, necesitan de esta clase de servicios para su correcto y óptimo funcionamiento, es decir, que utilicemos el tipo de base de datos que utilicemos siempre estará presenta la figura del administrador de bases de datos realizando tareas similares.

Cuando hablamos de administración en MongoDB también nos referimos a la tarea de gestionar y mantener el sistema y el desarrollo de dicho sistema. Y en este caso, podemos dividir la administración en tres grandes grupos, los cuales se pueden dividir en varios subgrupos como se muestra a continuación:

1. **Estrategias operativas:** Encontramos aquí todos aquellos términos relacionados tanto con las operaciones y el mantenimiento del sistema de MongoDB, incluyendo también las copias de seguridad, el mantenimiento necesario y la configuración.

    1.1. **Métodos de realización de copias de seguridad:** Es necesario disponer de una estrategia, que, en caso de pérdida de datos seamos capaz de reestablecerlos. MongoDB proporciona varios métodos para ello:

        1.1.1. **Servicio de gestión de MongoDB (MMS):** Esta herramienta soporta tanto la realización de copias como la restauración

de las mismas del despliegue del sistema. Este servicio MMS realiza copias de manera continua del sistema, y *MMS Backup* nos ofrece puntos de restauración de dicho sistema.

1.1.2. **Copia de ficheros de datos subyacentes:** La copia de seguridad se realiza a través de la copia de este tipo de ficheros. Se crean como capturas de diferentes momentos de los ficheros de datos.

1.1.3. **mongodump:** La función de esta herramienta es la de leer los datos de un sistema MongoDB y crear con ellos ficheros de formato BSON. Tras la creación de estos ficheros, la herramienta **mongorestore** es capaz de poblar la base de datos con el contenido de dichos ficheros.

1.2. **Monitorización en MongoDB:** esta tarea es indispensable en la administración. Para realizarla MongoDB nos proporciona tres estrategias diferentes:

1.2.1. A través de la utilización de utilidades que proporcionan informes de la actividad de la base de datos en tiempo real.

1.2.2. Utilizando comandos que nos devuelven las estadísticas correspondientes del sistema

1.2.3. También la herramienta MMS mencionada anteriormente toma datos válidos para llevar a cabo esta monitorización.

Algunas de las herramientas que nos ofrece MongoDB para realizar informes son por ejemplo **mongostat**, útil para comprender la distribución de los tipos de operaciones y el plan de capacidd, y **mongotop**, recomendable para comprobar si la actividad y el uso de la base de datos coincide con las previsiones.

1.3. **Configuración en tiempo de ejecución:** Para la configuración de MongoDB tenemos numerosas opciones como se muestra a continuación:

```
fork = true
bind_ip = 127.0.0.1
port = 27017
quiet = true
dbpath = /srv/mongodb
logpath = /var/log/mongodb/mongod.log
```

```
logappend = true
journal = true
```

1.3.1. **fork:** Con valor *true* habilita el demonio *mongod*, el cual permite correr la base de datos como un servidor común.

1.3.2. **bindIp:** De la manera configurada en el ejemplo escucha tan solo peticiones locales.

1.3.3. **port:** El puerto indicado es el puerto por defecto de las bases de datos MongoDB.

1.3.4. **quiet:** Cuando su valor es *true* su función es desactivar todo excepto las entradas más críticas en el fichero *output/log*. Para realizar tareas de diagnóstico o pruebas es recomendable que este parámetro tenga el valor *false*.

1.3.5. **dbpath:** Parámetro donde se especifíca la ruta donde se almacenan los ficheros de datos.

1.3.6. **logpath:** Ruta del fichero .log donde MongoDB escribirá su salida.

1.3.7. **logappend:** Al ser *true*, nos aseguramos de que el demonio *mongod* no sobreescriba los ficheros log ya existentes en la siguiente operación.

1.3.8. **journal:** Activa el registro.

2. **Gestión de datos:** Grupo en el que nos centramos en la gestión de los datos, su organización o su mantenimiento. Encontramos aquí tres pequeños subgrupos.

2.1. **Conocimiento del centro de datos**: MongoDB ofrece una serie de herramientas que nos va a permitir personalizar el comportamiento de los cluster de manera que pueda las operaciones se basen en la ubicación.

2.2. ***Capped Collections*:** Son colecciones de tamaño fijo que soportan operaciones de alto rendimiento, ya que insertan, recuperan y eliminan documentos en función del orden de inserción de los mismos. Para la creación de colecciones disponemos de la herramienta *createCollection()*.

2.3. **Expiración de datos de colecciones por TTL:** Es posible eliminar datos despues de un número específico de segundos o después de una hora determinada gracias a ello.

3. **Estrategias para la optimización:** Por último, como su propio nombre indica, aparecen aquí las técnicas que se utilizan para mejorar el rendimiento en una base de datos MongoDB. Debemos tener en cuenta que son muchos los factores que incluyen en el rendimiento de una base de datos, como por ejemplo el uso de índices, la estructura de las consultas, el modelo de datos o el diseño de la aplicación. Encontramos aquí varias técnicas relacionadas con la optimización:

   3.1. **Evaluación del rendimiento de las operaciones actuales:** Con MongoDB tenemos la capacidad de obtener características sobre el rendimiento de cada una de las operaciones realizadas sobre la base de datos, entre ellas encontramos por ejemplo:

      3.1.1. **db.currentOp():** Nos proporciona informes de las operaciones que están corriendo en una instancia de *mongod*.

      3.1.2. **explain():** En este caso obtenemos estadísticas sobre la consulta realizada e información sobre los índices utilizados.

   3.2. **Uso de *capped collections* para escrituras y lecturas rápidas:** Podemos definir las *capped collections* como colecciones cirulares de un tamaño fijo que mantienen documentos bien ordenados incluso sin la utilización de un índice, esto quiere decir que son capaces de soportar escrituras a gran velocidad y lecturas secuenciales. Un ejemplo para lo cual son útiles, son por ejemplo para la utilizarlos como logs, aunque no solo se utilizan únicamente para ello.

   3.3. **Optimización de consultas:** Encontramos en este caso numerosas opciones para la optimización de dichas consultas:

      3.3.1. **Creación de índices:** A través de la creación de índices se puede mejorar la eficiencia, cuando, por ejemplo se realizan varias consultas sobre uno o varios campos, ya que podemos tener índices sencillos o compuestos.

      3.3.2. **Limitar el número de resultados:** Se puede reducir el número de resultados que obtendremos con el método *limit()*, gracias al cual reduciremos la demanda de los recursos de la red.

3.3.3. **Utilización de proyecciones para obtener solo los datos necesarios:** Cuando solo necesitamos un pequeño conjunto de los datos de los documentos, es recomendable seleccionarlos, ya que si no obtendremos todos los campos de dicho documento, ralentizando la petición.

3.3.4. **Utilizar *$hint* para seleccionar un índice determinado:** El optimizador de consultas está diseñado para seleccionar el índice óptimo para realizar la operación que se determine, sin embargo, podemos forzar a MongoDB a que utilice un índice concreto para realizar nuestra consulta.

3.3.5. **Utilización del operador incrementar para mejorar el rendimiento del lado del servidor:** Es el operador *$inc* el encargado de bien incrementar o decrementar los valores de los documentos, la ventaja de la que dispone este operador es que lo realiza en el lado del servidor, con las ventajas que ello conlleva.

3.4. **Diseño:** Podemos tener en cuenta también otras opciones útiles cuando determinados el diseño del sistema, como por ejemplo:

3.4.1. **Consideraciones del esquema:**

3.4.1.1. **Esquema dinámico:** Recordamos que en MongoDB los datos poseen un esquema dinámico, es decir, no se fuerza a seguir una estructura en los documentos, aunque es posible que en ocasiones los documentos sigan una estructura bastante homogénea. Esto nos trae como ventaja tanto el desarrollo iterativo como el polimorfismo.

3.4.1.2. **Cadenas sensibles a mayúsculas y minúsculas:** MongoDB es un sistema sensible a la utilización de mayúsculas y minúsculas, algo a tener en cuenta a la hora de trabajar con ello.

3.4.1.3. **Campos sensibles al tipo:** Se ha mencionado anteriormente que estos sistemas almacenan los datos en formato BSON, el cual almacena no solo los datos, si no también información adicional sobre el tipo de los mismos.

3.4.2. **Consideraciones generales:**

3.4.2.1. **Por defecto, las actualizaciones afectan a un documento:** Para realizar la modificación de varios documentos

hay que tener en cuenta el valor del parámetro *multi*, ya que debe tener el valor *true* o 1.

3.4.2.2. **Limitación del tamaño del documento BSON:** Por defecto, el tamaño de estos documentos BSON es de 16MB, si se desa superar esta cantidad es necesaria la utilización de ***GridFS***, cuya función es la de dividir los ficheros en partes o trozos y almacenarlos como documentos separados. Dichos trozos tienen por defecto un tamaño de 255K, y la manera de almacenarlo es en dos colecciones, una utilizada para almacenar los trozos de ficheros y otra utilizada para almacenar los ficheros de metadatos.

3.4.2.3. **Transacciones que no están totalmente generalizadas:** Lo que permiten los sistemas MongoDB es la modificación de documentos a través de la utilización de una sola operación atómica, lo que hace que cubdra la mayor parte del uso de transacciones de otros sistemas, pero no todos.

3.4.3. **Consideraciones del conjunto de réplicas:**

3.4.3.1. **Utilizar un número de miembros impar en el conjunto:** Debido a que se realizan elecciones por consenso en estos cluster, para asegurarnos de que las decisiones se tomen de manera satisfactoria es importante que se cuente con un número impar de votos, ya que de esta manera siempre se obtendrá una respuesta, no es posible encontrarnos con un ”empate”.

3.4.3.2. **Mantener a los miembros actualizados hasta la fecha:** Un motivo por el cual es importante que todas las réplicas se encuentres actualizadas es debido a que los cluster de MongoDB soporta la conmuntación automática en caso de error, es decir, si un nodo tiene un fallo el sistema podrá seguir funcionando correctamente sin él.

3.4.4. **Consideraciones sobre el *sharding:*** Ya se ha hablado anteriormente del *sharding*, es lo que se ha venido llamando escalabilidad horizontal. Es conveniente tener en cuenta varios factores al trabajar con ello, como por ejemplo que los valores de las claves del *shard* son inmutables o que es recomendable elegir las mismas cuidadosamente, ya que no se puede elegir una nueva

clave para una colección que ya se encuentra *sharded*. Junto con ello, cuando se habilita este *sharding* en una colección existente, MongoDB se asegura de que dicha colección tenga el máximo tamaño posible para asegurarse de que se pueden crear trozos.

4.2. Preparación del entorno

El objetivo aquí es el de realizar las pruebas en escenarios comunes que nos podemos encontrar cuando trabajamos con MongoDB, los cuales les podemos dividir principalmente en dos entornos, aquel que se denomina *replica set* y el *sharding*. A continuación se explica cómo preparar dichos entornos, los cuales se utilizarán para diferentes conjuntos de pruebas, mientras que el replica set estará destinado a comprender más a fondo su comportamiento ante varias situaciones, en el sharding vamos a centrarnos más en el rendimiento, aunque también se tendrá en cuenta su comportamiento.

Tanto el replica set como el sharding van a funcionar sobre una sola máquina. Cada instancia de mongo se iniciará en un terminal diferente, simulando así los nodos que forman cada escenario.

4.2.1. Replica Set

En primer lugar, cuando hablamos de **replica set**, nos referimos a un conjunto de nodos en los cuales encontramos un maestro (instancia primaria) y uno o varios esclavos (instancias secundarias) donde se replicarán los datos. El objetivo aquí es proporcionar robustez al sistema, aumentar su disponibilidad y mejorar su integridad. En este caso es el maestro el encargado de llevar a cabo las operaciones de escritura y lectura dentro del sistema, aunque los esclavos también pueden ser configurados para realizar estas lecturas. La característica principal de este tipo de sistemas es la capacidad de recuperación que poseen, ya que si un nodo maestro falla otro nodo le sustuirá, de manera que dicho sistema podrá seguir en funcionamiento. La siguiente ilustración muestra de manera mas clara y sencilla cómo funciona.

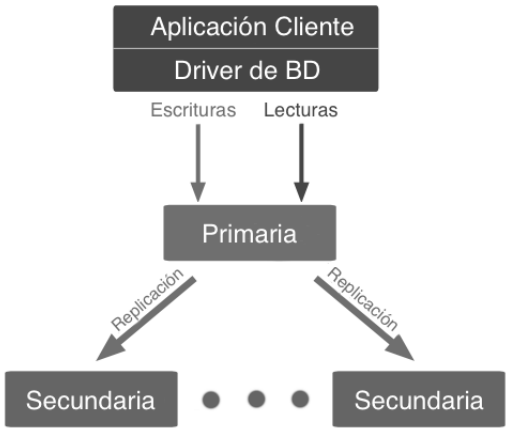

Figura 4.1: Replica Set

Encontramos también aquí la figura del árbitro, son nodos que no disponen de una copia de los datos, por lo que estos no pueden ocupar el lugar de un nodo maestro, su fin es el de ayudar a decidir quién será el siguiente nodo maestro, es relevante sobre todo cuando el número de nodos del que disponemos es un número par.

De cara a la administración este tipo de sistemas es importante, ya que la realización de backups, mantener el sistema disponible o garantizar la integridad de los datos es algo indispensable a la hora de trabajar como administrador de base de datos. Por este motivo en primer lugar montaremos un sistema con un nodo maestro y tres esclavos. Junto con ello se detallará la manera de hacer que uno de ellos ejerza el papel de árbitro, necesario para la realización de otras pruebas. Los pasos a seguir son los siguientes (se recuerda que se realizan en un entorno Linux):

1. En primer lugar se han de crear los directorios correspondientes donde cada nodo almacenará su información.

```
mkdir dataNodo1 dataNodo2 dataNodo3 dataNodo4
```

2. A continuación iniciamos cuatro instancias de mongod, las cuales se corresponden con cada uno de los nodos, indicando tanto la carpeta que le

corresponde, como el puerto donde escuchará y el nombre del conjunto.

```
mongod --dbpath dataNodo1 --port 27020 --replSet repProy
mongod --dbpath dataNodo2 --port 27021 --replSet repProy
mongod --dbpath dataNodo3 --port 27022 --replSet repProy
mongod --dbpath dataNodo4 --port 27023 --replSet repProy
```

3. El siguiente paso es abrir un nodo conectándonos a través de "mongo" e inicializar la réplica desde uno de ellos (en este caso el correspondiente al puerto 27020) utilizando el comando:

```
mongo --host natalia-K55A --port 27020
>rs.initiate({ "_id": "repProy", "members" :[
{ "_id" : 0, host : "natalia-K55A:27020"},
{ "_id" : 1, host : "natalia-K55A:27021"},
{ "_id" : 2, host : "natalia-K55A:27022"},
{ "_id" : 3, host : "natalia-K55A:27023"}]})
```

   En caso de querer que uno de los nodos ejerza el papel de árbitro, basta con añadir *arbiterOnly : true* en el nodo correspondiente, será utilizado posteriormente.

Una vez finalizados estos pasos dispondremos del conjunto de nodos deseados. Las operaciones se realizarán a través del nodo maestro, para poder realizar operaciones de lectura a través de cualquier nodo esclavo necesitamos ejecutar

```
rs.slaveOk()
```

Con esto ya disponemos de nuestro replica set.

4.2.2. Sharding

En segundo lugar, nos encontramos con el **sharding**, donde encontramos un tipo de escenario en el que se encuentra presente la escalabilidad horizontal, característica ya mencionada en numerosas ocasiones. La siguiente ilustración

muestra de manera gráfica dicho escenario.

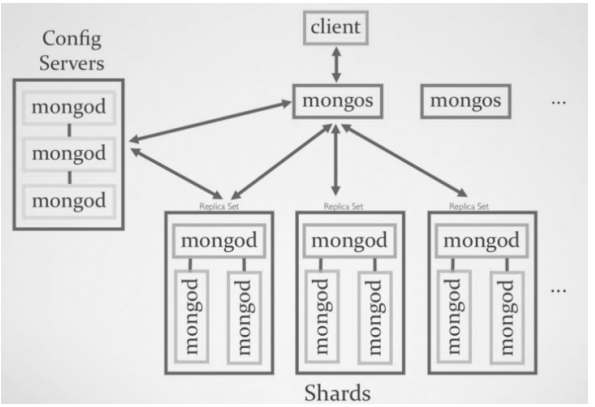

Figura 4.2: Sharding

Un shard puede estar compuesto bien por un replica set, o por una instancia de mongod. En nuestro caso vamos a optar por utilizar tres instancias de mongod, es decir, tres shards, ya que utilizar varios replica set complicaría tanto la comprensión del funcionamiento del mismo, como la toma de datos de los distintos parámetros. Al contrario que en el caso anterior, cada instancia almacenará datos, pero la manera de acceder a ellos, bien a través de operaciones de lectura o de operaciones de escritura, es a través de lo que se llama una instancia (puede haber más de una) *mongos*. En el caso de conectarnos directamente a uno de los shards que lo componen, tan solo se obtendrán los valores que éste almacena.

Para llevar a cabo la configuración de este escenario se han seguido los siguientes pasos:

1. Inicialmente se van a crear los directorios tanto de los tres shards, como un directorio para el servidor de configuración. Este servidor será el que almacenará los metadatos del cluster. Aunque es común utilizar uno para cada shard, en este caso, al estar ante un escenario de prueba basta con mantener tan solo una instancia del mismo.

```
mkdir dataShard1 dataShard2 dataShard3 dataConfig
```

2. A continuación hemos de iniciar todos los servicios necesarios, véase:

```
Servidor de configuración:
 mongod --configsvr --port 27020 --dbpath dataConfig
Instancia de mongos:
 mongos --configdb natalia-K55A:27020 --chunkSize 1
 --port 27017
Shard1:
 mongod --shardsvr --dbpath dataShard1 --port 27021
Shard2:
 mongod --shardsvr --dbpath dataShard2 --port 27022
Shard3:
 mongod --shardsvr --dbpath dataShard3 --port 27023
```

3. El siguiente paso es añadir los shards a través de mongos, para ello nos conectamos a mongos y posteriormente ejecutamos el comando addshard. Es importante utilizar la colección **admin**, ya que es donde se encuentra la configuración del shard.

```
mongo natalia-K55A:27017/admin
>db.runCommand({addshard:"natalia-K55A:27021",
 allowLocal:true})
>db.runCommand({addshard:"natalia-K55A:27022",
 allowLocal:true})
>db.runCommand({addshard:"natalia-K55A:27023",
 allowLocal:true})
```

4. Por último hemos de habilitar el sharding tanto en la base de datos como en la colección que se va a utilizar, e indicar el campo por el cual se realizará la división, en este caso se realizará, por ejemplo, por el campo1.

```
>db.runCommand ({enablesharding: "proyecto"})
>db.runCommand ({shardcollection: "proyecto.shard",
 key : {campo1: 1}})
```

Con todo ello ya tendremos nuestro *sharding* en funcionamiento.

4.3. Experimentando con MongoDB

A lo largo de la sección 4.1 se han visto numerosas características propias de la administración dentro de MongoDB, el objetivo aquí es revisar algunas de esas características mencionadas y observar el comportamiento que tiene este tipo de bases de datos ante determinadas situaciones, para, posteriormente evaluar los resultados obtenidos y comprender con qué estamos trabajando y en qué ocasiones puede ser útil su implementación.

Uno de los puntos importantes que encontramos cuando trabajamos con bases de datos es el diseño y tamaño de los datos, en este caso distribuidos en colecciones y documentos, que insertamos en dicha base, ya que puede llegar a influir fuertemente en el tiempo que se dedica tanto a la inserción, a la modificación o incluso a la eliminación de los mismos. Para analizar cómo influye el tamaño de los documentos a la hora de realizar todas estas operaciones, se han realizado varios ficheros con datos de prueba, creados a través de la utilización de scripts, que contienen desde 1.000 registros a 1 millón, y varían también en el número de campos que incluyen, desde 10 campos hasta la cantidad de 1.000. Estos campos, debido a la gran cantidad de datos necesarios se han creado de manera aleatoria, por lo que para la realización de este benchmark se les han asignado cadenas alfanuméricas de ocho caracteres, exceptuando uno de los campos, el cual será numérico para ayudar a realizar las pruebas de modificación. Tras esto ha sido necesario realizar modificaciones en los ficheros con el fin de hacer que cumplan una serie de criterios para llevar a cabo las modificaciones que deseamos, uno de estos criterios es desordenar el fichero de manera que los datos numéricos con las mismas características se encuentren en diferentes puntos del ficheros, para lo cual se ha utilizado la herramienta mongoexport, la cual, gracias a la sentencia "orderby", obtiene como salida un fichero ordenado, en este caso, por el campo1.

Al avanzar en el documento se echará en falta la realización de pruebas específicas para la búsqueda de datos, la cual se ha decidido no realizar porque se encuentra dentro de las modificaciones, es decir, para modificar datos hay que realizar una búsqueda en la colección según los criterios indicados.

Cabe destacar las condiciones en las que se van a realizar estas pruebas, ya que en el segundo conjunto de pruebas, donde nos centramos más en el comportamiento que en el rendimiento se utilizará un entorno gráfico, el cual se eliminará a la hora realizar las pruebas que corresponden a la inserción, a la modificación y al borrado. Además, los tiempos se han tomado utilizando

el comando *time* de Linux y un script en python, es decir, se ha utilizado pymongo, el cual se puede instalar facilmente en nuestro Ubuntu utilizando el comando *sudo apt-get install python-setuptools* y luego importando pymongo en el script correspondiente.

Se recuerdan de nuevo las características de la máquina en la cual se realiza este conjunto de pruebas:

```
Procesador Intel Core i7-3630QM a 2.40GHz x64
RAM 8GB
Sistema operativo Ubuntu 14.04 LTS
```

4.3.1. Inserción de datos

La inserción masiva de datos puede ser un punto a tener en cuenta cuando hablamos de bases de datos de tipos NoSQL. Por supuesto, influye el número de documentos que insertemos dentro de la colección deseada, pero hay otras características que merece la pena estudiar y comprobar cómo afectan, como por ejemplo, el número de campos que hay dentro de dicho documento, el cual es claro que influirá en el tiempo de inserción haciendo que esto aumente, ¿pero de qué manera?. Se ha de tener en cuenta también que en este caso se produce lo que hemos llamado anteriormente *sharding* o escalabilidad horizontal entre los nodos que tenemos disponibles.

Número de documentos	Número de campos	Real	User	Sys
1.000	10	0m0.101s	0m0.029s	0m0.018s
	100	0m0.167s	0m0.099s	0m0.017s
	1.000	0m1.778s	0m0.653s	0m0.011s
10.000	10	0m0.439s	0m0.119s	0m0.054s
	100	0m1.813s	0m0.778s	0m0.068s
	1.000	0m19.750s	0m.6330s	0m0.203s
100.000	10	0m3.846s	0m1.036s	0m0.152s
	100	0m17.356s	0m7.034s	0m0.055s
	1.000	3m46.339s	1m3.258s	0m1.876s
1.000.000	10	0m49.603	0m10.318s	0m0.875s
	100	3m59.496s	1m12.758s	0m5.348s

Tabla 4.1: Pruebas - Inserción de datos

A continuación, se representan de manera gráfica los datos que se han obtenido tras realizar las inserciones correspondientes. Se muestran tanto el tiempo de usuario como el tiempo real, ya que son los que se han considerado más relevantes.

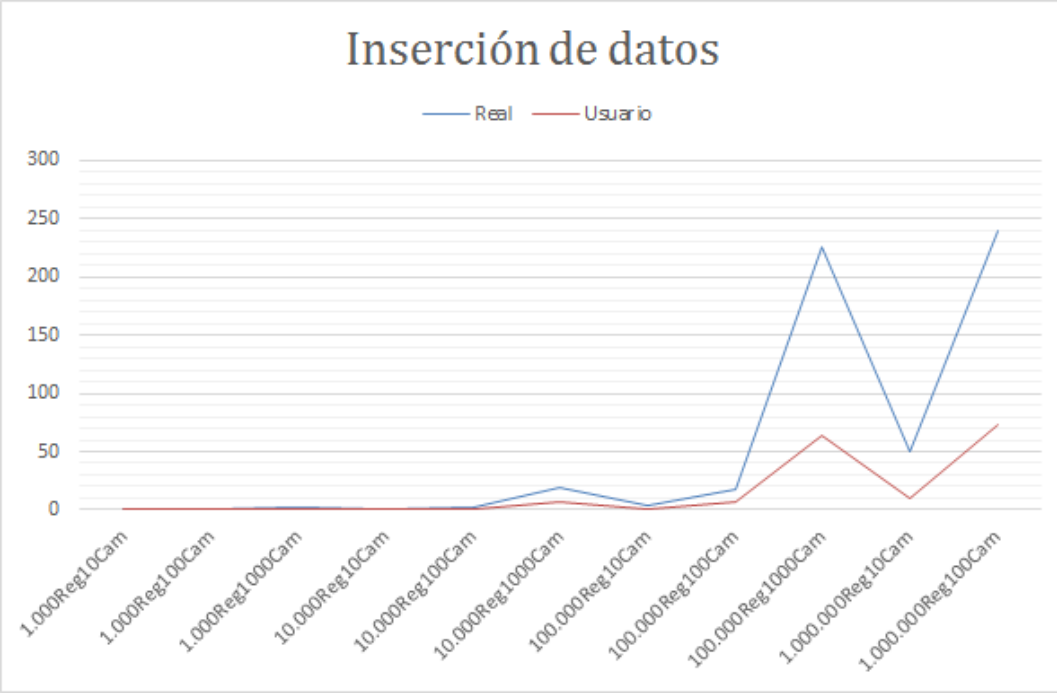

Figura 4.3: Gráfica Inserción de Datos

Se puede ver claramente la tendencia que siguen ambas gráficas a medida que aumentamos en el número de documentos. Aunque inicialmente, con ficheros de hasta 10.000 registros los tiempos tomados son muy similares, se puede ver que a partir de esta cifra influye no solo el número de registros, si no sobre todo el número de campos que dicho campo posee, los cuales hacen que el tiempo aumente considerablemente. El ejemplo más claro de esta influencia de los campos en el aumento tiempo, lo podemos observar claramente en la poca diferencia que existe en los tiempos obtenidos a la hora de realizar la inserción de 100.000 documentos con 1.000 campos y la inserción de 1.000.000

de registros y 100 campos.

Por supuesto, analizando más en profundidad, se puede observar que en el primero de los casos un documento posee mil campos, es decir, mil pares clave-valor, lo que en total supone cien millones de pares en total, lo mismo que en el segundo caso con un millón de documentos y cien campos. Lo que nos lleva a pensar por lo tanto, que no es tanto la estructura del documento lo que importa, si no el conjunto total de pares que se insertan en la base de datos.

4.3.2. Modificación de datos

En cuanto a la modificación de datos se van a tener en cuenta, además de las características ya mencionadas en la inserción de datos, como son el número de documentos y el número de campos dentro de ellos, aparece también el tipo de modificación que se realice, es decir, si realizamos la modificación en un rango de datos o tan solo en campos que cumplen una condición de igualdad, y la utilización o no de índices y el tipo de índices que se utilicen.

Cabe destacar dentro de la modificación la importancia de usar la sentencia **safe: true** dentro del comando update junto con la opción **multi: true**.

Esta es la manera que tenemos de asegurar que el tiempo que muestra time es el que se ha obtenido al finalizar dicha actualización.

Las modificaciones se van a realizar según dos criterios, un criterio de igualdad, en el cual se buscarán los documentos cuyo campo3 tenga el valor 45, y se hará también una busqueda por rango, en el cual se buscarán aquellos documentos que tengan en el campo3 un valor mayor que 50. Los ficheros poseen un 25 % de documentos que cumplen estas características, por ejemplo, en el fichero de mil registros hay un total de 2500 documentos con un valor de 45 y 2500 documentos con valores mayores que 50. También es necesario destacar que han sido tres los campos actualizados en cada uno de ellos, el campo1, el campo5 y el campo10.

Junto con ello, encontramos otra característica a destacar, la elección de la *shard key*, la cual en este caso ha sido el campo2, elegido debido a que no se actualiza ni se utiliza como criterio de modificación, lo que hace que en el caso en el que no indicamos ningún índice, MongoDB utilice como dicho índice el campo _id y el campo2. Por otro lado, cuando le indicamos que queremos como índice el campo3 estamos utilizando los tres campos.

Número de documentos	Número de campos	Tipo búsqueda	Índices utilizados	Real	User	Sys
1.000	10	Rango	_id, campo2	0m0.059s	0m0.044s	0m0.012s
			campo3	0m0.066s	0m0.048s	0m0.013s
		Igualdad	_id, campo2	0m0.067s	0m0.049s	0m0.013s
			campo3	0m0.075s	0m0.055s	0m0.014s
	100	Rango	_id, campo2	0m0.071s	0m0.047s	0m0.012s
			campo3	0m0.112s	0m0.049s	0m0.013s
		Igualdad	_id, campo2	0m0.078s	0m0.051s	0m0.013s
			campo3	0m0.084s	0m0.054s	0m0.014s
	1.000	Rango	_id, campo2	0m0.245s	0m0.048s	0m0.013s
			campo3	0m0.207s	0m0.053s	0m0.014s
		Igualdad	_id, campo2	0m0.273s	0m0.047s	0m0.012s
			campo3	0m0.209s	0m0.050s	0m0.013s
10.000	10	Rango	_id, campo2	0m0.088s	0m0.054s	0m0.014s
			campo3	0m0.789s	0m0.049s	0m0.013s
		Igualdad	_id, campo2	0m0.081s	0m0.049s	0m0.013s
			campo3	0m0.091s	0m0.053s	0m0.013s

Número de documentos	Número de campos	Tipo búsqueda	Índices utilizados	Real	User	Sys
10.000	100	Rango	_id, campo2	0m0.227s	0m0.051s	0m0.013s
			campo3	0m0.253s	0m0.049s	0m0.013s
		Igualdad	_id, campo2	0m0.197s	0m0.049s	0m0.013s
			campo3	0m0.165s	0m0.048s	0m0.012s
	1.000	Rango	_id, campo2	0m1.808s	0m0.051s	0m0.013s
			campo3	0m2.032s	0m0.047s	0m0.012s
		Igualdad	_id, campo2	0m1.693s	0m0.051s	0m0.013s
			campo3	0m1.441s	0m0.053s	0m0.013s
100.000	10	Rango	_id, campo2	0m0.278s	0m0.046s	0m0.012s
			campo3	0m0.412s	0m0.050s	0m0.013s
		Igualdad	_id, campo2	0m0.270s	0m0.046s	0m0.012s
			campo3	0m0.337s	0m0.050s	0m0.013s
	100	Rango	_id, campo2	0m1.635s	0m0.045s	0m0.012s
			campo3	0m4.910s	0m0.051s	0m0.012s
		Igualdad	_id, campo2	0m3.803s	0m0.052s	0m0.013s
			campo3	0m1.358s	0m0.048s	0m0.013s
	1.000	Rango	_id, campo2	0m42.361s	0m0.060s	0m0.019s
			campo3	0m53.244s	0m0.053s	0m0.019s
		Igualdad	_id, campo2	0m46.938s	0m0.052s	0m0.019s
			campo3	0m39.591s	0m0.058s	0m0.017s
1.000.000	10	Rango	_id, campo2	0m2.467s	0m0.045s	0m0.012s
			campo3	0m2.352s	0m0.056s	0m0.015s
		Igualdad	_id, campo2	0m3.022s	0m0.050s	0m0.013s
			campo3	0m2.216s	0m0.053s	0m0.013s
	100	Rango	_id, campo2	0m54.266s	0m0.042s	0m0.011s
			campo3	1m43.688s	0m0.060s	0m0.029s
		Igualdad	_id, campo2	0m49.317s	0m0.049s	0m0.012s
			campo3	0m52.781s	0m0.056s	0m0.016s

Tabla 4.3: Pruebas - Modificación de datos

En cuanto a la modificación obtenemos estas gráficas con los datos anteriores. En este caso lo más relevante es la comparación de tiempos obtenidos entre las diferentes situaciones en las que nos encontramos, por lo que tan solo se ha tenido en cuenta el tiempo real a la hora de crear las gráficas, que es el que verdaderamente sufre las variaciones.

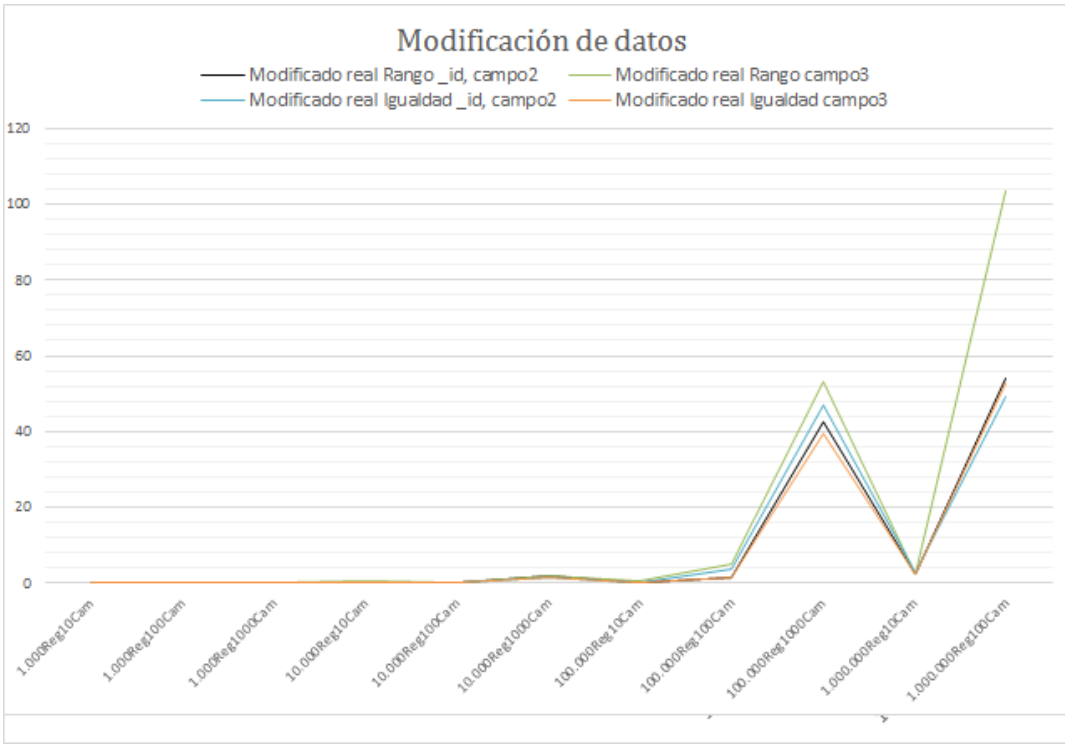

Figura 4.4: Gráfica Modificación de Datos

Cabe destacar varias características de las gráficas obtenidas. En primer lugar nos encontramos con que la forma de cada una de ellas es similar, al igual que en el caso de la inserción, influye el número de campos en dos puntos críticos, en el caso de cien mil documentos y en el de un millón, sin embargo, aunque en la mayoría de los casos se puede obtener la misma conclusión que en dicha inserción, si observamos la gráfica de la modificación según el rango, y utilizando además como índices tanto el _id, como el campo2, como el campo3,

observamos que el tiempo se dispara, por lo que es el único caso en el cual no influye tanto el número de pares clave-valor, como el número de documentos.

Por otro lado, llama la atención que, al contrario de lo que cabe pensar, la utilización del campo3 como índice penaliza en tiempo a la hora de realizar la modificación según el rango. Esta diferencia es clara en los dos grandes picos que se muestran en la gráfica. Además, la utilización de este índice tampoco obtiene grandes resultados a la hora de utilizarlo con el criterio de igualdad, ya que aunque en el caso de los cien mil documentos el tiempo es inferior, posteriormente, al utilizarlo con un millón de documentos, es el caso en el que no se usa dicho campo como índice el que obtiene mejor tiempo.

Finalmente se observan las diferencias que se han obtenido a la hora de utilizar como criterio de modificación el rango y la igualdad. En el primero de los casos, cuando no utilizamos como índice el campo3, es la igualdad la que necesita un mayor tiempo para realizar las modificaciones en el caso de los cien mil registros, pero esta diferencia cambia a la hora de trabajar con un millón. Por otro lado, al utilizar el campo3 como índice, es, como cabía esperar y no ha pasado en el caso anterior, la modificación por rango la que obtiene mayores tiempos en ambos casos.

Se puede concluir por lo tanto de estas modificaciones que MongoDB no se comporta en todos los casos como cabe esperar, comunmente, cuando trabajamos con índices en bases de datos relacionales, estos tiempos, en la gran mayoría de los casos, suelen disminuir, mientras que, como se ha visto, aquí incluso los tiempos se han visto penalizados.

4.3.3. Borrado de datos

Por último, la eliminación de los datos es similar a la inserción. En este caso se seguirán las mismas pruebas ya presentadas en dicha sección anterior.

Número de documentos	Número de campos	Real	User	Sys
1.000	10	0m0.056s	0m0.045s	0m0.011s
	100	0m0.056s	0m0.045s	0m0.011s
	1.000	0m0.057s	0m0.046s	0m0.011s
10.000	10	0m0.058s	0m0.046s	0m0.011s
	100	0m0.061s	0m0.049s	0m0.012s
	1.000	0m0.064s	0m0.051s	0m0.013s
100.000	10	0m0.060s	0m0.046s	0m0.010s
	100	0m0.060s	0m0.048s	0m0.012s
	1.000	0m0.106s	0m0.052s	0m0.013s
1.000.000	10	0m0.066s	0m0.054s	0m0.013s
	100	0m0.180s	0m0.049s	0m0.016s

Tabla 4.4: Pruebas - Borrado de datos

Encontramos en este caso en la siguiente ilustración los tiempos obtenidos en el borrado de datos.

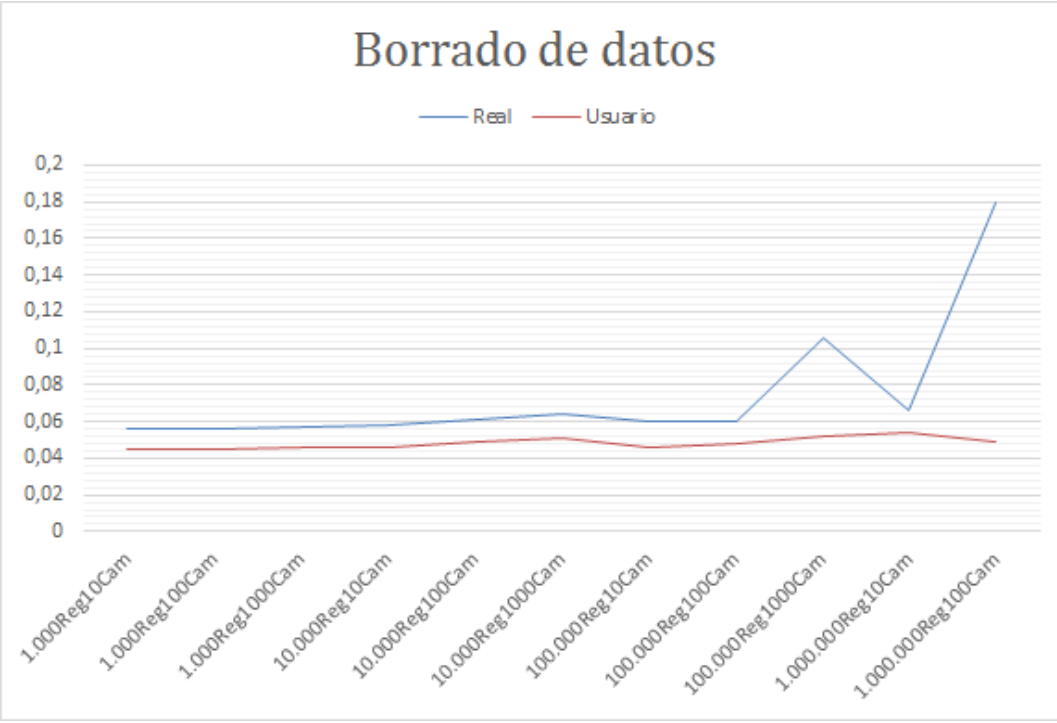

Figura 4.5: Gráfica Borrado de Datos

Se puede observar en la imagen anterior la importancia que tiene el número de campos a la hora de eliminar documentos a partir de, en este caso, 100.000 documentos. Hasta llegar a esta cantidad, tanto el tiempo real como el tiempo de usuario se mantienen constantes en tiempos alrededor de 0.06 segundos, un tiempo muy bajo para la gran cantidad de datos que se están manejando.

Es a partir de este número de registros cuando se pueden destacar varias características que presenta la curva, en primer lugar los tiempos de usuario y los tiempos reales difieren, quedándose el primero de ellos prácticamente constante durante toda la gráfica, mientras que el segundo presenta varios picos. Son precisamente estos picos lo que nos lleva a pensar en la importancia del número de campos, ya que después del primer pico, al eliminar un millón de

documentos con un total de diez campos, la curva vuelve a situarse en un tiempo similar a los iniciales.

Por otro lado, esta variación de tiempo en los campos no responde de la misma manera que lo hacía en la inserción, es decir, anteriormente, en la inserción, cuando teníamos el mismo número de pares clave-valor el tiempo obtenido era similar, en cambio, aquí la diferencia es relativamente grande si tenemos en cuenta los tiempos en los que nos movemos, ya que se ha pasado de un tiempo de 0.106 segundos a 0.180 segundos. A pesar de ser tan solo una diferencia de 0.074 segundos, hemos de tener en cuenta que este tiempo es mayor a cualquier tiempo de borrado obtenido anteriormente.

4.3.4. Otras pruebas de interés

Caso 1 - Replica Set con ausencia de árbitro Cuando nos encontramos en un escenario Maestro-Esclavo encontramos también, de manera opcional, la presencia del árbitro, el cual se encarga de ayudar a decidir quién será el siguiente nodo maestro en caso de que el primero falle, pero, si no disponemos de dicha figura ¿cómo se realiza ese trabajo?. Vamos a encontrar dos situaciones diferentes en este caso, en la primera de ellas vamos a disponer de un maestro y de un número de esclavos par, mientras que en el segundo tendremos un maestro y un número de esclavos impar. Ésta es una característica importante, ya que no se recomienda que el número de nodos sea par, ya que el esfuerzo a la hora de tomar decisiones es mayor.

- En primer lugar montamos un escenario similar al presentado en la sección anterior, pero en vez de utilizar cuatro nodos se utilizarán tres, uno de ellos será el maestro y los otros dos serán los esclavos. El resultado al ejecutar el comando *rs.status()* nos proporciona la siguiente salida:

```
> rs.status
function () { return db._adminCommand("replSetGetStatus"); }
repProy:PRIMARY> rs.status()
{
"set" : "repProy",
"date" : ISODate("2014-06-15T19:47:00Z"),
"myState" : 1,
"members" : [
{
"_id" : 0,
"name" : "natalia-K55A:27020",
"health" : 1,
"state" : 1,
"stateStr" : "PRIMARY",
"uptime" : 565,
"optime" : Timestamp(1402861552, 1),
"optimeDate" : ISODate("2014-06-15T19:45:52Z"),
"self" : true
},
{
```

```
"_id" : 1,
"name" : "natalia-K55A:27021",
"health" : 1,
"state" : 2,
"stateStr" : "SECONDARY",
"uptime" : 65,
"optime" : Timestamp(1402861552, 1),
"optimeDate" : ISODate("2014-06-15T19:45:52Z"),
"lastHeartbeat" : ISODate("2014-06-15T19:46:59Z"),
"lastHeartbeatRecv" : ISODate("2014-06-15T19:47:00Z"),
"pingMs" : 0,
"syncingTo" : "natalia-K55A:27020"
},
{
"_id" : 2,
"name" : "natalia-K55A:27022",
"health" : 1,
"state" : 2,
"stateStr" : "SECONDARY",
"uptime" : 63,
"optime" : Timestamp(1402861552, 1),
"optimeDate" : ISODate("2014-06-15T19:45:52Z"),
"lastHeartbeat" : ISODate("2014-06-15T19:46:59Z"),
"lastHeartbeatRecv" : ISODate("2014-06-15T19:47:00Z"),
"pingMs" : 0,
"syncingTo" : "natalia-K55A:27020"
}
],
"ok" : 1
}
```


Con esta información, la cual también se puede obtener a través de la información que nos proporcionan las instancias de mongod, obtenemos datos acerca de quien es el nodo maestro, y desde él insertamos en la base de datos proyecto, en la colección "prueba1" el fichero utilizado en el benchmark de MySQL vs. MongoDB, con el cual realizaremos esta

prueba.

Sin proporcionar permisos de lectura a los nodos esclavos, ya que no se considera relevante en este caso, se cierra la instancia de mongod que corresponde con el nodo maestro, es decir, la que está escuchando en el puerto 27020, y observamos el comportamiento del resto del sistema a través de las instancias mongod de los nodos esclavos, en ambos casos la salida es muy similar como se puede observar a continuación. La siguiente salida ha sido filtrada eliminando líneas que no aporten información relevante.

```
Puerto 27021:
 22:27:14.141 [rsBackgroundSync] replSet sync source
 problem: 10278 dbclient error communicating with server:
 natalia-K55A:27020
 22:27:14.141 [rsBackgroundSync] replSet syncing to:
 natalia-K55A:27020
 22:27:14.142 [rsBackgroundSync] repl: couldn't connect
 to server natalia-K55A:27020
 22:27:14.738 [rsHealthPoll] DBClientCursor::init call()
 failed
 22:27:14.739 [rsHealthPoll] replset info
 natalia-K55A:27020 heartbeat failed, retrying
 22:27:14.739 [rsHealthPoll] replSet info
 natalia-K55A:27020 is down (or slow to respond):
 22:27:14.739 [rsHealthPoll] replSet member
 natalia-K55A:27020 is now in state DOWN
 22:27:14.740 [rsMgr] not electing self, natalia-K55A:27022
 would veto with 'natalia-K55A:27021 is trying to elect
 itself but natalia-K55A:27020 is already primary and more
 up-to-date'
 22:27:15.971 [conn173] replSet info voting yea for
 natalia-K55A:27022 (2)
 22:27:16.742 [rsHealthPoll] replSet member
 natalia-K55A:27022 is now in state PRIMARY
 22:27:17.142 [rsBackgroundSync] replSet syncing to:
```

```
 natalia-K55A:27022

Puerto 27022:
 22:27:14.141 [rsBackgroundSync] replSet sync source
 problem: 10278 dbclient error communicating with server:
 natalia-K55A:27020
 22:27:14.141 [rsBackgroundSync] replSet syncing to:
 natalia-K55A:27020
 22:27:14.142 [rsBackgroundSync] repl: couldn't connect
 to server natalia-K55A:27020
 22:27:15.057 [rsHealthPoll] DBClientCursor::init call()
 failed
 22:27:15.057 [rsHealthPoll] replset info
 natalia-K55A:27020 heartbeat failed, retrying
 22:27:15.058 [rsHealthPoll] replSet info
 natalia-K55A:27020 is down (or slow to respond):
 22:27:15.058 [rsHealthPoll] replSet member
 natalia-K55A:27020 is now in state DOWN
 22:27:15.970 [rsMgr] replSet info electSelf 2
 22:27:16.142 [rsMgr] replSet PRIMARY
```

Es a la hora de decidir quién será el nuevo nodo maestro donde la información proporcionada por estos nodos cambia, ya que mientras en el esclavo 1 (puerto 27021) vota por el esclavo 2, en este último ya se informa de que ha sido seleccionado y pasa a ser el nuevo maestro. Si nos volvemos a conectar a los dos nodos que nos quedan vemos que ahora podemos acceder a los datos insertados anteriormente desde el nodo que escuchaba en el puerto 27022, y que la estructura del Replica Set queda de la siguiente manera:

```
repProy:PRIMARY> rs.status()
{
"set" : "repProy",
"date" : ISODate("2014-06-15T20:51:40Z"),
"myState" : 1,
"members" : [
```

```
{
"_id" : 0,
"name" : "natalia-K55A:27020",
"health" : 0,
"state" : 8,
"stateStr" : "(not reachable/healthy)",
"uptime" : 0,
"optime" : Timestamp(1402863640, 12344),
"optimeDate" : ISODate("2014-06-15T20:20:40Z"),
"lastHeartbeat" : ISODate("2014-06-15T20:51:38Z"),
"lastHeartbeatRecv" : ISODate("2014-06-15T20:27:12Z"),
"pingMs" : 0
},
{
"_id" : 1,
"name" : "natalia-K55A:27021",
"health" : 1,
"state" : 2,
"stateStr" : "SECONDARY",
"uptime" : 3932,
"optime" : Timestamp(1402863640, 12344),
"optimeDate" : ISODate("2014-06-15T20:20:40Z"),
"lastHeartbeat" : ISODate("2014-06-15T20:51:39Z"),
"lastHeartbeatRecv" : ISODate("2014-06-15T20:51:39Z"),
"pingMs" : 0,
"syncingTo" : "natalia-K55A:27022"
},
{
"_id" : 2,
"name" : "natalia-K55A:27022",
"health" : 1,
"state" : 1,
"stateStr" : "PRIMARY",
"uptime" : 4372,
"optime" : Timestamp(1402863640, 12344),
"optimeDate" : ISODate("2014-06-15T20:20:40Z"),
"self" : true
```

```
}
],
"ok" : 1
}
```

El seguir encontrándose el anterior nodo maestro registrado en el conjunto es lo que hace que los otros dos nodos lancen mensajes de error continuamente al no encontrar dicho nodo.

Si nos centramos ahora en el rendimiento de la elección del nuevo nodo maestro en este escenario, podemos ver que la caída del nodo se ha producido en el instante *22:27:14.141*, y la decisión de que fuera el segundo esclavo el nuevo maestro se ha confirmado en el instante *22:27:16.142* (tiempos del nuevo nodo maestro), es decir, la decisión se ha tomado en apenas dos segundos, aunque la decisión al nodo que escucha en el puerto 27021 le ha llegado en el instante *22:27:16.742*.

- En segundo lugar vamos a utilizar el escenario ya explicado en la preparación del entorno, es decir, dispondremos de un nodo maestro y de tres esclavos. La información que obtenemos al ejecutar el comando *rs.status()* será muy similar a la presentada en el caso anterior, por lo que se comenta directamente los resultados obtenidos. Cabe destacar que el nodo maestro es también aquel nodo que escucha desde el puerto 27020 al igual que en el caso anterior.

```
Puerto 27021:
 23:19:58.232 [rsMgr] not electing self, natalia-K55A:27023
 would veto with 'natalia-K55A:27021 is trying to elect
 itself but natalia-K55A:27020 is already primary and more
 up-to-date'
 23:19:59.340 [conn26] replSet info voting yea for
 natalia-K55A:27022 (2)
 23:20:00.232 [rsHealthPoll] replSet member
 natalia-K55A:27022 is now in state PRIMARY

Puerto 27022:
 23:19:57.615 [rsBackgroundSync] replSet sync source
 problem: 10278 dbclient error communicating with server:
```

```
 natalia-K55A:27020
 23:19:59.340 [rsMgr] replSet info electSelf 2
 23:19:59.620 [rsMgr] replSet PRIMARY

Puerto 27023:
 23:19:58.621 [rsMgr] not electing self, natalia-K55A:27022
 would veto with 'natalia-K55A:27023 is trying to elect
 itself but natalia-K55A:27020 is already primary and more
 up-to-date'
 23:19:59.341 [conn29] replSet info voting yea for
 natalia-K55A:27022 (2)
 23:20:00.620 [rsHealthPoll] replSet member
 natalia-K55A:27022 is now in state PRIMARY
```

De nuevo en este caso, el nuevo nodo maestro ha sido aquel que escucha a través del puerto 27022. Analizando las salidas obtenidas en las instancias mongod del resto de nodos se puede observar que son similares a las del caso anterior, ambos votan al mismo nodo, el del puerto 27022. En cuanto al rendimiento, la caída se ha detectado aproximadamente en el instante *23:19:57.615*, y el nuevo nodo maestro se ha establecido en el *23:19:59.620* según también el nuevo nodo maestro, el mismo tiempo empleado en el caso anterior.

Por lo tanto, a pesar de que se recomienda no utilizar un número par de nodos, ya que el que sea impar ayuda a la hora de tomar decisiones de quién será el nuevo nodo maestro, el tiempo empleado en resolver el problema en este caso ha sido el mismo desde el punto de vista del nuevo nodo maestro. Hay que tener en cuenta también que es un conjunto de nodos pequeño, pero aun así no ha habido ni una pequeña diferencia.

Caso 2 - Replica Set con presencia de árbitro En esta ocasión se van a realizar las mismas pruebas que en el mismo anterior pero contando con un nodo que ejercerá el papel de árbitro.

- En el primer escenario vamos a disponer del nodo maestro, de dos nodos esclavos y otro nodo más que será el arbitro. El primero de ellos al igual que en el caso anterior escuchará en el puerto 27020, mientras que el árbitro escuchará en el puerto 27023. La información que nos proporciona el comando rs.status() acerca de él es la que se muestra a continuación:

```
{
"_id" : 3,
"name" : "natalia-K55A:27023",
"health" : 1,
"state" : 7,
"stateStr" : "ARBITER",
"uptime" : 77,
"lastHeartbeat" : ISODate("2014-06-16T18:27:41Z"),
"lastHeartbeatRecv" : ISODate("2014-06-16T18:27:41Z"),
"pingMs" : 0
}
```

Al igual que en el caso anterior poblamos la base de datos con el fichero correspondiente, eliminamos la instancia mongod del maestro y examinamos el comportamiento del resto de nodos.

En esta ocasión ha sido el nodo que escucha en el puerto 27021 el seleccionado para ocupar el lugar del nodo maestro.

```
Puerto 27021:
 22:28:59.851 [rsBackgroundSync] replSet sync source
 problem: 10278 dbclient error communicating with server:
 natalia-K55A:27020
 20:29:01.183 [rsMgr] replSet info electSelf 1
 20:29:01.852 [rsMgr] replSet PRIMARY

Puerto 27022:
```

```
20:29:01.062 [rsMgr] not electing self, natalia-K55A:27021
would veto with 'natalia-K55A:27022 is trying to elect
itself but natalia-K55A:27020 is already primary and more
up-to-date'
20:29:01.183 [conn19] replSet info voting yea for
natalia-K55A:27021 (1)
20:29:03.062 [rsHealthPoll] replSet member
natalia-K55A:27021 is now in state PRIMARY

Puerto 27023:
20:29:01.061 [rsHealthPoll] replSet info natalia-K55A:27020
heartbeat failed, retrying
20:29:01.183 [conn17] replSet info voting yea for
natalia-K55A:27021 (1)
20:29:03.062 [rsHealthPoll] replSet member
natalia-K55A:27021 is now in state PRIMARY
```

De nuevo, observando los tiempos en los que el nodo maestro detecta la caída y reconoce que ha sido el elegido como nuevo maestro pasan dos segundos, el mismo que se ha registrado también en el nodo que ejerce de árbitro.

- A continuación se añadirá un nodo esclavo más, por lo que el árbitro para a escuchar en el puerto 27024 y entremos un total de tres esclavos. El método a seguir es el mismo, y el resultado es similar al del caso anterior, también el nodo que escuha en el puerto del nodo 27022 ha sido seleccionado como nuevo nodo maestro.

```
Puerto 27021:
20:43:22.602 [rsMgr] not electing self, natalia-K55A:27023
would veto with 'natalia-K55A:27021 is trying to elect
itself but natalia-K55A:27020 is already primary and more
up-to-date'
20:43:24.638 [conn16] replSet info voting yea for
natalia-K55A:27022 (2)
20:43:27.267 [rsHealthPoll] replSet member
natalia-K55A:27022 is now in state PRIMARY
```

```
Puerto 27022:
 20:43:22.267 [rsBackgroundSync] replSet sync source
 problem: 10278 dbclient error communicating with server:
 natalia-K55A:27020
 20:43:24.637 [rsMgr] replSet info electSelf 2
 20:43:25.268 [rsMgr] replSet PRIMARY

Puerto 27023:
 20:43:24.638 [rsMgr] not electing self, natalia-K55A:27022
 would veto with 'natalia-K55A:27023 is trying to elect
 itself but natalia-K55A:27020 is already primary and more
 up-to-date'
 20:43:24.638 [conn11] replSet info voting yea for
 natalia-K55A:27022 (2)
 20:43:26.033 [rsHealthPoll] replSet member
 natalia-K55A:27022 is now in state PRIMARY

Puerto 27024:
 20:43:22.293 [rsHealthPoll] replSet info
 natalia-K55A:27020 heartbeat failed, retrying
 20:43:24.638 [conn12] replSet info voting yea for
 natalia-K55A:27021 (1)
 20:43:26.294 [rsHealthPoll] replSet member
 natalia-K55A:27022 is now in state PRIMARY
```

En este caso encontramos una mayor diferencia de tiempo entre los eventos que hemos venido siguiendo, el error al detectar que el nodo maestro ha caído y el establecimiento de uno nuevo. Además, encontramos que dicha diferencia en el nuevo nodo maestro, el que escucha en el puerto 27022, es de aproximadamente tres segundos, mientras que en el nodo que ejerce de árbitro es de cuatro. Esta pequeña diferencia de tiempo bien puede estar causada por disponer de cinco nodos, lo cual nos lleva a pensar si en pequeños escenarios como este merece la pena incluir la figura del árbitro, el cual es otro nodo a consumir recursos y puede ralentizar la toma de decisiones en vez de influir positivamente.

Caso 3 - Caída de nodo durante la inserción en sharding Vamos a trabajar en este caso con el escenario del sharding. A la hora de realizar una inserción en este tipo de entornos el contenido se reparte entre los shards que lo componen, el objetivo aquí es comprender cómo se comportan los nodos cuando nos encontramos en un proceso de inserción y uno de los nodos se cae. Para ello crearemos el escenario anteriormente explicado e insertamos el fichero de cien mil registros utilizado en las pruebas anteriores.

Las consecuencias al eliminar uno de los nodos se detectan en la instancia mongos, donde se avisa de que no es posible conectar con la instancia caída, en este caso la que escucha en el puerto 27023. Además, al conectarnos a esta instancia para, por ejemplo, listar los elementos de la colección correspondiente nos muestra el siguiente error, el cual corresponde con la ausencia de dicho nodo:

```
mongos> db.prueba4.find()
error: {
"$err":"socket exception [CONNECT_ERROR] for natalia-K55A:27023",
"code" : 9001,
"shard" : "shard0002"
}
```

Por otro lado, si nos conectamos a las instancias restantes podemos ver que sí que podemos listar y contar los elementos que han sido insertados en ellos, en la instancia que escucha a través del puerto 27021 encontramos 30324 documentos, mientras que en el 27022 tenemos 34269 de los 100.000 insertados en un inicio.

En este caso las consecuencias han sido las esperadas, ya que al utilizar shards que no sean un conjunto de replicación enontramos el riesgo de que uno de ellos se caiga y no dispongamos de una copia de seguridad que nos permita recuperar los datos insertados. Por este motivo es recomendable que en estos escenarios se utilice también la replicación para obtener cierta seguridad, ya que son escenarios en los que se puede llegar a manejar gran cantidad de datos, y llegar a perderlos puede suponer un gran problema en la gran mayoría de los casos.

4.4. Conclusiones

El objetivo de este conjunto de pruebas, tal y como se ha venido diciendo, es el de utilizar los resultados obtenidos y los escenarios empleados para obtener dichos resultados en la docencia, concretamente en la asignatura "Administración de bases de datos". Por este motivo se puede echar en falta bien una mayor profundización en el tema o bien la elaboración de otro tipo de pruebas.

Realizando una visión global podemos ver claramente dos tipos de conjuntos de pruebas, el primero de ellos enfocado al rendimiento de las bases de datos MongoDB, mientras que el segundo está más centrado en el comportamiento, aunque también se han analizado sus tiempos de realización.

El objetivo que ha tenido la ejecución del primer conjunto de pruebas, ha sido el de conocer mejor las operaciones más comunes que se llevan a cabo en cualquier tipo de sistema de bases de datos, ya sea relacional o no relacional, algo relevante a la hora de familiarizarse con las bases de datos. Además, se pretende también comprobar el efecto que tiene la estructura del documento sobre el tiempo a la hora de realizar estas operaciones, ya que el diseño de de estos documentos es una de las principales tareas que deben ser desarrolladas por un administrador de bases de datos.

En cuanto al segundo conjunto de pruebas, el objetivo principal es el de conocer mejor su comportamiento ante diferentes situaciones que nos podemos encontrar de manera concisa y sencilla, ya que a lo largo de las diversas asignaturas que guardan relación con las bases de datos, el estudio está centrado en las bases de datos relacionales, por lo que su comportamiento se puede aprender a lo largo de las mismas. No ocurre esto con las bases NoSQL, motivo por el cual se presentan los dos escenarios principales que encontramos en MongoDB y casos típicos que podrían ocurrirnos a la hora de trabajar con dicho sistema.

Otro punto que cabe destacar, aunque no era el objetivo del proyecto, es la no utilización de las diversas herramientas de administración que nos proporciona MongoDB, cada una de ellas ha sido probada, pero descartada por diversos motivos.

1. **mongotop**: debido a los escenarios utilizados esta herramienta no es posible utilizarla, ya que solo sirve para instancias mongod, y recordamos que en el sharding tabajamos con mongos.

2. **mongostat**: muestra diversa información acerca de la consulta que se realice, si es una inserción, una actualización, si se eliminan datos, etc.

Junto con ello muestra otros datos como el tiempo, el número de conexiones abiertas o el estado de replicación, entre otras. El motivo por el cual no se utiliza esta herramienta es el poco interés de los datos que nos proporciona.

3. **MongoDB Management Service (MMS)**: herramienta web muy completa que nos proporciona MongoDB. Depende en qué entorno trabajemos, en Windows no ha sido posible hacerla funcionar, es una herramienta de compleja instalación y configuración según los datos que nos interesen. Para utilizarla de manera completa es necesario también disponer de *Munin*, herramienta de monitorización. A pesar de que es una herramienta muy completa no se ha utilizado debido a la manera en la que presenta los datos, ya que estos se mandan a la web y los presenta en unas gráficas, lo que conlleva un retardo y poca precisión.

Capítulo 5

Conclusiones Generales

5.1. Conclusiones

Finalmente los objetivos del proyecto se han alcanzado con éxito.

El objetivo principal era el de comprender y conocer más en profundidad la administración en bases de datos MongoDB a través de la realización de diversas pruebas de distinto índole. Junto con ello, a través del análisis de los diversos resultados obtenidos se pretendía comprender cómo se comporta ante diferentes escenarios que nos podemos encontrar a la hora de trabajar con esta base de datos.

A su vez, se pretendía que dichos análisis y resultados se presentaran de forma clara y concisa, de manera que puedan ser utilizados en la docencia, con el objetivo de que el alumno cuente también con una ligera visión de lo que son las bases de datos no relaciones y cómo funcionan.

Todos estos objetivos se han visto cumplidos a pesar de la aparición de diversos problemas que ya se presentan resueltos a lo largo del documento.

5.2. Trabajo futuro

Este proyecto ha sido una ligera introducción a la administración de bases de datos MongoDB. El objetivo era utilizar conceptos y escenarios que pudieran compararse con bases de datos relaciones y que dieran una visión general de dicha base de datos, por lo que como ya se ha venido diciendo a lo largo de la documentación, es posible realizar más tipos de pruebas y crear una

investigación más detallada.

Por lo tanto, a la hora de realizar pruebas más en profundidad cabría utilizar características mencionadas en la sección 4.1, en la cual se presentan las funcionalidades que MongoDB posee relacionadas con la administración.

Capítulo 6

Anexo A - Contenido del CD

El CD contiene el siguiente material:

- Documentación del proyecto (memoria.pdf).
- Scripts en javascript utilizados en la creación de los ficheros de pruebas.
- Scripts en python utilizados tanto en MongoDB como en MySQL para realizar las actualizaciones y borrados, útiles para tomar los tiempos de las operaciones.

Bibliografía

Academia.edu - NoSQL Database [En línea]. Disponible en: http://www.academia.edu/5352898/NoSQL_Database_New_Era_of_Databases_for_Big_Data_Analytics_-_Classification_Characteristics_and_Comparison [Último acceso: 22/05/2014]

Bases de datos orientadas a grafos y su enfoque en el mundo real, Washington A. Velásquez Vargas [En línea]. Disponible en: http://www.academia.edu/5731075/Bases_de_datos_orientadas_a_grafos_y_su_enfoque_en_el_mundo_real [Último acceso: 22/05/2014]

BIGDATA-STARTUP - MultiValue Databases [En línea]. Disponible en: http://www.bigdata-startups.com/open-source-tools/multivalue-databases/ [Último acceso: 22/05/2014]

BSON [En línea]. Disponible en: http://bsonspec.org/ [Último acceso: 22/05/2014]

Cassandra [En línea]. Disponible en: http://cassandra.apache.org/ [Último acceso: 22/05/2014]

Couchbase - Why NoSQL? [En línea]. Disponible en: http://www.couchbase.com/why-nosql/nosql-database [Último acceso: 22/05/2014]

db4objectos [En línea]. Disponible en: http://www.db4o.com/ [Último acceso: 22/05/2014]

Descarga generatedata [En línea]. Disponible en: http://benkeen.github.io/generatedata/ [Último acceso: 22/05/2014]

Descarga MongoDB [En línea]. Disponible en: http://www.mongodb.org/downloads/ [Último acceso: 22/05/2014]

Exploring the Different Types of NoSQL Databases, PILLAR GLOBAL [En línea]. Disponible en: http://blog.3pillarglobal.com/exploring-different-types-nosql-databases [Último acceso: 22/05/2014]

Genbetadev - NoSQL: clasificación de las bases de datos según el teorema CAP [En línea]. Disponible en: http://www.genbetadev.com/bases-de-datos/nosql-clasificacion-de-las-bases-de-datos-segun-el-teorema-cap [Último acceso: 22/05/2014]

Generatedata [En línea]. Disponible en: http://www.generatedata.com/ [Último acceso: 22/05/2014]

Intechnology [En línea]. Disponible en: http://www.intechnology.co.uk/resource-centre/byte-size.aspx# [Último acceso: 22/05/2014]

Json - Introducing JSON [En línea]. Disponible en: http://www.json.org/ [Último acceso: 22/05/2014]

MMS - MongoDB Management Service. Disponible en: https://mms.mongodb.com/ [Último accesso: 23/06/2014]

MongoDB [En línea]. Disponible en: http://www.mongodb.com [Último acceso: 22/05/2014]

Munin [En línea]. Disponible en: http://munin-monitoring.org/ [Último acceso: 23/06/2014]

Musings on NoSQL, Fred Beringer, Diciembre 2009 [En línea]. Disponible en: http://www.fredberinger.com/2009/12/musings-on-nosql/ [Último acceso: 22/05/2014]

Neo4j [En línea]. Disponible en: http://www.neo4j.org/ [Último acceso: 22/05/2014]

NoSQL [En línea]. Disponible en: http://nosql-database.org/ [Último acceso: 22/05/2014]

NoSQL vs. SQL: SQL is the new NoNoSQL [En línea]. Disponible en: http://www.nosql-vs-sql.com/ [Último acceso: 22/05/2014]

Objectivity [En línea]. Disponible en: http://www.objectivity.com/ [Último acceso: 22/05/2014]

OpenQM - OpenQ; Multivalue Database [En línea]. Disponible en: http://www.openqm.org/docs/ [Último acceso: 22/05/2014]

Oracle - Distribuited Database Concepts [En línea]. Disponible en: http://docs.oracle.com/cd/B10501_01/server.920/a96521/ds_concepts.htm [Último acceso: 22/05/2014]

Wikipedia - ACID [En línea]. Disponible en: http://en.wikipedia.org/wiki/ACID [Último acceso: 22/05/2014]

Wikipedia - Bases de datos orientadas a objetos [En línea]. Disponible en: http://es.wikipedia.org/wiki/Base_de_datos_orientada_a_objetos [Último acceso: 22/05/2014]

Wikipedia - CouchDB [En línea]. Disponible en: http://es.wikipedia.org/wiki/CouchDB [Último acceso: 22/05/2014]

Wikipedia - MultivalueDB [En línea]. Disponible en: http://en.wikipedia.org/wiki/MultiValue [Último acceso: 22/05/2014]