Universidad de Valladolid

Escuela de Ingeniería Informática

Trabajo Fin de Grado

Grado en Ingeniería Informática
Mención: Tecnologías de la información

Comparativa del rendimiento de bases de datos NoSQL con grandes volúmenes de datos

Autor:
Victor Guijarro Esteban

Tutor:
Dña. Carmen Hernández Díez

Resumen

En la actualidad, el uso de bases de datos NoSQL se ha extendido de forma generalizada, llegando a sustituir a las bases relacionales SQL “clásicas” en muchos aspectos. Esto es debido principalmente a la gran cantidad de información que se hace necesario almacenar y analizar, y a la necesidad imperiosa de su disponibilidad permanente.

Las limitaciones de las bases de datos relacionales en cuanto entra en juego ese curioso, y en ocasiones esquivo, concepto al que llamamos «Big Data», se hacen palpables, y no es demasiado arriesgado decir que las bases de datos NoSQL están originadas en las necesidades específicas para tratar con recursos de datos extremadamente grandes o complejos.

Existen varios tipos bien diferenciados de bases de datos NoSQL, con sus propias características y peculiaridades, pero en el mercado actual destacan dos de ellas, cada una perteneciente a un tipo diferente; MongoDB y Cassandra.

Estas bases de datos NoSQL están entre los 10 motores de bases de datos más usados en la actualidad, pero lo cierto es que hace ya más de un lustro que estas Bases de Datos NOSQL están entre las más usadas, y parece un ejercicio muy interesante analizar y comprender su funcionamiento, y probar sus virtudes o defectos ante un gran volumen de datos

Conceptos clave: Base de datos, NoSQL, Big Data, Rendimiento, MongoDB, Cassandra.

Tabla de Contenidos

1. Introducción **11**

1.1. Marco del proyecto y Estado del Arte . . . 12
1.2. Objetivos . . . 14
1.3. Metodología . . . 15

2. Big Data y NoSQL **17**

2.1. El nuevo mundo: Big Data . . . 19
2.2. La nueva alternativa: NoSQL . . . 21
2.2.1. Diferencias con las bases de datos relacionales: ACID Vs BASE . . . 23
2.2.2. Tipos de bases de datos NoSQL . . . 25
2.2.3. Escalabilidad horizontal . . . 28
2.2.4. Ventajas y Desventajas . . . 30

3. MongoDB y Cassandra **33**

3.1. MongoDB . . . 33
3.1.1. ¿Quién usa MongoDB? . . . 34
3.1.2. Características . . . 35
3.1.3. Consultas . . . 37
3.1.4. Naturaleza distribuida . . . 39
3.1.5. Seguridad . . . 41
3.2. Apache Cassandra . . . 42
3.2.1. ¿Quién usa Cassandra? . . . 42
3.2.2. Características . . . 43
3.2.3. Modelo de datos . . . 44
3.2.4. Lenguaje de Consultas . . . 45
3.2.5. Naturaleza distribuida . . . 47
3.2.6. Seguridad . . . 49
3.3. Principales diferencias . . . 50

4. Preparación del experimento **53**

4.1. Obtención de los datos . . . 53
4.1.1. Elección del conjunto de datos . . . 54
4.1.2. Descripción del conjunto de datos elegido . . . 55
4.2. Obtención de los modelos de bases de datos . . . 55
4.2.1. Modelado de datos NoSQL . . . 55
4.2.2. Modelado de los datos elegidos . . . 57
4.3. Adecuación de los ficheros . . . 59
4.4. Implementación de las bases de datos . . . 60

4.4.1. Creación de las bases de datos . . . 60
4.5. Máquinas de prueba y métricas de rendimiento . . . 61

5. Pruebas realizadas **63**
5.1. Operaciones sobre las bases de datos . . . 63
5.1.1. Inserciones . . . 63
5.1.2. Consultas . . . 64
5.1.3. Actualizaciones . . . 65
5.1.4. Borrado . . . 66
5.2. Mediciones y Ejecución de comandos . . . 66

6. Análisis de los resultados **69**
6.1. MongoDB . . . 69
6.2. Cassandra . . . 70
6.3. Comparación gráfica . . . 72
6.4. Valoración de las hipótesis formuladas y conclusiones extraídas del experimento . . . 75

7. Conclusiones **77**

Anexos **83**

A. Contenido del soporte digital **85**

B. Comandos y logs **87**
I. Comandos para la modificación de la fuente de datos . . . 87
II. Hardware de la máquina virtual . . . 88
III. Comandos para la realización de las pruebas . . . 89

C. Scripts **91**
I. inicializacion.sh . . . 91
II. creacion.cql . . . 91
III. mongo_script_pruebas.sh . . . 92
IV. cassandra_script_pruebas.sh . . . 93

Lista de Figuras

1.1. Ranking BD-Engine Julio 2018 . . . 13
1.2. Ranking BD-Engine Julio 2019 . . . 13
1.3. Ranking BD-Engine Octubre 2016 . . . 13
1.4. Ranking BD-Engine Julio 2014 . . . 14
1.5. Gráfico de evolución de la popularidad de las bases de datos MongoDB y Cassandra frente a Oracle y MySQL . . . 14

2.1. Infografía sobre los datos generados en un minuto de 2018 . . . 18
2.2. Ecosistema Big Data . . . 19
2.3. Principales características consideradas en el Big Data . . . 20
2.4. Triángulo CAP . . . 22
2.5. Ejemplo de almacenamiento de información con un esquema de clave-valor . . . 25
2.6. Ejemplo de almacenamiento de información en una base de datos documental. . . . 26
2.7. Ejemplo de almacenamiento de información con una base de datos orientada a columnas. 27
2.8. Ejemplo de almacenamiento de información con un esquema orientado a grafos. . . . 28
2.9. Escalabilidad vertical Vs. Escalabilidad horizontal. . . . 29

3.1. Replicación en MongoDB . . . 39
3.2. Sharding en MongoDB . . . 40
3.3. Modelo de filas y columnas en Cassandra. . . . 44
3.4. Modelo de datos de Cassandra. . . . 45

4.1. Diseño Bases de Datos Relacionales. . . . 56
4.2. Ejemplo de diagrama Entidad Relación. . . . 56

6.1. Comparación de tiempos de inserción entre MongoDB y Cassandra en I5. . . . 72
6.2. Comparación de tiempos de inserción entre MongoDB y Cassandra en MV. . . . 72
6.3. Comparación de tiempos de inserción entre MongoDB y Cassandra en I3. . . . 73
6.4. Comparación de tiempos de búsqueda entre MongoDB y Cassandra en I5. . . . 73
6.5. Comparación de tiempos de búsqueda entre MongoDB y Cassandra en MV. . . . 74
6.6. Comparación de tiempos de búsqueda entre MongoDB y Cassandra en I3. . . . 74
6.7. Comparación de tiempos de actualización entre MongoDB y Cassandra en I5. . . . 74
6.8. Comparación de tiempos de actualización entre MongoDB y Cassandra en MV. . . . 75
6.9. Comparación de tiempos de actualización entre MongoDB y Cassandra en I3. . . . 75

Lista de Tablas

2.1. Características ACID vs. Base 24
2.2. Principales diferencias entre las características de las bases de datos relacionales y NoSQL. 31

3.1. Principales diferencias entre MongoDB y Cassandra. 51

4.1. Máquinas sobre las que se realizarán las pruebas 61

6.1. Datos obtenidos en las pruebas de MongoDB sobre el portátil i5 69
6.2. Datos obtenidos en las pruebas de MongoDB sobre la máquina virtual 70
6.3. Datos obtenidos en las pruebas de MongoDB sobre el portátil I3 70
6.4. Datos obtenidos en las pruebas de Cassandra sobre el portátil I5 70
6.5. Datos obtenidos en las pruebas de Cassandra sobre la máquina virtual 70
6.6. Datos obtenidos en las pruebas de Cassandra sobre el portátil I3 70
6.7. Datos obtenidos al repetir las pruebas de búsqueda sobre Cassandra. 71

Capítulo 1

Introducción

Vivimos en la *Era de la Información*, donde tanto nuestra historia como nuestro día a día están completamente ligados a las tecnologías de la información y la comunicación (TIC) [1]. Este periodo de nuestra historia presenta sus antecedentes en tecnologías del siglo XIX y XX, como el teléfono, la radio y la televisión, pero sin duda alguna los avances en la informática y en internet han sido los grandes instigadores de los tiempos en los que vivimos y de los que están aún por llegar.

Desde hace ya varias décadas, la información y los datos han alcanzado un papel y un valor inmenso en nuestro mundo, y cómo almacenarlos correctamente y cómo sacar provecho de ellos se han vuelto asuntos de importancia capital.

Dentro de la tecnología informática, las bases de datos han ido aumentando su importancia y sus aplicaciones, siendo la mejor solución para el almacenamiento de la información. Entre los diversos sistemas de bases de datos existentes, desde 1970 las bases relacionales no tardaron en consolidarse como un nuevo paradigma en los modelos de bases de datos, y ambos, tanto las tecnologías de BD relacionales como el paradigma que las sostienen, han llegado a instaurarse como el estándar de facto para el almacenamiento y el manejo de los datos.

Sin embargo, día a día el manejo de la información se hace más complicado y complejo; Hace unas décadas el *tamaño de internet*, o mejor dicho, de la información que contiene podía medirse en TB (Terabytes), mientras que en la actualidad se estima que estamos en el orden de los ZB [2] (Zettabytes). Con las últimas tecnologías, como los millones y millones de dispositivos *IoT* y el *Edge Computing* [3] o el *Big Data* y las redes sociales , se generan más y más datos (que pueden tener un tamaño muy grande), y además se hace imperativo la correcta explotación de los mismos, para sacar de ellos provecho y generar valor.

En los últimos años ha aumentado el interés por las bases de datos **NoSQL** (Not only SQL), un nuevo conjunto de tecnologías que pueden contribuir al manejo de la información, y como menciona Neal Leavitt [4]:"Las organizaciones que recopilan grandes cantidades de datos no estructurados son cada vez más propensas al uso de las bases de datos NoSQL". Este nuevo paradigma surgió en gran medida debido a las dificultades que tienen las bases de datos relacionales a la hora de enfrentarse a las ingentes cantidades de información asociadas al **Big Data**, y es que el incremento en el volumen de datos que se hace necesario manejar avanza a mayor velocidad que el incremento de la capacidad del hardware. [5]

Y es este interés que suscitan las bases de datos NoSQL en la actualidad el que ha dado pie a este trabajo de Fin de Grado. Este tipo de bases de datos se ha vuelto muy popular en los últimos años, y cada vez son más las empresas que apuestan por emplear estas bases de datos (aunque esa decisión pueda no estar demasiado justificada en algunas ocasiones), y es por eso que en este trabajo se profundizará en

el paradigma NoSQL y más concretamente en 2 de los gestores de BD NoSQL más populares, y se tratará de averiguar cual de ellos podría dar mejores resultados al hacer frente a una gran cantidad de datos.

1.1. Marco del proyecto y Estado del Arte

Durante los últimos años las bases de datos NoSQL han atraído mucha atención en muchos de los sectores tecnológicos más punteros. Todo este interés ha suscitado numerosas comparaciones y análisis de estas bases de datos, generalmente comparándolas con las bases de datos relacionales más usadas (MySQL o PostgreSQL, debido a las licencias de uso). También, sobre todo en los últimos 5 años, se han realizado un número significativo de TFGs/TFMs en los que las bases de datos NoSQL eran protagonistas.

Sharma, P y Prabha, J. en este artículo de la 3ª *International Conference on Internet of Things and Connected Technologies (ICIoTCT)* de 2018 [6] realizan una pequeña comparativa de rendimiento con diversos parámetros entre Cassandra y MongoDB ante una carga de trabajo considerable y en tiempo real, proveniente de redes sociales (Twitter).

En este artículo de la *International Conference on Computing and Communication Technologies for Smart Nation (IC3TSN)* de 2017 [7] los autores realizan un análisis y una comparación teórica de las bases de datos NoSQL más representativas de cada uno de los tipos.

Fraczek, K. y Plechawska-Wojcik, M. [8] realizan una comparativa de rendimiento entre bases de datos relacionales y no relacionales en el contexto de aplicaciones webs.

Por su parte, Bhamra, K. [9] se centra en realizar un análisis y una comparativa completamente teóricos de MongoDB y Cassandra, de forma que este documento sería el más aproximado al objetivo de este trabajo que se ha podido encontrar.

En un entorno académico mucho más cercano, se puede encontrar una serie de Trabajos de Fin de Grado centrados en bases de datos NoSQL.

Escudero, D. [10] se centra en su TFG en comprender, profundizar y desgranar a Cassandra y su funcionamiento, realizando un estudio teórico, además de ver como se construye una base de datos con este sistema gestor e implementarlo con algunos casos prácticos.

Román, N. [11] pone el foco de su TFG en MongoDB, profundizando en este sistema de bases de datos y más concretamente, en su manejo y administración, aunque también se realiza un experimento sobre las operaciones más habituales para ver el rendimiento que puede ofrecer el gestor de bases de datos.

García, R. [12] y Bas, V. [13] se centran en sus respectivos TFGs en realizar una comparativa de rendimiento entre un gestor de bases de datos relacional y uno NoSQL.

En estos y en la mayoría de los artículos y trabajos similares que se han encontrado, y que comparan bases de datos relacionales y NoSQL, se llega a la conclusión de que las bases de datos NoSQL brillan en situaciones y circunstancias muy específicas, aunque pueden ser una alternativa con unos resultados similares en muchas circunstancias.

La segunda parte más significativa que da peso y enmarca este proyecto, es la popularidad que han ganado estas bases de datos en los últimos años. Para visualizar ésto, se puede tomar como referencia la web db-engines [14] que puntúa más de 300 sistemas gestores de bases de datos y los ordena en función de su popularidad. Si bien, hay que entender que el concepto de *popularidad* está profundamente relacionado con las menciones de los SGBD en medios especializados y ofertas de trabajos, su relevancia en redes sociales, el interés general –Google Trends– o los hilos de discusión sobre ellas, y no tiene en cuenta cosas como el número de instalaciones o el uso real que tienen estos sistemas en entornos reales.

De acuerdo al ranking de bases de datos en Julio de 2018 y de 2019, que se puede ver en las imágenes 1.1 y 1.2, se puede apreciar claramente la relevancia que tienen los dos SGDB objetivos de este estudio, ya que MongoDB ocupa la 5ª posición y Cassandra la 10ª.

343 systems in ranking, July 2018

Rank Jul 2018	Rank Jun 2018	Rank Jul 2017	DBMS	Database Model	Score Jul 2018	Score Jun 2018	Score Jul 2017
1.	1.	1.	Oracle	Relational DBMS	1277.79	-33.47	-97.09
2.	2.	2.	MySQL	Relational DBMS	1196.07	-37.62	-153.04
3.	3.	3.	Microsoft SQL Server	Relational DBMS	1053.41	-34.32	-172.59
4.	4.	4.	PostgreSQL	Relational DBMS	405.81	-4.86	+36.37
5.	5.	5.	MongoDB	Document store	350.33	+6.54	+17.56
6.	6.	6.	DB2	Relational DBMS	186.20	+0.56	-5.05
7.	7.	↑ 9.	Redis	Key-value store	139.91	+3.61	+18.40
8.	8.	↑ 10.	Elasticsearch	Search engine	136.22	+5.18	+20.25
9.	9.	↓ 7.	Microsoft Access	Relational DBMS	132.58	+1.59	+6.45
10.	10.	↓ 8.	Cassandra	Wide column store	121.06	+1.84	-3.07

Figura 1.1: Ranking BD-Engine Julio 2018

350 systems in ranking, July 2019

Rank Jul 2019	Rank Jun 2019	Rank Jul 2018	DBMS	Database Model	Score Jul 2019	Score Jun 2019	Score Jul 2018
1.	1.	1.	Oracle	Relational, Multi-model	1321.26	+22.04	+43.47
2.	2.	2.	MySQL	Relational, Multi-model	1229.52	+5.89	+33.45
3.	3.	3.	Microsoft SQL Server	Relational, Multi-model	1090.83	+3.07	+37.42
4.	4.	4.	PostgreSQL	Relational, Multi-model	483.28	+6.65	+77.47
5.	5.	5.	MongoDB	Document	409.93	+6.03	+59.60
6.	6.	6.	IBM Db2	Relational, Multi-model	174.14	+1.94	-12.06
7.	7.	↑ 8.	Elasticsearch	Search engine, Multi-model	148.81	-0.01	+12.59
8.	8.	↓ 7.	Redis	Key-value, Multi-model	144.26	-1.86	+4.35
9.	9.	9.	Microsoft Access	Relational	137.31	-3.70	+4.73
10.	10.	10.	Cassandra	Wide column	127.00	+1.82	+5.95

Figura 1.2: Ranking BD-Engine Julio 2019

Echando la vista varios años atrás, podemos ver en las imágenes 1.3 y 1.4, esta tendencia se ha mantenido al menos desde 2014, ya que desde entonces MongoDB ha estado siempre ubicado en la 4ª o 5ª posición, mientras que Cassandra ha ido oscilando de la 7ª a la 10ª.

316 systems in ranking, October 2016

Rank Oct 2016	Rank Sep 2016	Rank Oct 2015	DBMS	Database Model	Score Oct 2016	Score Sep 2016	Score Oct 2015
1.	1.	1.	Oracle	Relational DBMS	1417.10	-8.46	-49.85
2.	2.	2.	MySQL	Relational DBMS	1362.65	+8.62	+83.69
3.	3.	3.	Microsoft SQL Server	Relational DBMS	1214.18	+2.62	+90.95
4.	↑ 5.	4.	MongoDB	Document store	318.80	+2.81	+25.54
5.	↓ 4.	5.	PostgreSQL	Relational DBMS	318.69	+2.34	+36.56
6.	6.	6.	DB2	Relational DBMS	180.56	-0.62	-26.25
7.	7.	↑ 8.	Cassandra	Wide column store	135.06	+4.57	+6.05
8.	8.	↓ 7.	Microsoft Access	Relational DBMS	124.68	+1.36	-17.16
9.	↑ 10.	↑ 10.	Redis	Key-value store	109.54	+1.75	+10.75
10.	↓ 9.	↓ 9.	SQLite	Relational DBMS	108.57	-0.05	+5.90

Figura 1.3: Ranking BD-Engine Octubre 2016

223 systems in ranking, September 2014

Rank	Last Month	DBMS	Database Model	Score	Changes
1.	1.	Oracle	Relational DBMS	1466.91	-3.95
2.	2.	MySQL	Relational DBMS	1297.14	+15.92
3.	3.	Microsoft SQL Server	Relational DBMS	1208.87	-33.62
4.	4.	PostgreSQL	Relational DBMS	255.79	+5.94
5.	5.	MongoDB	Document store	240.98	+3.63
6.	6.	DB2	Relational DBMS	197.03	-9.39
7.	7.	Microsoft Access	Relational DBMS	140.48	+0.86
8.	8.	SQLite	Relational DBMS	92.61	+3.74
9.	↑ 10.	Cassandra	Wide column store	87.86	+5.96
10.	↓ 9.	Sybase ASE	Relational DBMS	85.42	-0.75

Figura 1.4: Ranking BD-Engine Julio 2014

Además, en el gráfico mostrado en la figura 1.5 se puede ver como ha evolucionado la popularidad de estos sistemas desde 2013, siendo casi siempre una tendencia ascendente para ambos, y se puede ver con mucha claridad que su popularidad dista mucho de estar cercana a los SGDB relacionales más habituales como Oracle o MySQL.

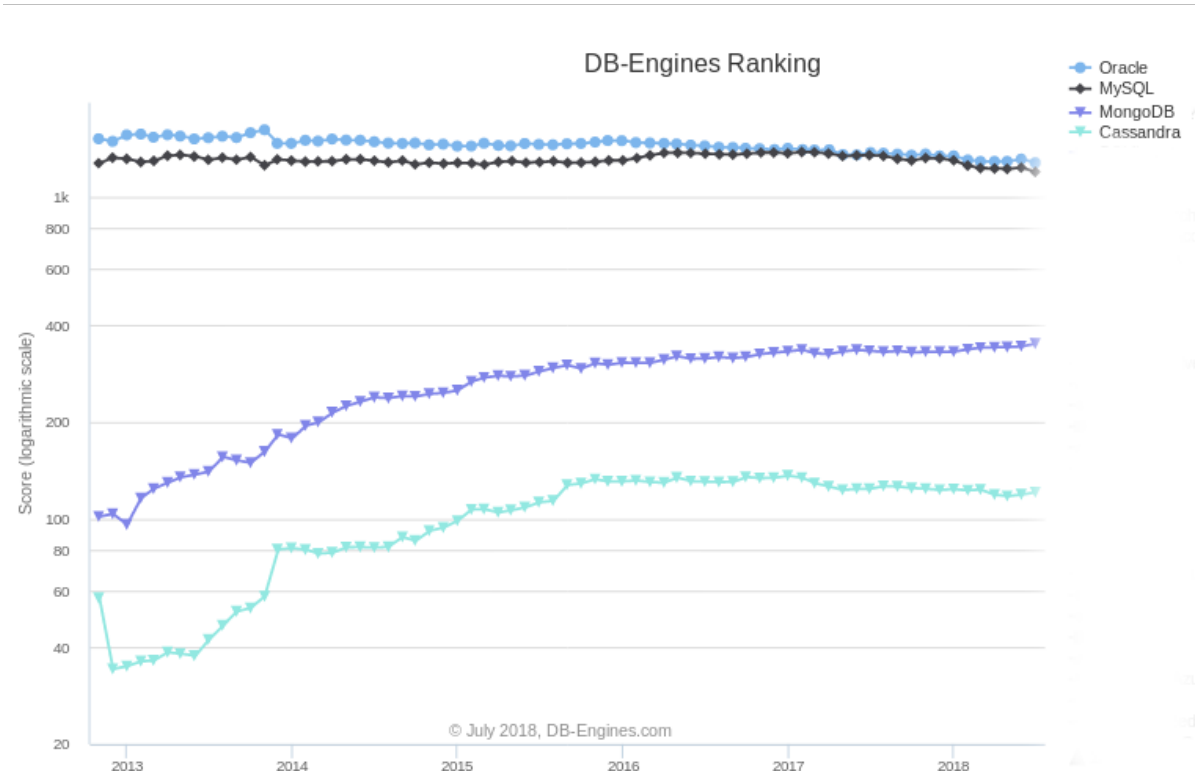

Figura 1.5: Gráfico de evolución de la popularidad de las bases de datos MongoDB y Cassandra frente a Oracle y MySQL

1.2. Objetivos

Tanto la motivación como los objetivos de este proyecto se han comentado o se han podido entrever a lo largo de este capitulo introductorio, sin embargo, en esta sección se concretarán y dejarán completamente claros a continuación:

- El objetivo principal de este trabajo es el de profundizar en las bases de datos no relacionales. De

esta forma se pretende tener una clara visión panorámica del mundo NoSQL y de sus implicaciones, haciendo una leve comparativa con las bases relacionales clásicas cuando sea necesario para dejar claro alguno de los conceptos principales.

- Concretamente, se profundizará en dos de las bases de datos NoSQL más populares en la actualidad; MongoDB y Cassandra. Así pues, se tratará de alcanzar una profunda comprensión de su funcionamiento y estructura interna, además de aprender a instalarlas y configurarlas debidamente, y a manejar el lenguaje de consulta que empleen.

- Por último, se quiere realizar una comparativa de rendimiento entre estas dos bases de datos elegidas cuando se las somete a una gran cantidad de datos.

  En este punto, se hace necesario remarcar un aspecto importante que no entra dentro de los objetivos de este trabajo de fin de grado, y es que, como se verá en el Capítulo 2, las bases de datos NoSQL tienen un importante componente distribuido que no será explorado dentro de las pruebas de rendimiento, aunque sí en los aspectos teóricos.

  Al finalizar este trabajo, se espera tener los conocimientos y datos necesarios para poder elegir entre una de estas bases de datos y ver cual de ellas pueda ser más adecuada ante un problema determinado que requiera manejar una gran cantidad de datos.

1.3. Metodología

Este trabajo presenta dos partes bien diferenciadas; una eminentemente teórica, que comprende los dos primeros puntos de los objetivos mencionados en la sección anterior, y una que resulta casi completamente práctica, que son las pruebas sobre estas bases de datos para comparar su rendimiento.

Para completar la parte teórica será necesario realizar un trabajo de investigación sobre el mundo del Big Data, NoSQL y concretamente, sobre las bases de datos MongoDB y Cassandra. También se realizará la instalación y configuración de cada una de las bases de datos, para lo que serán especialmente útiles los manuales y la documentación oficial de cada una de ellas.

La parte práctica de este trabajo consistirá en la implementación de las dos bases de datos y en la realización de las pruebas de rendimiento pertinentes sobre ellas.

Para poder llevar esto a cabo, primero será necesario conseguir una gran cantidad de datos, que tomaremos de un banco de datos que se especificará más adelante. Una vez que dispongamos del volumen de carga necesario, éste se fraccionará de manera que podamos probar las bases de datos con diferentes volúmenes de datos.

En cualquier caso, antes de poder probar los datos será necesario crear dos modelos de bases de datos equivalentes, uno para MongoDB y otro para Cassandra, de forma que la comparativa pueda llevarse a cabo de forma correcta. Una vez se disponga de los datos y de las bases de datos operativas, se podrán realizar las diversas pruebas sobre ellas.

Finalmente, para poder evaluar el rendimiento, se tomarán los tiempos que cada uno de los sistemas gestores de bases de datos necesite para poder realizar las operaciones o las consultas pertinentes. Estas mediciones se repetirán de 3 a 5 veces, para poder descartar anomalías en el comportamiento o *outliers*. Tanto las pruebas como las mediciones serán automatizadas a través de scripts, que se añadirán a los apéndices de esta memoria, y los datos y resultados obtenidos se mostrarán en tablas y gráficos en la sección correspondiente de este documento.

Capítulo 2

Big Data y NoSQL

Desde el nacimiento de la informática, ésta ha tenido como uno de sus principales objetivos la recolección, almacenamiento y gestión de datos. En 1970 se establecen los fundamentos teóricos del modelo relacional de base de datos y gracias a ello se pasó de almacenar cantidades limitadas de información en tarjetas perforadas al almacenamiento masivo del orden de los terabytes de información.

El modelo relacional fue diseñado y creado para soportar cientos o miles de usuarios simultáneos, así como para poder manejar grandes cantidades de datos, pero debido al avance que existe día a día en las tecnologías de la información junto al incremento de la información recabada a través de medios como el web, el modelo relacional ha comenzado a tener problemas cuando se trata de grandes volúmenes de información.

Con la profunda penetración de la red de redes en todos los aspectos de la vida diaria, los usuarios han comenzado a tener una mayor integración con los sistemas, lo que ha propiciado que servicios como las redes sociales, blogs, e-comerce, etc, tuvieran cada vez una mayor demanda de almacenamiento de datos, lo cual, a su vez, trajo consigo importantes retos en los temas de escalabilidad, rendimiento y velocidad. Especialmente en el tema de escalabilidad, las bases de datos relacionales comenzaron a dar muestras de que las exigencias de sus modelos relacionales generan tiempos de respuesta muy altos para poder incrementar la manera en que la información es almacenada sin perder coherencia entre ella.

Para darnos una idea de la cantidad de información que la web genera y su evolución, podemos remitirnos a los estudios hechos por la empresa DOMO [15, 16] en 2018 y 2019, que analizan la cantidad de información que los internautas (4.39 mil millones de personas aproximadamente en 2019) generan por minuto:

- Se reproducen 4 millones y medio de vídeos sólo en YouTube, hay aproximadamente un millón de usuarios viendo vídeos en streaming en Twitch y se consumen cerca de 700.000 horas de vídeo en Netflix.
- Realizan 4 millones y medio de consultas en Google.
- Escriben más de 500.000 tweets en Twitter y se realizan más de 200.000 videollamadas por Skype.
- Suben unas 50.000 fotos y postean cerca de 300.000 stories en Instagram.
- Se envían más de 180 millones de emails.
- Se realizan cerca de millón y medio de *swipes* en Tinder, 10.000 viajes en Uber y 2.000 reservas por Airbnb.

Estos datos aquí mencionados sólo son una muestra ilustrativa, y tanto en la infografía que se ve en la figura 2.1 como en el informe completo citado anteriormente se pueden apreciar otros datos muy interesante sobre las operaciones que se hacen aproximadamente en cada minuto del año 2019.

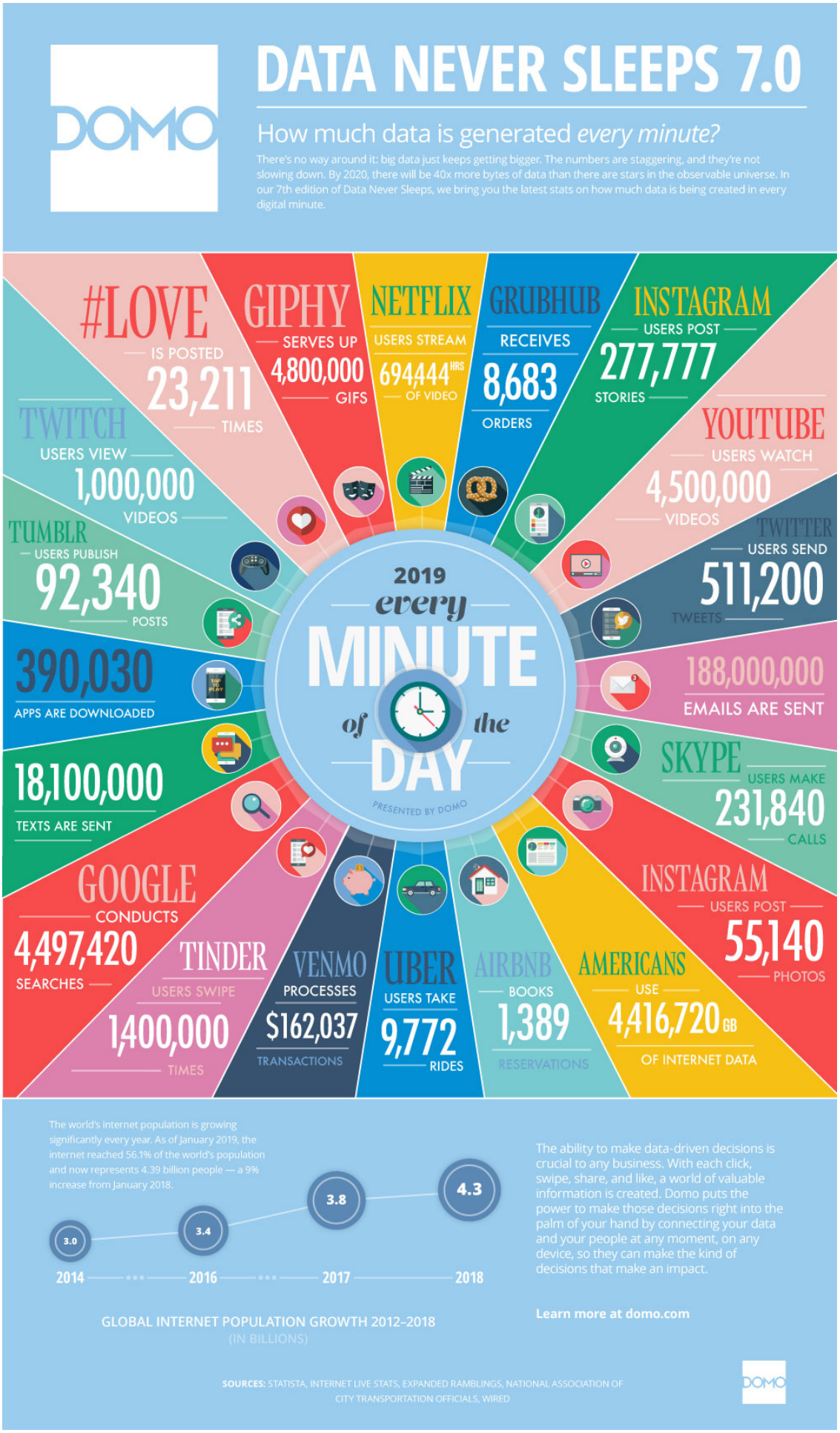

Figura 2.1: Infografía sobre los datos generados en un minuto de 2018

2.1. El nuevo mundo: Big Data

El propio concepto del *Big Data* es en ocasiones bastante confuso y está lleno de matices. En términos generales, puede referirse a esa tendencia tecnológica que ha abierto las puertas hacia un nuevo enfoque de entendimiento y toma de decisiones, la cual se utiliza para describir enormes cantidades de datos (estructurados, no estructurados y semi estructurados) que tomaría demasiado tiempo y sería muy costoso cargarlos a una base de datos relacional para su análisis.

De tal manera que este concepto se aplica para toda aquella información que no puede ser procesada o analizada utilizando procesos o herramientas tradicionales. Sin embargo, –y es aquí donde más confusión tiende a causar el término– Big Data no se refiere a ninguna cantidad de información en específico, aunque es usualmente utilizado cuando se habla en términos de petabytes y exabytes de datos.

Tampoco se refiere a ningún tipo concreto de datos, ni a ninguna de las maneras en las que pueden ser generados, representados o utilizados, por ejemplo: dispositivos móviles, audio, vídeo, sistemas GPS, sensores digitales en equipos industriales, automóviles, anemómetros, etc., de tal forma que las aplicaciones que analizan o utilizan estos datos requieren que la velocidad de respuesta sea lo suficientemente rápida como para lograr llevar a cabo su función en el momento preciso.

Así pues, tratando de dar una definición más formal a éste concepto, podríamos decir que el *Big Data* agrupa las técnicas de almacenamiento, análisis y manejo de inmensos repositorios de datos –en un tiempo razonable– y dando por hecho que estos repositorios de datos son tan grandes que resulta imposible tratarlos con las herramientas de bases de datos y analíticas convencionales.

De esta forma, han surgido multitud de diversas tecnologías que continuamente están evolucionando, mejorando las anteriores y dejándolas obsoletas a un ritmo vertiginoso. Cada una de estas nuevas tecnologías por si mismas no sería capaz de satisfacer las necesidades originadas por las grandes cantidades de datos que se tratan en el big data, sin embargo, la combinación de estas piezas da resultado a lo que se ha llamado como "Ecosistema Big Data" [17], y cuyos principales componentes y tecnologías de referencia pueden observarse en la imagen 2.2, y si bien no cabe duda de que el almacenamiento es una parte importante, queda patente que es sólo una pieza más de lo que en realidad es el big data.

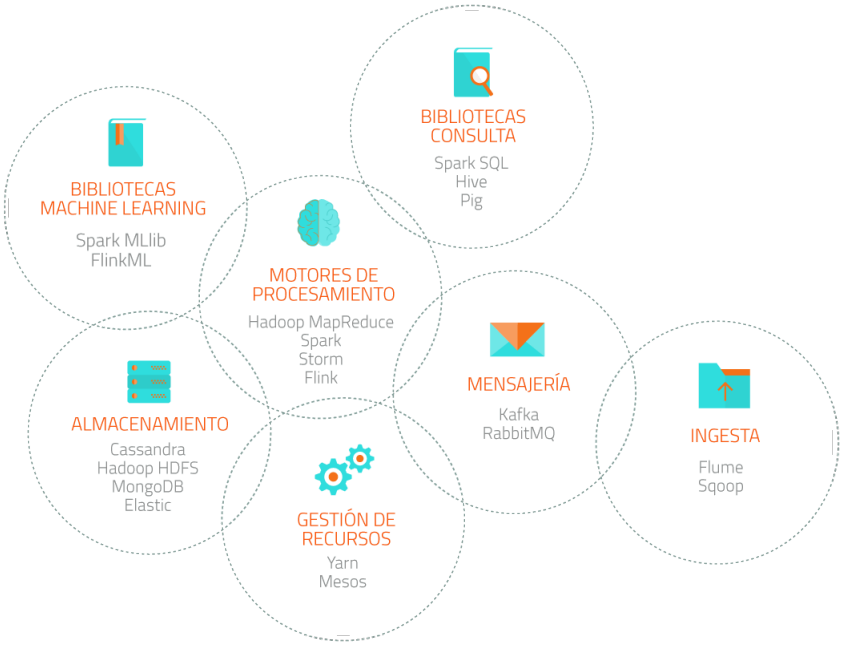

Figura 2.2: Ecosistema Big Data

El BigData y sus componentes centran sus características en las tres partes fundamentales que se listan a continuación, aunque recientemente se han incluido otras que se pueden apreciar también en la imagen 2.3:

- **Volumen**. Grandes volúmenes de datos, generalmente del orden de TeraBytes de información o mayor.
- **Variedad**. El concepto de BigData también suele venir acompañado de diversos tipos de fuentes de datos, ya sean estructurados, no estructurados o ambos a la vez.
- **Velocidad**. El procesamiento y posterior análisis ha de realizarse prácticamente en tiempo real para poder mejorar la toma de decisiones en base a la información generada.

Figura 2.3: Principales características consideradas en el Big Data

De este punto, podemos dirimir que el Big Data como tal, en un sentido bastante estricto, se puede considerar una cuestión de escala. No se fija en ningún momento una cantidad máxima de datos o una velocidad mínima en la que deben ser procesados, por que estos límites vienen dados por la capacidad (de almacenamiento, procesamiento, de la red...) de los propios sistemas, y evidentemente, cada sistema tiene los suyos propios.

Con esta premisa de que el Big Data es una cuestión de escala, y tomando como idea que el concepto del Big Data hace referencia al momento en el que el repositorio de datos a manejar es simplemente inviable utilizando otras técnicas más convencionales, podemos ver fácilmente que no se puede considerar *el mismo Big Data* según hablemos de, por ejemplo, un dispositivo wearable, una Raspberry Pi, un portátil de uso doméstico o un servidor perteneciente a un centro de procesamiento de datos.

Claramente, cada uno de estos ejemplos tiene unas limitaciones de hardware y software que definen para cada situación lo que es *viable*, de esta forma en un dispositivo modesto y con poca capacidad, el análisis de unos cuantos MegaBytes o de un único GigaByte de información podría llevarse a cabo únicamente empleando técnicas de Big Data, mientras que en otros dispositivos más potentes esa misma información

podría manejarse sin ningún problema con técnicas convencionales y el límite a lo que llamaríamos Big Data podría estar en el orden de TeraBytes de información o unidades superiores.

Actualmente se cree que la información generada a partir de la abundancia de sensores, micrófonos, cámaras, elementos IoT y un largo etc, en nuestra vida diaria será dentro de poco el segmento más grande de toda la información disponible. En este punto, Big Data será una herramienta indispensable para quienes se dedican a la investigación, pues gracias a ella será posible descubrir cosas que sin estas herramientas habría tomado años de investigación y recopilación de información.

La cuestión que afronta el Big Data es, por tanto, el modo en el que esa ingente cantidad de datos, lo que en muchas ocasiones es nombrado como el "data deluge" (el diluvio universal de datos) que se almacenan y producen diariamente pueda ser gestionado de tal modo que se convierta en conocimiento; y con el conocimiento, en valor. [18]

2.2. La nueva alternativa: NoSQL

Cuando se consideran las tecnologías necesarias para abordar el problema del Big Data, tiende a ser natural considerar en primer lugar el sistema de gestión de bases de datos, que se encargará de almacenar la gran cantidad de datos que deberemos tratar.

La mayor parte de las bases de datos más utilizadas se encuentran ya optimizadas para almacenar y gestionar grandes cantidades de datos. Desde hace ya muchos años los sistemas basados en el modelo relacional se han venido utilizando de manera exitosa tanto en la industria como en entornos de investigación. Sin embargo, el umbral que define el término Big Data implica, en primer lugar, un cambio de paradigma en el modelo de gestión de la información, ya que al enfrentar a las bases de datos relacionales con el big data, surge un importante problema de "escalabilidad".

Y este problema de escalabilidad de SQL fue reconocido por empresas Web 2.0, con grandes necesidades de datos e infraestructura, como Google, Amazon y Facebook. Ellos solos tuvieron que buscar soluciones propias a este problema. Estas soluciones dieron lugar a una serie de sistemas de gestión de base de datos nuevos, con un enfoque en el rendimiento, la fiabilidad y la coherencia. Para esto se reutilizaron y mejoraron varias estructuras de indexación existentes con el propósito de mejorar la búsqueda y el rendimiento de lectura.

En primer lugar, surgieron bases de datos desarrolladas por estas grandes empresas para satisfacer sus necesidades específicas; como BigTable de Google, que se considera el primer sistema NoSQL y DynamoDB de Amazon. El éxito de estos sistemas patentados inició el desarrollo de varios sistemas de bases de datos inspirados por ellos –la mayoría de código abierto– siendo los más conocidos a día de hoy Hypertable, Cassandra, MongoDB, HBase y Redis.

Mientras que en las bases de datos que siguen el modelo relacional se garantizan determinadas propiedades que a priori parecen completamente imprescindibles (ACID), hoy por hoy, es complicado gestionar grandes volúmenes de datos sin relajar algunas de ellas. Precisamente de esta relajación y de la necesidad de cumplir con otras propiedades y necesidades nuevas, emerge un nuevo paradigma de gestión de datos: **NoSQL**.

Las propiedades o características que deben poseer estos nuevos sistemas, principalmente en el entorno de Internet, son:

- *Disponibilidad.* Estos sistemas en si mismos deben tener una altísima disponibilidad para los usuarios, de forma que siempre, o al menos la mayor parte del tiempo, sean accesibles y funcionales.
- *Tolerancia a fallos*, especialmente a fallos de red, de forma que, aunque queden aislados máquinas o partes del sistema, no afecten a la disponibilidad y funcionalidad del sistema.

- *Gran capacidad de almacenamiento* y *Alta capacidad de Entrada/Salida*. Estas características pueden resultar evidentes en el contexto en que nos encontramos, pero sin duda son las dos más vitales e importantes para estos nuevos sistemas, ya que resultan ser los factores más limitantes para estos sistemas.

Para conseguir cumplir con todo esto, un sistema con una arquitectura distribuida resulta ideal, aunque normalmente habrá que renunciar a las garantías ACID, ya que se requieren resultados de rendimiento óptimos. Por ejemplo, lo más normal es que la consistencia no esté garantizada, dado que en una arquitectura distribuida los cambios deben propagarse entre las diferentes máquinas. Cumplir con los principios ACID supondría una lacra para el rendimiento que haría el sistema poco óptimo para el escenario más normal de Big Data.

Hoy en día se pueden encontrar diferentes sistemas de bases de datos distribuidos que potencian diferentes atributos. De hecho, existe un teorema sobre sistemas distribuidos, **el teorema CAP** o teorema de Brewer, que los agrupa de forma clara. Según este teorema, un sistema distribuido o un sistema web no puede satisfacer de manera completa los siguientes 3 atributos al mismo tiempo:

- **Consistencia fuerte**: Todos los nodos ven los mismos datos al mismo tiempo. Esto implica que tras una actualización o cambio en la base de datos realizada por un nodo, éste debe asegurarse de que el resto de nodos realicen dicha operación también y que la respuesta dada a una petición debe ser idéntica independientemente del nodo que responda.
- **Alta disponibilidad**: Cualquier petición recibida en un nodo del sistema debe obtener una respuesta, aunque los demás nodos del sistema fallen o simplemente no están disponibles.
- **Tolerancia al particionamiento**: Una petición debe ser procesada por el sistema incluso si se pierden mensajes entre alguno o todos los nodos del sistema o si se crean "islas de nodos" incapaces de entablar contacto entre ellas. En este punto también se incluye la capacidad del sistema de funcionar aún cuando se estén añadiendo o quitando nodos en ese momento.

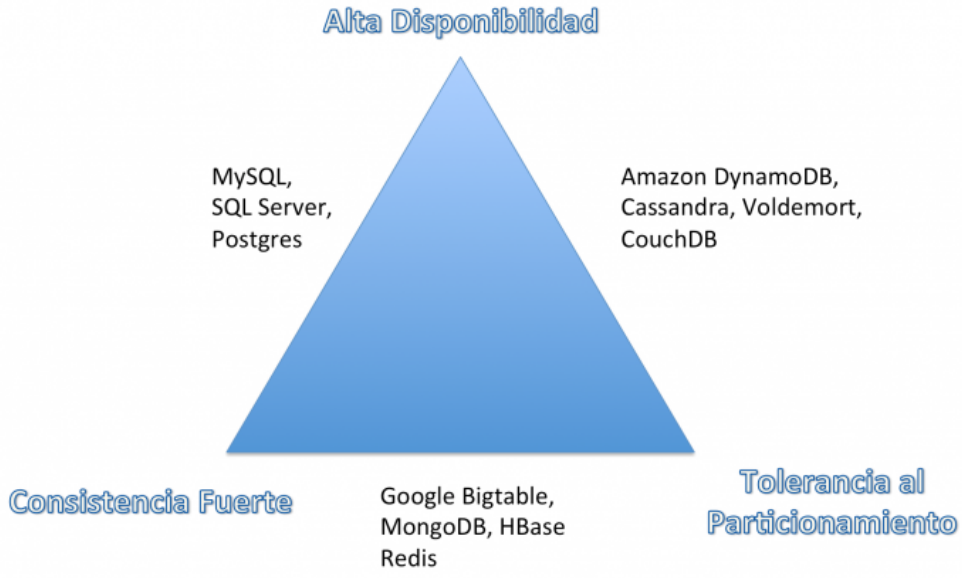

Figura 2.4: Triángulo CAP

Teniendo en cuenta que según el Teorema de Brewer una base de datos distribuida sólo puede cumplir con dos de estos atributos de manera satisfactoria, se pueden establecer grupos de estos sistemas de acuerdo a estas características. En la imagen 2.4 se pueden observar las 3 agrupaciones que conforman el triángulo CAP:

* CA: el sistema siempre responderá a las peticiones y los datos procesados serán consistentes. En este caso no se permite la partición del sistema.

* CP: el sistema ejecutará las operaciones de forma consistente, aunque se pierda la comunicación entre nodos (partición del sistema), pero no se asegura la disponibilidad.

* AP: el sistema siempre responderá a las peticiones, aunque se pierda la comunicación entre nodos. A cambio, los datos procesados pueden no ser consistentes.

Como puede observarse, en el grupo de sistemas que cumplen con los requisitos de disponibilidad y consistencia (CA) se encuentran los derivados del modelo relacional. En la parte de la derecha (AP) están aquellos inspirados en Amazon Dynamo, entre los que encontramos a Cassandra, y abajo (CP) los hijos de Google BigTable, con MongoDB entre ellos.
Esta clasificación resulta de mucha utilidad a la hora de tomar una decisión para el diseño de una plataforma para afrontar problemas relacionados con Big Data. Además, también pone de manifiesto lo pertinente que ha resultado la elección entre las dos bases de datos que vamos a estudiar, ya que desde el primer momento, y sin profundizar aún demasiado, estamos viendo que estas dos bases de datos poseen unas filosofías completamente diferentes.

2.2.1. Diferencias con las bases de datos relacionales: ACID Vs BASE

A lo largo de esta sección, en varios puntos se ha comentado que las propiedades ACID están fuertemente ligadas a las bases de datos relacionales, aunque no se ha explicado que significa dicho acrónimo ni por qué es importante en las bases de datos relacionales. Esto se explicará a continuación.

ACID

En informática y más concretamente dentro del mundo de las bases de datos, ACID es un conjunto de características o propiedades que garantizan que las transacciones en una base de datos sean **fiables**. En el contexto de bases de datos relacionales, una transacción es una o más sentencias SQL que se ejecutan como una secuencia de operaciones que forman una unidad lógica única de trabajo. Un ejemplo de una transacción sería la transferencia de fondos de una cuenta a otra, la cual implica múltiples operaciones individuales, probablemente en diferentes tablas.

En 1970, Jim Gray definió las propiedades que necesitaba tener una transacción confiable –Atomicidad, Consistencia y Durabilidad–, y desarrolló tecnologías para automatizarlas [19]. Más tarde, en 1983, Andreas Reuter y Theo Härder crearon el término "ACID" para describir estas 4 –incluyendo ya el aislamiento– propiedades [20].

1. **Atomicity** (Atomicidad): asegura que la operación se ha realizado con éxito. De esta forma se requiere que cada transacción sea "todo o nada" y que si una parte de la transacción falla, todas las operaciones de la transacción fallan, y por lo tanto la base de datos no sufre cambios.

2. **Consistency** (Consistencia): propiedad que se encarga de asegurar que sólo se va a comenzar aquello que se pueda finalizar. Dicho de otra forma, esta propiedad asegura que cualquier transacción llevará a la base de datos de un estado válido a otro estado válido, asegurando que cualquier dato que se escriba en la base de datos tiene que ser válido de acuerdo a todas las reglas que se hayan definido.

3. **Isolation** (Aislamiento): la ejecución de una transacción no puede afectar a otras. Más concretamente, se asegura que la ejecución concurrente de las transacciones resulte en un estado del sistema que

se obtendría si estas transacciones fueran ejecutadas una atrás de otra y de forma completamente independiente.

4. **Durability** (Durabilidad): cuando se realiza y confirma una transacción, nos asegura que ésta se almacenará de forma persistente, aún habiendo fallos en el sistema o incluso, cortes eléctricos.

BASE

El enfoque BASE, abandona las propiedades ACID de consistencia y aislamiento en favor de la disponibilidad, la "degradación elegante" y el rendimiento. El acrónimo BASE está compuesto por:

1. **Basically Availability**.

2. **Soft-state**.

3. **Eventual consistency**.

Bob Ippolito resume las características de BASE de la siguiente manera: Una aplicación funciona básicamente todo el tiempo pero no tiene que ser consistente todo el tiempo, ya que al final acabará alcanzando la consistencia entre los nodos en algún momento [21].

BASE es diametralmente opuesto a ACID. Donde ACID es pesimista y fuerza a que se alcance la consistencia al final de cada transacción, BASE es optimista y acepta que la consistencia de la base de datos acabará llegando con el tiempo. Este enfoque puede parecer un absoluto disparate, y más para usuarios acostumbrados a las bases de datos relacionales, pero lo cierto es que resulta ser un enfoque muy manejable y que conduce a niveles de escalabilidad y flexibilidad simplemente imposibles con ACID.

En la tabla 2.1, Brewer contrasta las propiedades de los dos enfoques, considerando los dos conceptos más como un espectro de posibilidades en lugar de alternativas excluyentes entre si. [22, 23]

ACID	BASE
Consistencia fuerte	Consistencia débil
Aislamiento	Disponibilidad primero
Centrar esfuerzos en "Commit"	Hacer el mejor esfuerzo (Best effort)
Transacciones anidadas	Respuestas aproximadas
Conservador (pesimista)	Agresivo (optimista)
Difícil evolución	Fácil evolución
¿Disponibilidad?	Sencillo
	Rápido

Tabla 2.1: Características ACID vs. Base

Las diferencias conceptuales y paradigmáticas de ACID y BASE ponen de manifiesto lo diferentes –y en ocasiones, prácticamente opuestas– que resultan ser las bases de datos relacionales (SQL) y las NoSQL, pero desde luego no son las únicas. A continuación se listan algunas de las diferencias más destacables que nos podemos encontrar entre los sistemas NoSQL y los sistemas SQL [24]:

- **No utilizan SQL como lenguaje de consultas**. La mayoría de las bases de datos NoSQL evitan utilizar este tipo de lenguaje o de utilizarlo, lo emplean más como un lenguaje de apoyo. Por poner algunos ejemplos, Cassandra utiliza el lenguaje CQL, MongoDB utiliza JSON y BigTable hace uso de GQL.

- **No utilizan estructuras fijas** (tablas) **para el almacenamiento de los datos**. Permiten hacer uso de otros tipos de modelos de almacenamiento de información como sistemas de clave–valor, objetos o grafos.

- **No suelen permitir operaciones JOIN**. Al disponer de un volumen de datos tan extremadamente grande suele resultar deseable evitar los JOIN. Esto se debe a que, cuando la operación no es la búsqueda de una *clave*, la sobrecarga puede llegar a ser muy costosa. Las soluciones más directas consisten en desnormalizar los datos, o bien realizar el JOIN mediante software, en la capa de aplicación.

- **Arquitectura distribuida**. Las bases de datos relacionales suelen estar centralizadas en una única máquina o bien poseer una estructura distribuida de tipo maestro–esclavo, sin embargo, la naturaleza de NoSQL posibilita el normal almacenamiento de los datos de manera distribuida.

2.2.2. Tipos de bases de datos NoSQL

Hablar de bases de datos NoSQL es hablar de estructuras que nos permiten almacenar información en aquellas situaciones en las que las bases de datos relacionales generan problemas debido principalmente a cuestiones de escalabilidad y rendimiento de las bases de datos relacionales, donde se dan cita miles de usuarios concurrentes y con millones de consultas diarias.

Además de lo comentado anteriormente, las bases de datos NoSQL son sistemas de almacenamiento de información que no cumplen con el esquema entidad–relación. Tampoco utilizan una estructura de datos en forma de tabla donde se van almacenando los datos sino que para el almacenamiento hacen uso de otros formatos muy diversos.

Dependiendo de como se almacene la información, podemos encontrarnos con 4 grandes tipos de bases de datos NoSQL bien diferenciados entre si [24, 25]:

Clave-Valor

Estas bases de datos guardan la información como una colección de pares de valores, en donde hay una clave y un valor. El valor puede ser tan complejo como se quiera y funciona como una "caja negra", es decir, no se pueden hacer consultas filtrando por lo que esté en esa parte, todas las consultas que se pueden realizar son filtrando los valores de la clave o *Key*. que debe ser única.

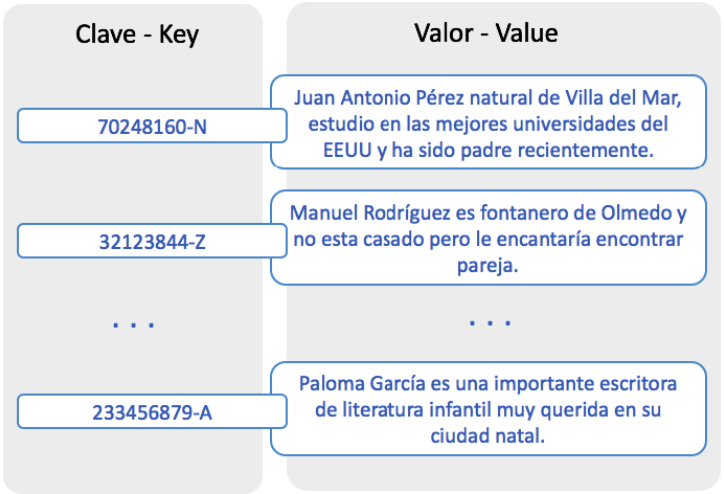

Figura 2.5: Ejemplo de almacenamiento de información con un esquema de clave-valor

Este tipo de BDs tienen un alto rendimiento realizando operaciones CRUD (Create, Read, Update y Delete), pero con consultas que requieren JOINs tienen un desempeño pobre.
Las bases de datos del tipo Key-Value garantizan la atomicidad a nivel de Clave, es decir las operaciones que se realizan sobre una clave específica son atómicas.

Una de sus principales ventajas es que son escalables horizontalmente, lo cual quiere decir que para mejorar el rendimiento o para manejar más cantidad de datos, se pueden agregar más computadoras (de bajo costo), pues es una BD distribuida. Algunos casos de uso en donde las BDs de Key-Value pueden ser usadas son:

- Información de sesión para aplicaciones Web.
- Perfiles y preferencias de usuarios.
- Carrito de compras para mercados/tiendas on-line.

Por el contrario, las BDs de clave-valor tienen un peor desempeño cuando los datos están interconectados entre sí, con gran cantidad de relaciones, como por ejemplo, en las redes sociales.

La base de datos NoSQL más famosa que se construye en un almacén de valores clave Key-Value es DynamoDB de Amazon.

Documental

Estas bases de datos guardan los datos en documentos individuales. Por lo general estos documentos son guardados en formatos bien conocidos como XML, JSON o BSON, entre otros. Son auto descriptivos y tienen estructuras jerárquicas. Además son similares a las BDs de Key-Value, solo que en este caso se pueden hacer consultas sobre los valores de todos los datos que contienen.

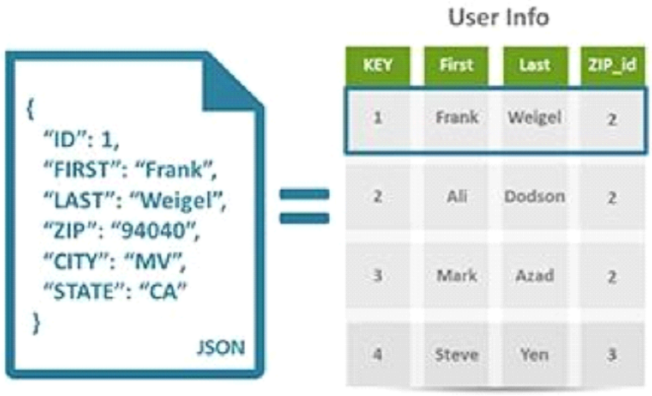

Figura 2.6: Ejemplo de almacenamiento de información en una base de datos documental.

Las operaciones son atómicas a nivel de documento y son escalables horizontalmente. Algunos casos de uso para este tipo de BDs:

- Registro de eventos.
- Blogs y otras aplicaciones Web.
- Metadatos de aplicaciones Web y Móviles.

Este tipo de bases de datos tienen su peor rendimiento cuando se requiere que haya consistencia en operaciones que involucran a múltiples documentos. Tampoco se deberían usar cuando los datos se encuentran naturalmente normalizados, con muchas relaciones entre los datos.

La aplicación de base de datos más popular y conocida que se basa en un almacén de documentos, es MongoDB.

Orientadas a columnas

En este tipo de bases de datos, los datos se almacenan en columnas, en lugar de almacenarse en filas (como se hace en la mayoría de los sistemas de gestión de bases de datos relacionales).

Un almacén de columnas está compuesto por una o más familias de columnas que se agrupan de forma lógica en determinadas columnas en la base de datos. Una clave se utiliza para identificar y señalar a un número de columnas en la base de datos. Cada columna contiene filas de nombres o tuplas, y valores, ordenados y separados por comas.

Las filas en cada familia de columnas no requieren tener las mismas columnas, por lo que las columnas pueden ser agregadas a una sola fila o a todas las filas de la familia de columnas.

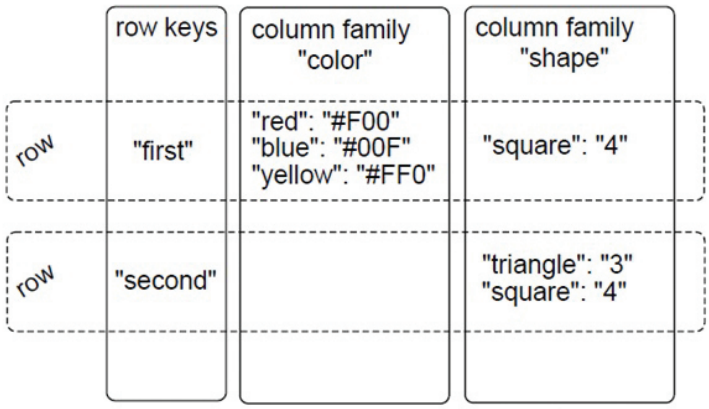

Figura 2.7: Ejemplo de almacenamiento de información con una base de datos orientada a columnas.

Las bases de datos orientadas a columnas tienen acceso rápido de lectura y escritura a los datos almacenados. En un almacén de columnas, las filas que corresponden a una sola columna se almacenan como una sola entrada de disco, lo cual facilita el acceso durante las operaciones de lectura y escritura. Dicho de otra forma, una base de datos en columnas está optimizada para lograr una recuperación rápida de columnas de datos, normalmente en aplicaciones analíticas.

La atomicidad de las operaciones se garantiza solo a nivel de una fila. Algunos casos de uso en los que se pueden usar este tipo de BDs son:

- Registro de eventos.
- Blogs.
- Expiración de uso de datos (TTL por columna o "Time To Live") y contadores.

Este tipo de bases de datos tienen su peor rendimiento cuando se necesita que las operaciones de lectura y escritura sean ACID.

Las bases de datos más populares que usan el almacén de columnas incluyen Google BigTable, HBase y Cassandra.

Grafos

Este tipo de bases de datos guardan los datos como entidades y relaciones entre las entidades. Las entidades son llamadas *nodos* y las relaciones, que unen los nodos, son llamados *bordes o aristas* y juntos conforman una representación gráfica de los datos. Están más orientadas a las relaciones que a las agregaciones, puesto que se puede emplear la teoría de grafos para recorrer la base de datos usando las relaciones entre los nodos, de forma que sea un proceso rápido y eficiente.

Este tipo de bases de datos escala verticalmente, a diferencia de los otros tipos de BDs NoSQL, y también pueden garantizar operaciones ACID. Algunos casos de uso en los que se pueden usar este tipo de BDs son:

- Datos altamente interconectados, como por ejemplo, redes sociales.
- Aplicaciones de "routing" o de ubicación espacial.

Las *Graph Database* tienen su peor desempeño cuando existen muchas actualizaciones de los datos de

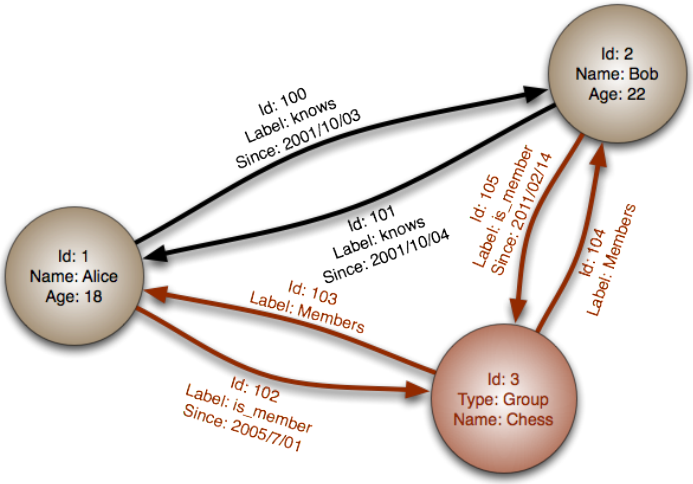

Figura 2.8: Ejemplo de almacenamiento de información con un esquema orientado a grafos.

los nodos. Tampoco se deben usar en aplicaciones que necesiten crecer demasiado en tamaño y requieran escalar horizontalmente.

Actualmente, Neo4J, InfoGrid e InfiniteGraph son las bases de datos gráficas más populares.

Estos 4 tipos de bases de datos NoSQL son los principales exponentes de las bases de datos NoSQL, y aunque en general resulta sencillo dirimir si una base de datos pertenece a un tipo u otro, esta diferenciación no siempre resulta clara en algunos casos. Por ejemplo, en relación a Cassandra; dependiendo de los autores o textos consultados, esta base de datos aparece clasificada como una base de datos orientada a columnas, una base de datos de clave-valor o ambas a la vez.

También es importante decir que estos 4 tipos, si bien son los más conocidos y empleados, no son los únicos, ya que podemos encontrar bases de datos multimodelo, multidimensionales, orientadas a objetos, orientadas a la *nube* o a eventos, entre otros, que cubren necesidades muy específicas y tienen una escasa implantación [26].

2.2.3. Escalabilidad horizontal

Anteriormente se ha comentado que al enfrentar a los sistemas SQL de bases relacionales con el big data, surge un importante problema de escalabilidad. La escalabilidad se refiere a la capacidad que tiene un sistema para adaptarse al crecimiento y a las nuevas necesidades. Por ejemplo, el desarrollo de un sistema puede comenzar con una base de 50 usuarios que tendrán requerimientos para cumplir con esta carga de trabajo, pero con el tiempo el sistema puede crecer a 50,000 usuarios dejando muy atrás los requerimientos de recursos que se plantearon inicialmente.

La escalabilidad no sólo se presenta por el crecimiento de usuarios, también puede crecer la complejidad del sistema, por ejemplo, si en un comienzo un sistema hipotético sólo contemplaba funciones de publicaciones de texto, quizás en un futuro se agreguen chat, búsquedas, reportes, etc. En este escenario, el sistema debe poder escalar para adaptarse a los nuevos requisitos. Pero en el contexto en el que nos encontramos, cuando nos referimos a la escalabilidad, estamos haciendo especial hincapié en el **rendimiento**.

Existen muchas maneras, tanto software como hardware, para tratar de mejorar la escalabilidad de un sistema o un software y conseguir que éstos se adapten a los nuevos requerimientos, como por ejemplo optimizar los algoritmos, añadir más memoria o capacidad de disco, añadir discos de estado sólido (SSD), mejorar la refrigeración, emplear arquitecturas de red... Esto da pie a que haya dos tipos de escalamiento bien diferenciados, que además poseen unas filosofías completamente opuestas y que se pueden ver en la imagen 2.9.

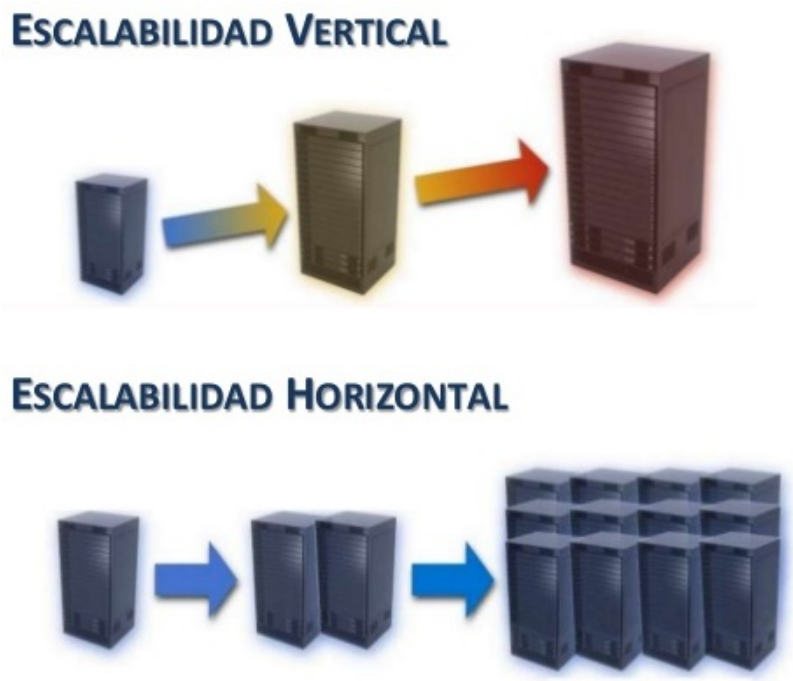

Figura 2.9: Escalabilidad vertical Vs. Escalabilidad horizontal.

En primer lugar tenemos la **escalabilidad vertical**, que a priori resulta más simple, pues implica cambiar el hardware por uno más potente. Esto podría lograrse cambiando componentes sueltos del servidor, como el disco duro, la memoria RAM, el procesador, etc. pero también puede ser la migración completa de un sistema a otro servidor con un hardware más potente.

El esfuerzo de este crecimiento es mínimo, pues apenas tiene repercusiones en el software ni hay que realizar modificaciones, ya que el proceso se puede resumir en respaldar y migrar los sistemas al nuevo hardware. Por el contrario, el crecimiento del sistema estará limitado completamente por el hardware que podamos adquirir –y de que exista realmente un hardware superior que pueda mejorar el rendimiento–, lo cual además podría tener un inmenso coste económico.

Por la otra parte, existe una filosofía diametralmente opuesta al crecimiento vertical: la **escalabilidad horizontal**, que es sin duda más potente, aunque también más complicada de implementar. Este modelo implica tener varios servidores (conocidos como *nodos*) trabajando como un todo. De esta manera se crea una red de servidores conocida como *clúster*, con la finalidad de repartirse el trabajo entre todos los nodos. Cuando se hace necesario incrementar el rendimiento del clúster, simplemente se añaden nuevos nodos.

Con el escalamiento horizontal nos encontramos en una situación en la que el crecimiento es prácticamente ilimitado, ya que sólo depende de la cantidad de servidores que podamos añadir al clúster. Al disponer de varios nodos trabajando juntos, si uno de ellos fallase, los demás podrían seguir trabajando, y además también permite que se balancee la carga de trabajo entre ellos.

Este modelo puede tener algunas desventajas, ya que una aplicación que no fuese diseñada originalmente para trabajar en clúster, podría necesitar grandes cambios para adaptarla. También es un modelo que requiere de mucha configuración y mantenimiento.

En cualquier caso, si bien estos tipos de escalamiento poseen una filosofía muy diferente, no son mutuamente excluyentes, y es que nada impide mejorar los elementos hardware (escalabilidad vertical) de los elementos de un clúster que ya está funcionando de acuerdo a los principios de la escalabilidad horizontal [27].

2.2.4. Ventajas y Desventajas

Esta forma de almacenar la información propia de las bases de datos NoSQL ofrece ciertas ventajas sobre los modelos relacionales. Entre las ventajas más significativas podemos destacar:

- **Se ejecutan en máquinas con pocos recursos**: estos sistemas no necesitan unas gran potencia de cómputo, por lo que se pueden montar en máquinas de un coste más reducido.
- **Escalabilidad horizontal**: mejorar el rendimiento de estos sistemas tiende a ser tan sencillo como añadir más nodos, con la única operación de indicar al sistema cuáles son los nuevos nodos que están disponibles.
- **Pueden manejar gran cantidad de datos**: esto es debido a que utilizan una estructura distribuida, en muchos casos mediante tablas Hash. También pueden emplear modelos y estructuras de datos mucho más flexibles.
- **No generan cuellos de botella**: probablemente el principal problema de los sistemas SQL radique en que necesitan transcribir cada sentencia para poder ser ejecutada, y cada sentencia compleja requiere además de un nivel de ejecución aún más complejo, lo que ante muchas peticiones puede ralentizar el sistema de forma muy significativa.
- **Son de código abierto**: si bien no todas, la mayoría de las bases de datos nosql que podemos encontrarnos son open source. El hecho de no tener que pagar una licencia, unido a que podemos utilizar máquinas con pocos recursos, implica también que **son económicamente rentables**.

Evidentemente no todo son ventajas y no en todas las situaciones son aplicables las bases de datos NoSQL. A continuación se listan algunas de las **principales desventajas** que podemos encontrarnos al emplear este tipo de sistemas.

- La mayoría de las bases de datos NoSQL **no admiten funciones de "fiabilidad"**, que sí son soportadas por sistemas de bases de datos relacionales (ACID). Esto también significa que las bases de datos NoSQL que no soportan esas características, sacrifican la consistencia en favor del rendimiento y de la escalabilidad.
- La implementación de las características de fiabilidad y coherencia debe ser realizada por los desarrolladores de la aplicación que emplee la base de datos, dentro de su propio código, lo que **incrementa mucho la complejidad del sistema**.
    - Esto a su vez podría **limitar el número de aplicaciones en las que podemos confiar** para realizar transacciones seguras y confiables, como por ejemplo los sistemas bancarios.
- **Problemas de compatibilidad con SQL**. Las nuevas bases de datos utilizan sus propias características en el lenguaje de consulta y no son 100 % compatibles con el SQL de las bases de datos relacionales. El soporte a problemas con las *queries* de trabajo en una base de datos NoSQL es más complicado y en determinados casos acaba siendo un proceso mucho más lento y complejo.
- A su vez, no hay que olvidar que al no usar un lenguaje de consulta estándar como SQL, cada sistema gestor de bases de datos nosql posee un lenguaje de consulta propio. Esta **falta de estandarización** supone dificultades a la hora de encontrar personal que pueda manejar estas bases de datos o problemas si se decidiese migrar de un sistema de bases de datos a otro. Dicho de otra manera, el uso de sistemas relacionales SQL está más adaptado y el perfil profesional necesario para emplearlos es mayoritario.

- **Hay menos información y apoyo disponible**. Por varios motivos, además. Uno de ellos es que las bases de datos NoSQL son mucho más nuevas que las relacionales, el otro motivo principal es nuevamente la falta de estandarización, ya que la solución a un problema en un sistema NoSQL difícilmente sería aplicable a otros sistemas NoSQL.

- **Escasez de plataformas y herramientas**. Existen muchas bases de datos NoSQL, y muchas de ellas no pueden funcionar fuera de un sistema Linux y/o cuentan con herramientas de administración poco usables.

Además, en la tabla 2.2 podemos ver las principales diferencias entre algunas de las características más importantes en las bases de datos.

Característica	**Bases de datos NoSQL**	**Bases de datos relacionales**
Rendimiento	Alto	Bajo
Confiabilidad	Baja	Alta
Disponibilidad	Buena	Buena
Consistencia	Baja	Alta
Almacenamiento de datos	Optimizadas para grandes volúmenes de datos	De volúmenes medianos a grandes
Escalabilidad	Alta (Horizontal)	Alta (Vertical)

Tabla 2.2: Principales diferencias entre las características de las bases de datos relacionales y NoSQL.

Tras ver los *pros y contras* de las bases de datos NoSQL, podríamos dudar de lo acertado de emplear estas bases de datos según en que circunstancias estemos. Como se ha visto, estos sistemas ofrecen unas posibilidades de rendimiento, flexibilidad y escalabilidad que de ninguna manera podríamos encontrar en sistemas relacionales, por la otra parte, la solución de *consistencia eventual* puede llevar a problemas de fiabilidad o coherencia de los datos.

De esta forma, se puede aseverar que los sistemas NoSQL no son adecuados cuando la consistencia de los datos y la imposibilidad de error sean requisitos fundamentales del sistema. Si no se tiene ese requisito, entonces NoSQL es una posibilidad a tener en cuenta, especialmente si no se dispone de un gran presupuesto, hay que tratar con estructuras de datos variables o hay que analizar (leer, concretamente) grandes cantidades de datos [28, 29].

Capítulo 3

MongoDB y Cassandra

En el capítulo anterior hemos podido ver con bastante profundidad las particularidades de NoSQL y su vital participación dentro del mundo del *Big Data*, así como las diferencias que presenta este nuevo modelo en relación con los sistemas clásicos relacionales y las posibles ventajas o inconvenientes que podrían aportar o surgir.

También en el capítulo anterior se ha empezado a vislumbrar, de una forma introductoria, alguna de las características y particularidades más interesantes de los dos sistemas de gestión de base de datos objetivos de este trabajo. En este capítulo se profundizará aún más en MongoDB y Cassandra; en su filosofía, el funcionamiento interno y sus implicaciones en el rendimiento, así como su evolución a lo largo de los años y los cambios o adaptaciones que hayan realizado para adaptarse a su "nicho de mercado".

3.1. MongoDB

MongoDB es un sistema de base de datos multiplataforma orientado a documentos de esquema libre. Esto significa que cada entrada o registro puede tener un esquema de datos diferente, con atributos que no tienen por qué repetirse de un registro a otro.

Fue desarrollado originariamente en 2007 por una empresa americana llamada *10gen* siendo originalmente una parte de un sistema de su propiedad. En 2009 esta compañía decidió liberar MongoDB y centrarse en su desarrollo, dejando a un lado el sistema del que formaba parte. Posteriormente, la empresa cambió su nombre por *MongoDB Inc.*, confirmando así que iba a centrar esfuerzos únicamente en desarrollar su *buque insignia*.

Es de código abierto y podemos encontrarlo en la plataforma GitHub [30]. Está escrito en C++ –principalmente–, C y JavaScript, de forma que resulta ser muy eficiente a la hora de ejecutar sus tareas, y además, está licenciado como una combinación de "GNU Affero General Public License" (GNU AGPL 3.0) y Apache License v2, de modo que se trata de un software de licencia libre, aunque también dispone de licencias comerciales propias (MongoDB, Inc.) para otros de sus productos.

Funciona y está soportado en la mayoría de los sistemas operativos más usados en el mundo, incluyendo Microsoft Windows (a partir de Windows Vista) y macOS de Apple, aunque es en Linux donde más destaca, pues está disponible para múltiples distribuciones; Amazon Linux, Debian, Ubuntu, SUSE y RHEL (Red Hat Enterprise Linux).

En este aspecto de los sistemas soportados, hay un par de apuntes históricos o curiosidades reseñables. El primero de ellos es que actualmente MongoDB sólo está soportado en sistemas operativos de 64 bits. Esto es debido a que se sabía desde 2009 que las versiones de 32 bits tenían un importante problema a la hora de manejar más de 2GB de datos [31], lo que llevó a la decisión de enfocar el software únicamente

para los sistemas operativos de 64 bits. La segunda curiosidad es mucho más reciente, ya que a finales de 2017 el equipo de MongoDB decidió abandonar el soporte para Solaris, que había sido uno de los primeros sistemas soportados por este gestor de bases de datos [32].

3.1.1. ¿Quién usa MongoDB?

En el primer capítulo se hacía bastante hincapié en la popularidad de MongoDB, ya que está considerado como el sistema gestor de bases de datos NoSQL más popular en la actualidad. Esto plantea a su vez otras preguntas, quizás más importante que el propio dato sobre la popularidad de este software: ¿Quién lo usa y para qué?
Ciertamente, MongoDB tiene un importante nicho de mercado en pequeñas y medianas empresas que necesitan un tratamiento de datos potente y flexible. Pero es en las grandes empresas done podemos ver realmente el potencial de esta base de datos, ya que la lista de estas grandes empresas que lo usan, a parte de ser extensa, resulta ser también bastante impresionante.

Plataformas de redes sociales como LinkedIn o Facebook. Empresas ligadas al mundo de los videojuegos como SEGA o Square Enix. Lineas aéreas como Air France o agencias de viajes como Expedia. Empresas de telecomunicaciones como Telefónica y medios de comunicación como el Washington Post o Forbes. Y algunas otras grandes empresas tecnológicas presentes en todo el mundo como Ebay, Google, Adobe, NOKIA, Bosch, McAfee o Cisco, así como el CERN (Organización Europea para la Investigación Nuclear) que utiliza MongoDB para manejar los grandes volúmenes de datos que genera el acelerador de partículas, son sólo algunos de los ejemplos más contundentes de los clientes de MongoDB [33].

La ciudad de Chigago merece una mención a parte, ya que quizás sea uno de los ejemplos más curiosos que podemos encontrar. Recientemente ha desarrollado una plataforma de operaciones inteligente llamada *WindyGrid* usando MongoDB. WindyGrid reúne a diario millones de piezas de datos de diferentes departamentos de la ciudad, creando una especie de sistema nervioso central para la ciudad, cuyo objetivo es ayudar a mejorar los servicios, reducir costos y crear una ciudad más habitable.

Además, Mongo colabora estrechamente con socios como IBM o Red Hat, grandes empresas dentro del mundo de las tecnologías de la información, entre otras muchas.

En cuanto a la pregunta de para qué emplean estas compañías MongoDB, hay que decir que muchas de ellas no lo han publicado o que simplemente conforma una de las muchas partes de su complejo sistema. En cualquier caso, hay compañías que si han explicado para qué están usando MongoDB, y en general sus usos recaen en alguno de los siguientes grupos [34]:

1. Visión unificada del conjunto: aseguradoras como MetLife o aerolíneas como Air France emplean esta base de datos para crear un repositorio central que recopila la información de sus muchos usuarios, proveniente de muchas fuentes de datos, para crear una visión completa de los mismos.

2. Internet de las cosas: en el mundo de IoT tenemos muchos dispositivos conectados y generando y compartiendo información, por eso, bases de datos nosql como MongoDB son escogidas por muchas empresas para llevar a cabo sus proyectos de IoT.

3. Tecnologías móviles: cada vez son más las compañías que están empezando a apostar por MongoDB como tecnología backend en el mundo de los smartphones.

4. Análisis en tiempo real: el sistema WindyGrip de la ciudad de Chicago resulta un buen ejemplo de la necesidad cada vez mayor de conseguir resultados de manera inmediata y de la implantación de MongoDB para lograrlo.

5. Personalización: buscan crear y ofrecer experiencias hechas a la medida de los usuarios en tiempo real. Para ello es necesario hacer un análisis rápido y certero del perfil de usuario, comportamiento, datos demográficos, gustos, etc... Expedia es un buen ejemplo del uso de MongoDB en este aspecto.

6. Administración de contenido: permiten gestionar archivos junto a sus metadatos. Forbes construyó todo su sistema de gestión de contenidos en MongoDB. Además, lo utiliza para analítica en tiempo real, de forma que cuando algún artículo se hace viral, Forbes detecta la forma en que se está compartiendo entre los usuarios y de este modo sabe qué tipo de contenido le debe ofrecer a sus lectores.

3.1.2. Características

Las características que más destacan los usuarios de MongoDB son su velocidad y su rico pero sencillo sistema de consulta de los contenidos de la base de datos. Es más, en su página web podemos ver que lo definen de la siguiente manera: *"MongoDB es una base de datos documental con la escalabilidad y flexibilidad que quieres, con las consultas y el indexado que necesitas"* [35].
Específicamente, se puede decir que las principales características de este SGBD son las siguientes:

- MongoDB almacena los datos en documentos similares a JSON, lo que significa que los campos pueden variar de un documento a otro y la estructura de los datos se puede cambiar con el tiempo.
- El modelo de base de datos orientada a documentos permite mapear cómodamente los objetos de las aplicaciones que usen la base de datos, facilitando el trabajo con los datos y la forma de almacenarlos.
- Este modelo también resulta sencillo de aprender y utilizar para los desarrolladores, facilitando la migración hacia este sistema. También disponen de numerosos drivers, con licencia Apache v2, para un gran número de lenguajes de programación muy diversos.
- Las consultas Ad-Hoc, la indexación y la agregación en tiempo real brindan formas muy potentes de acceder y analizar los datos.
- Es una base de datos distribuida en *su núcleo*, por lo que la alta disponibilidad, el escalado horizontal y la distribución geográfica están incorporados prácticamente por defecto y son fáciles de emplear.
- Es gratuito y de código abierto, publicado bajo la Licencia Pública General Affero de GNU.
- En la versión más reciente de MongoDB, la versión 4.0, se ha añadido el soporte de transacciones ACID en múltiples documentos, por lo que según sus desarrolladores, es la única base de datos de código abierto que combina la velocidad, flexibilidad y potencia del modelo orientado a documentos con las garantías de ACID. A través del "aislamiento de instantáneas" (snapshot isolation), las transacciones proporcionan una visión coherente de los datos y hacen cumplir la ejecución de todo o nada para mantener la integridad de los datos.

En esta lista se pueden ver algunas características que ya se han comentado en este documento y también han salido a relucir algunas nuevas que se comentarán en los siguientes puntos, pero hay una con una importancia vital, y es la última listada: *multi-document ACID* o soporte para transacciones.

En las versiones de MongoDB anteriores a la versión 4.0 sí se podían ver satisfechas las propiedades ACID, pero sólo dentro de un mismo documento. Como se ha visto en el capítulo anterior, el no implementar las propiedades ACID en toda la base de datos implica que ésta no asegure la durabilidad, la integridad, la consistencia y el aislamiento requeridos obligatoriamente en el concepto de transacción.

Esto podía crear diversos problemas; como problemas de consistencia, leyendo versiones obsoletas del documento o devolviendo *datos basura* de escrituras que no deberían haber ocurrido; problemas de bloqueos a nivel de documento al realizar escrituras concurrentes en el mismo documento; problemas de pérdidas de información, ya que las escrituras no eran durables ni verificables; y si la ausencia de ACID multidocumental podría provocar estos tres problemas, a su vez éstos podían provocar problemas de rendimiento y desempeño al enfrentarse a cantidades de información muy altas.

Algunos de estos problemas estuvieron presentes hasta hace no mucho tiempo y se consideraban como un *mal menor o simplemente inevitables*, otros se solucionaron con la versión 3.0 del software, pero en cualquier caso, todos ellos se verían solventados con la completa implantación de ACID. Y llama mucho la atención que ACID resultase ser la solución elegida para una base de datos NoSQL, ya que, como se ha visto en el capítulo 2, este paradigma tiende a huir de las obligaciones de ACID en favor de la libertad y flexibilidad de BASE.

Desde luego, la evolución vista hasta ahora de esta plataforma demuestra su firme intención de competir abiertamente con los sistemas relacionales clásicos, incluso llegando a adquirir muchas de sus características y acercándose más a ellos, y su posicionamiento como una base de datos de propósito general, en vez de buscar un nicho de mercado más orientado a soluciones particulares y concretas, como hacen otras bases de datos NoSQL.

Estructura de documentos: BSON

MongoDB es una base de datos orientada a documentos, esto quiere decir que un documento es la unidad básica de datos que podremos encontrar. Los documentos son análogos a los objetos JSON (JavaScript Object Notation) pero se almacenan en la base de datos en una representación binaria de JSON, más rica en tipos de datos, conocida como BSON.

Los documentos de MongoDB están compuestos por pares de campos y valores separados por comas, y tienen la siguiente estructura:

```
{
   campo1: valor1,
   campo2: valor2,
   campo3: valor3,
   ...
   campoN: valorN
}
```

En un documento se pueden agregar, eliminar, modificar o renombrar nuevos pares de campo-valor en cualquier momento. El valor de un campo puede pertenecer a cualquiera de los tipos de datos BSON [36], incluidos otros documentos, arrays e incluso arrays de documentos. El siguiente ejemplo proporcionado en la documentación de MongoDB especifica la manera de incluir otros documentos o arrays, así como tipos de datos complejos como fechas:

```
{
   _id: ObjectId("5099803df3f4948bd2f98391"),
   name-documento: {
        first: "Alan",
        last: "Turing"
                    },
    birth: new Date('Jun 23, 1912'),
```

```
    death: new Date('Jun 07, 1954'),
     contribs-array: [ "Turing machine", "Turing test", "Turingery" ],
    views : NumberLong(1250000)
}
```

Una colección es una agrupación de documentos de MongoDB, y a su vez, una base de datos está compuesta por colecciones de documentos. Se podría entender el concepto de colección como una aproximación a una tabla de un sistema relacional, con la principal diferencia que las colecciones no imponen un esquema, por lo que los documentos dentro de una colección pueden tener diferentes campos. Típicamente, todos los documentos en una colección tienen un propósito similar o relacionado por lo que es habitual que compartan el mismo número de campos o atributos.

Para crear una base de datos en MongoDB, una vez que el SGDB ha sido instalado y la Shell de Mongo está en ejecución, simplemente se le debe decir el nombre de la base de datos que se quiere usar. Si esa base de datos aún no existe, se creará automáticamente al crear una colección y añadirle el primer documento. De igual manera, una colección se crea automáticamente al almacenar un documento en ella:

```
use BD_TFG
db.coleccion_TFG.insert( { x: 1 } )
```

De esta forma, se ha indicado el nombre de la base de datos que queremos usar (BD_TFG) y el nombre de la colección (coleccion_TFG) pero ni la base de datos ni la colección estarán realmente creadas hasta el momento en que se ejecuta la sentencia .**insert()**, que es la que añade el documento y propicia la creación de la colección y de la base de datos.

El documento que se ha insertado solo debería tener un campo (x) y un valor (1), sin embargo, durante la inserción, mongod creará el campo **_id** y le asignará un valor de tipo **ObjectId** único y específico para cada máquina y momento en que se ejecuta la operación.
El campo _id puede ser añadido con un valor concreto, igual que si se tratase de otro campo, pero hay que entender que está reservado para su uso como clave principal; su valor debe ser único en la colección, es inmutable y no puede ser un array.

Al igual que para la inserción mostrada anteriormente, MongoDB usa la notación por puntos para acceder a los elementos de un array o a los campos de un documento embebido dentro de otro.
Para acceder a un elemento específico de un array se puede indicar la posición o índice (comenzando a contar la primera posición como 0) del elemento deseado sobre el array, indicando su nombre. Para el ejemplo de Alan Turing visto anteriormente, si quisiésemos acceder a la tercera posición del array se debería emplear la siguiente notación: *contribs-array.2*.
El caso de los documentos dentro de un documento es análogo, pero empleando el nombre del campo en vez de su posición. En ese mismo ejemplo anterior se emplearía la siguiente notación para acceder al nombre: *name-documento.first*.

3.1.3. Consultas

Soporta consultas *Ad Hoc*, es decir aquellas en las que los usuarios pueden personalizar las consultas en tiempo real y buscar cualquier elemento que deseen, sin ajustarse a, por ejemplo, consultas prediseñadas para generar informes. MongoDB soporta la búsqueda por campos, rangos y expresiones regulares. Las consultas pueden devolver campos específicos de documentos y también incluyen funciones de JavaScript definidas por el usuario.

Para realizar estas consultas, emplea el método find() aplicado sobre la colección deseada, de forma que si suponemos que la colección *coleccion_TFG* almacena los documentos de ejemplo empleados, se podría emplear la sentencia *db.coleccion_TFG.find( )* para buscar todos los documentos en la colección. Para buscar por un valor específico, se haría de la siguiente forma: *db.coleccion_TFG.find(campo1: valor1)*. El método find() no es el único método existente para realizar una consulta, otros muy usados son findOne() para que la consulta sólo devuelva un documento, aunque haya más que coincidan con el criterio de búsqueda y findAndModify(), que permite realizar una consulta y posteriormente, actualizar los valores de uno o más campos.

Si bien la forma de realizar las consultas tiene este formato especial, que en realidad es JavaScript, las consultas son equivalentes a las tradicionales SQL, tanto que poseen los mismos operadores y se aplican de una forma muy similar. $in, $eq, $and, $or son solo algunos de los ejemplos de los operadores que se pueden usar en MongoDB con una funcionalidad idéntica a los operadores de SQL, con la única diferencia que llevan el símbolo "$" delante. Así pues, la siguiente consulta devolvería los documentos que cumplan simultáneamente con los criterios de que el nombre sea Alan y el número de vistas sea mayor de 30:
db.coleccion_TFG.find("name-documento.first": "Alan", views: $gt: 30).

Nótese que la sintaxis de esta consulta no emplea el operador $and, ya que implícitamente incluye el operador lógico AND cuando se emplean consultas compuestas sin especificar un operador.

Indexado

Los índices son estructuras de datos especiales (árboles-B) que almacenan una pequeña parte del conjunto de datos de la colección en una forma fácil de recorrer. Un índice almacena el valor de un campo específico o conjunto de campos, ordenados por su valor. El orden de las entradas del índice admite coincidencias de igualdad eficientes y operaciones de consulta basadas en rango.

Los índices son los que realmente permiten la ejecución eficiente de las consultas en MongoDB. Sin índices, no queda más remedio que realizar un escaneo de la colección completa, es decir, escanear cada documento presente en una colección para seleccionar aquellos documentos que coincidan con la declaración de consulta. Si existe un índice apropiado para una consulta, MongoDB puede usar el índice para limitar el número de documentos que debe inspeccionar y optimizar así los tiempos.

Se puede afirmar que el concepto de índices es muy similar a los encontrados en bases de datos relacionales. MongoDB define los índices a nivel de colección y cualquier campo o sub-campo de un documento puede ser indexado, pudiéndose también incluir índices secundarios.

Este SGDB crea un índice único –y que no puede eliminarse– en el campo _id en el momento en que se crea la colección, de esta forma previene además que se inserten dos documentos con el mismo valor de id. Se puede crear un índice sobre un campo con la siguiente sentencia: *db.collection.createIndex(campo: indice)*, empleando el nombre de la colección para llamar a createIndex y el nombre del campo o los campos que queramos indexar. En este caso, el valor del índice puede ser 1 o -1, dependiendo de como se quieran ordenar los resultados. Con un valor 1 el índice los ordena de forma ascendente y con un valor -1 los ordena de una manera descendente.

Agregación

Las operaciones de agregación procesan y agrupan los valores de varios documentos y pueden realizar una variedad de operaciones en ellos para reducirlos y devolver un solo resultado.

MongoDB proporciona un framework que permite realizar operaciones similares a las que se obtienen con el comando SQL "GROUP BY". El framework de agregación está construido como un *pipeline* en el que los datos van pasando a través de diferentes etapas, durante las cuales son modificados, agregados, filtrados y formateados hasta obtener el resultado deseado. Durante todo este proceso se puede utilizar índices si

existieran, y además se produce en memoria. Asimismo, MongoDB proporciona una función *MapReduce* que puede ser utilizada para el procesamiento por lotes de datos y operaciones de agregación [37].

3.1.4. Naturaleza distribuida

Otra de las principales características de este sistema es su concepción como una base de datos distribuida. Presenta dos maneras de distribuir los datos entre varias máquinas, cada una de ellas para abordar un problema diferente: la replicación y el sharding.

Replicación

MongoDB, como muchos sistemas NoSQL, es considerablemente más flexible que las bases de datos relacionales, lo que puede ocasionar problemas de volatilidad de los datos ante caídas de los servidores o cortes eléctricos.

La replicación proporciona redundancia y aumenta la disponibilidad de los datos, ya que con múltiples copias de los datos en diferentes servidores aumenta el nivel de tolerancia a fallos. En algunos casos, la replicación además puede proporcionar una mayor capacidad de lectura, ya que los clientes pueden enviar operaciones de lectura a diferentes servidores. También se emplea para mantener copias para fines dedicados, como recuperación ante desastres, informes o copias de seguridad.

MongoDB soporta un tipo de replicación primario-secundario basado en *replica sets*. Un replica set es un conjunto de instancias o nodos de mongod que mantienen los mismos datos.

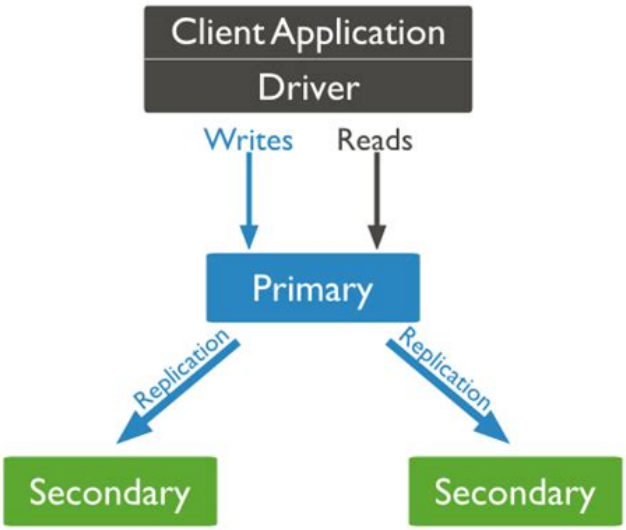

Figura 3.1: Replicación en MongoDB

Dentro de este replica set hay un nodo primario que recibe todas las operaciones, tanto de lectura como de escritura, y después de ejecutarlas, almacena todos los cambios que se han realizado en la base de datos dentro de un registro de operaciones. Los nodos secundarios utilizan dicho registro para replicar los datos y alcanzar el mismo estado que el nodo primario, la función por defecto de un nodo secundario es la de *copia de seguridad* aunque pueden ser configurados para responder a peticiones de lectura. Adicionalmente, si el nodo primario dejase de estar disponible, un nodo secundario puede ocupar temporal o permanentemente el puesto de nodo primario.

Existe un tercer tipo de nodos que puede estar presente en un replica set, los árbritros, aunque esta clase de nodo no almacena ni replica los datos, si no que tiene una responsabilidad de gestión interna, como la de determinar si un nodo está funcionando correctamente o la de elegir un nuevo nodo primario en caso de que el actual no esté disponible.

La replicación en MongoDB es sencilla de implementar y configurar, pero tiene algunas limitaciones. Por ejemplo, tras el *failover*, o lo que es lo mismo, la elección de un nodo secundario como nuevo nodo primario si este fallase, se requiere un tiempo durante el cual habrá problemas de disponibilidad. Por otra parte, la replicación no ayuda demasiado a aumentar la escalabilidad, ya que las escrituras siempre se hacen en el nodo primario. Además, compartir la carga de lectura con los nodos secundarios puede llevar implícita una pérdida de consistencia si éstos no están completamente actualizados.

Sharding

En MongoDB es natural complementar la replicación con el *sharding* o particionado de la información. De esta manera se pueden alcanzar los objetivos de alto rendimiento, capacidad de almacenamiento y escalabilidad horizontal necesarios para los sistemas Big Data.

El Sharding es una técnica que consiste en particionar los datos de una base de datos horizontalmente, agrupándolos de algún modo que tenga sentido y que permita un enrutado eficiente. Un *Sharded cluster* o clúster particionado de Mongo consta de los siguientes componentes:

1. Shard: cada partición contiene un subconjunto de los datos fragmentados. Cada fragmento se puede implementar como un conjunto de replicación.

2. Mongos: son instancias que actúan como enrutadores de consultas y operaciones de escritura, proporcionando una interfaz entre las aplicaciones cliente y el clúster de shards.

3. Servidores de configuración: los servidores de configuración almacenan metadatos y configuraciones para el clúster.

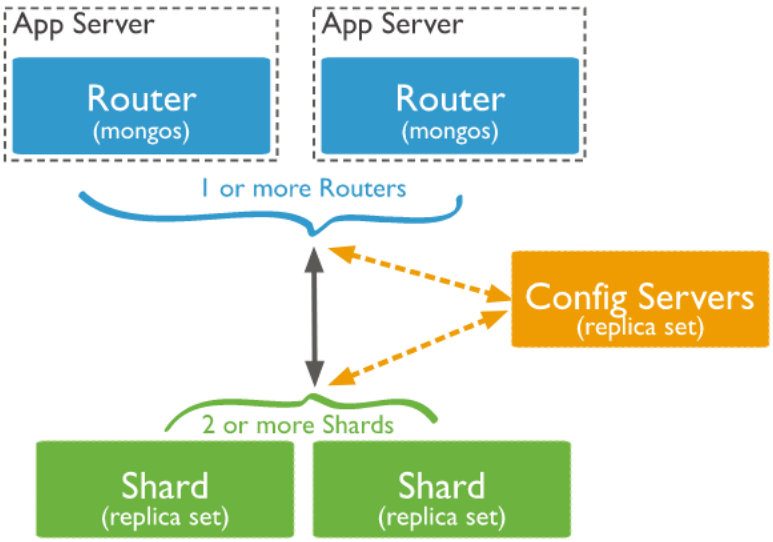

Figura 3.2: Sharding en MongoDB

La fragmentación de los datos se realiza a nivel de colección, distribuyendo los documentos entre las máquinas del clúster. Para distribuir los documentos emplea una "clave de partición", que es un campo o campos que existen en todos los documentos de la colección.

Los datos se parten en "chunks" –que se parten nuevamente si alcanzan un tamaño superior al *tamaño de chunk*, que por defecto tiene un valor de 64MB– y después migra los chunks dentro de los shards del clúster, procurando que exista un balance entre los distintos shards y los chunks que se almacenan en ellos.

Los *mongos* rastrean qué datos están en qué shard almacenando en caché los metadatos de los servidores de configuración, después los emplean para enrutar las operaciones de las aplicaciones y los clientes a las instancias de mongod.

3.1.5. Seguridad

Mención a parte merece el tema de la seguridad, pues la falta de **seguridad por defecto** ha sido uno de los aspectos más criticados a lo largo de la historia de MongoDB. Originalmente, y durante muchas versiones, estaba diseñado para que los nuevos usuarios pudiesen empezar a utilizar mongo lo más rápido posible. Si bien esto es un punto positivo, resulta que la configuración básica por defecto era también por defecto insegura.

Se podían hallar varios problemas, pero sin duda el mayor de ellos se encontraba en un parámetro de configuración llamado bind_ip y cuyo valor por defecto era 0.0.0.0 (todas las interfaces). Esta configuración permitía que todas las interfaces de red pudiesen establecer una conexión con la base de datos sin ningún tipo de limitación, y junto a otros fallos de seguridad como no establecer credenciales de autenticación, ha sido una de las brechas de seguridad más usadas para atacar a este tipo de bases de datos.

Aunque a partir de la versión 3.6 este parámetro por defecto se volvió más restrictivo, cambiando su valor a 127.0.0.1 (localhost), aún se pueden encontrar noticias sobre grandes ataques de ramsonware y robos de información sobre servidores MongoDB mal configurados [38, 39].

Los desarrolladores de MongoDB son conscientes de que la seguridad ha sido uno de los aspectos más criticados de su sistema y por ello en la documentación de su página web podemos encontrar una *checklist* [40] con pasos a seguir para que los administradores de servidores mongod puedan configurarlos de manera segura. En esta lista se encuentran los siguientes puntos, que si bien no se entrará en detalles sobre ninguno, dan una idea general del nivel de seguridad que se puede alcanzar en este SGDB:

- Habilitar el control de acceso y exigir la autenticación.
- Configurar el control de acceso basado en roles.
- Encriptar la comunicación, así como cifrar los datos.
- Limitar la exposición de la red (limitar las interfaces en las que se escuchan las conexiones entrantes).
- Auditar la actividad del sistema.
- Ejecutar MongoDB con un usuario dedicado y con opciones de configuración seguras.

3.2. Apache Cassandra

Cassandra, con un nombre directamente tomado de la mitología griega, se define como una base de datos NoSQL orientada a columnas, distribuida y masivamente escalable, pues tiene la capacidad de escalar linealmente. Además, introduce conceptos interesantes que otros sistemas nosql no poseen, como el soporte para *multi-data center* o la comunicación *peer-to-peer* entre sus nodos.

El desarrollo inicial de Cassandra tiene su origen en Facebook, que lo diseñó para responder al crecimiento tanto del número de consultas que se hacían a través de su web en tiempo real como del del volumen de los datos almacenados. En 2008 fue liberado como proyecto open source y en febrero de 2010 se convirtió en un proyecto *top-level* de la fundación Apache. Este desarrollo estuvo inspirado e influenciado por Amazon Dynamo y de Google BigTable, tratando de aunar lo mejor de ambos.

Actualmente, Cassandra es un proyecto libre y su código es accesible desde la plataforma GitHub [41]. Está escrito principalmente en Java y posee una licencia Apache License 2.0. Existen versiones comerciales de Cassandra, como las patentadas por la empresa DataStax, la cual ha estado involucrada desde casi el principio en el desarrollo de Cassandra. Según esta empresa, DataStax Enterprise 6 [42], que es la última versión lanzada del producto, es dos veces más rápida que la versión de código abierto de Cassandra y también posee una seguridad mucho más avanzada.

Cassandra tiene soporte multiplataforma, y puede ser usado en los sistemas operativos más comunes, incluyendo múltiples distribuciones Linux como Red Hat Enterprises, CentOS, Ubuntu o Debian, así como MacOs y Windows, tanto en las versiones de 64 bits como en las de 32 bits.

En este aspecto, hay que destacar una cosa, y es que durante bastante tiempo la empresa DataStax además de proporcionar su propia versión de pago de Cassandra, también ofrecía una versión gratuita llamada *DataStax Distribution of Apache Cassandra^{TM} (DDC)* [43] más "amigable" para el usuario en algunos aspectos, sobre todo para la instalación en sistemas Windows, ya que ofrecía un instalador (MSI installer). Actualmente, la instalación de Cassandra en Windows resulta bastante complicada para buena parte de los usuarios [44], al tener que realizar la instalación desde línea de comandos en vez de utilizar los .exe o .msi que son habituales en este sistema operativo. Además, en la documentación oficial de Cassandra, no hay ninguna mención específica a la instalación en sistemas Windows, y la guía de instalación puede resultar insuficiente para usuarios que no estén acostumbrados a sistemas Linux.

3.2.1. ¿Quién usa Cassandra?

Cassandra es la base de datos orientada a columnas más popular en la actualidad, y una de las 10 bases de datos más populares, contando tanto las relacionales como las no relacionales.

Tiene un nicho de mercado considerable en pequeñas o medianas empresas que necesitan contar con una gran escalabilidad y una alta disponibilidad sin que el rendimiento se vea comprometido, sin embargo, resulta mucho más revelador conocer las grandes empresas que emplean esta base de datos y de qué manera lo hacen:

- Apple, que emplea 75.000 nodos de Cassandra para almacenar más de 10 PB de datos.
- Netflix emplea 2.500 nodos como base de datos de back-end para su servicio de streaming, dando más de un billón de respuestas a peticiones en un día.
- El CERN lo emplea en el experimento ATLAS, para archivar su sistema de monitorización online.
- Facebook inicialmente lo usaba para su sistema de búsquedas en la bandeja de entrada. Desde 2012 emplea Cassandra en Instagram.

- Uber lo usa desde 2016 para almacenar las grandes cantidades de datos generados en tiempo real por su sistema.

Otros ejemplos conocidos y muy significativos, pero de los cuales no se ha encontrado información concreta sobre cómo o para qué lo usan son: Amazon, GitHub, Reddit, Twitter, Hulu, Spotify, ING, Microsoft, NY Times, Walmart o McDonalds.

Se puede apreciar que hay muchos ejemplos de redes sociales o de aplicaciones de streaming entre estos ejemplos. Tal es así, que los escenarios típicos en los que se puede encontrar a Cassandra como bases de datos suelen estar fuertemente relacionados con el Internet de las Cosas, aplicaciones de detección de fraudes, catálogos de productos, mecanismos para realizar recomendaciones a usuarios, listas de reproducción de contenidos multimedia y aplicaciones de mensajería. Además, es importante comentar que el 40 % de los *"Fortune 100"* emplean Cassandra para alguna de sus operaciones [45, 46].

3.2.2. Características

Los usuarios de esta base de datos destacan fundamentalmente la gran disponibilidad que caracteriza a esta plataforma, así como su escalabilidad y rendimiento. Especialmente en términos de escalabilidad está considerada como una de las mejores bases de datos NoSQL, o incluso la mejor. También se suele destacar su capacidad para trabajar con datos de naturaleza crítica gracias a la potencia de la escalabilidad lineal y su tolerancia a fallos, aún empleándose sobre un hardware básico.

Concretamente, las características que definen a Cassandra y la hacen única entre las bases de datos NoSQL son las siguientes:

- Cassandra es esencialmente un híbrido entre un modelo clave-valor y una base de datos tabular, por lo que resulta ser conceptualmente bastante cercana a las tablas de las bases de datos relacionales; pero aún así permite trabajar con datos no estructurados y crear, eliminar y alterar las tablas en tiempo de ejecución sin bloquear las actualizaciones o consultas.

- Como lenguaje de consulta emplea CQL (Cassandra Query Language), que utiliza una sintaxis similar a SQL, aunque con menos funcionalidades. Además existen drivers disponibles para múltiples lenguajes de programación, como Java, Python, Node.JS, Go y C++. CQL no permite *joins* y en lugar de ello, opta por la desnormalización de los datos.

- Posee una arquitectura distribuida basada en *Peer-to-Peer* (P2P). Todos los nodos del clúster tienen el mismo rol y no existen ni puntos únicos de fallo ni cuellos de botella. Al no existir un nodo maestro, cada nodo puede dar servicio a cualquier solicitud.

- Su rendimiento aumenta linealmente a medida que se agregan nuevos nodos. Este concepto llamado escalabilidad lineal, viene a decir que si, por ejemplo, con 2 nodos se pueden realizar 100.000 operaciones por segundo, con 4 nodos serán 200.000, lo cual además de potencia, suma mucha predictibilidad a este sistema.

- La arquitectura distribuida de Cassandra está diseñada para poder desplegarse sobre múltiples centros de datos, con el objetivo de alcanzar una gran capacidad de redundancia y recuperación ante desastres. Posee una gran tolerancia a fallos, ya que los datos se replican automáticamente en múltiples nodos y a través de múltiples centros de datos. Además se puede reemplazar cualquier nodo *en caliente*, sin interrumpir el servicio de la aplicación ni afectar a su rendimiento.

- La consistencia de los datos se puede ajustar tanto para las lecturas como para las escrituras.

- Soporta la integración con otras tecnologías de Apache. Por ejemplo, se puede integrar con Apache Hadoop para soportar Map Reduce. También existe soporte para Apache Pig y Apache Hive.

- Es gratuito y de código abierto, publicado bajo licencia Apache 2.0.

3.2.3. Modelo de datos

Aunque en algunos textos se puede encontrar que esta base de datos es definida como de tipo clave-valor, Cassandra es en realidad una base de datos orientada a columnas, lo que significa que en vez de almacenar tuplas de datos ordenadas de acuerdo a una estructura fija o esquema, almacena "familias de columnas" en *keyspaces*.

El concepto de familia de columnas se asemeja a una tabla en un SGBDR y es más, se las llama tablas desde la versión 3 del lenguaje CQL. Estas tablas contienen filas y columnas. Tal y como se aprecia en la imagen 3.3 cada fila está identificada por una clave única y puede tener múltiples columnas. Cada una de estas columnas está compuesta por 3 campos: el nombre de la columna, su valor y una marca temporal que indica la última vez que el valor fue actualizado.

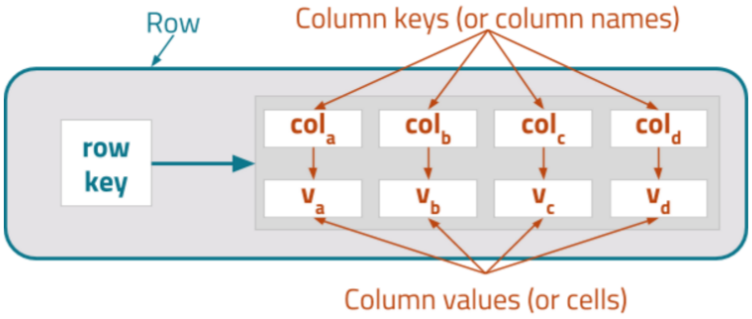

Figura 3.3: Modelo de filas y columnas en Cassandra.

Como la marca temporal no es un dato realmente accesible para el usuario, si no que se emplea para la gestión interna y resolución de conflictos, simplificando mucho se puede decir que este SGBD relaciona los valores de las claves –las filas– con un número variable de tuplas de nombre y valor, que son las columnas. Con este concepto simplificado en mente se puede entender mejor el motivo por el que en ocasiones Cassandra es clasificada como de clave-valor.

A diferencia de una tabla en el modelo relacional, diferentes filas en una misma familia de columnas no tienen por que compartir el mismo conjunto de columnas, y además, una columna puede ser añadida a una o a múltiples filas en cualquier momento.

El espacio de claves o keyspace es el contenedor para una o varias familias de columnas, y ha de ser definido previamente. De la misma manera que una base de datos relacional es una colección de tablas, un keyspace es una colección ordenada de familias de columnas. La estructura completa del modelo de datos en Cassandra puede verse en la imagen 3.4 e introduce dos conceptos aún no mencionados: super columnas y clúster.

Una super columna es un conjunto de columnas que puede referenciarse por un nombre. Se implementa como una estructura que se compone de dos atributos: el nombre y una lista de columnas. En la práctica a una super columna se la considera igual que una columna, sólo que en vez de almacenar un valor almacena columnas. Es un recurso bastante útil, ya que permite tener una colección de valores anidados asociados a otro valor, con la limitación de que una super columna no puede estar dentro de otra super columna.

Por su parte, un clúster es el elemento de más alto nivel que puede referenciarse por un nombre. Es de naturaleza más física que los anteriores y está relacionado con el hardware, ya que agrupa los nodos sobre los que se ejecuta Cassandra. Puede contener uno o más keyspaces.

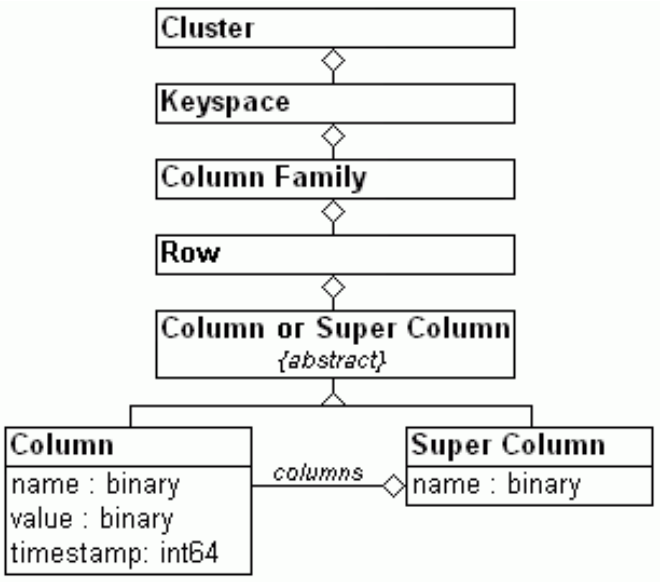

Figura 3.4: Modelo de datos de Cassandra.

3.2.4. Lenguaje de Consultas

La interfaz para comunicarse con Cassandra es CQL3. Este lenguaje de consulta se puede entender como una simplificación de SQL que soporta menos funciones –no soporta *joins o subqueries*– pero que es suficiente para poder explotar el potencial de Cassandra. Las operaciones que se muestran a continuación son las correspondientes a las clásicas operaciones CRUD y la creación de índices [47].

Operaciones CRUD

Lo primero es la creación del keyspace, que es el almacén donde se guardan los datos de una aplicación. La creación se realiza a través de una sentencia CREATE que consta de la siguiente sintaxis: *CREATE KEYSPACE [IF NOT EXISTS] keyspace_name WITH options*. Las sentencias ALTER y DROP para modificar y borrar un keyspace poseen una estructura prácticamente idéntica a la de creación. La opción *if not exists* o *if exits* es recurrente a la hora de crear o borrar cualquier estructura en CQL y evita errores comunes como tratar de crear un keyspace que ya existe o tratar de borrar algo que aún no existe.

Las opciones que se pueden indicar a través de *WITH options* estarán presentes en todos los datos almacenados en el keyspace, y son básicamente 2: La estrategia de replicación, que se comentará en la sección 3.2.5 y una opción para definir si se debe usar el log de registro para las actualizaciones, que por defecto está activado.

Un ejemplo de creación, modificación y uso de un espacio de claves con varias opciones diferentes es el siguiente:

```
CREATE KEYSPACE Excelsior
 WITH replication = {'class': 'SimpleStrategy', 'replication_factor' : 2};

CREATE KEYSPACE Excalibur
    WITH replication = {'class': 'NetworkTopologyStrategy', 'DC1' : 1, 'DC2' : 3}
    AND durable_writes = false;

ALTER KEYSPACE Excelsior
    WITH replication = {'class': 'SimpleStrategy', 'replication_factor' : 3};
```

```
DROP KEYSPACE Excalibur;

USE Excelsior;
```

Como puede apreciarse, tanto la creación como la modificación y el borrado de un keyspace poseen una sintaxis similar, que resulta bastante cercana a SQL. Es importante decir, ya que es la mayor diferencia con SQL, que en CQL es obligatorio especificar el parámetro *class* de la opción *replication* para definir la estrategia de replicación y el parámetro *replication_factor* para definir en cuantos nodos se han de replicar los datos.

Como en la mayoría de bases de datos existentes, es necesario decir qué keyspace se va a usar antes de empezar a crear tablas o añadir datos, esto se hace mediante la sentencia *USE nombre_keyspace*. Para crear una tabla, una vez se ha indicado el espacio de claves que se está usando, se emplea una sintaxis muy similar a SQL que se puede ver en el siguiente ejemplo:

```
CREATE TABLE IF NOT EXISTS monkeySpecies (
    species text PRIMARY KEY,
    common_name text,
    population varint,
    average_size int
) WITH comment='Important biological records'
   AND read_repair_chance = 1.0;

CREATE TABLE prueba (
    pk int,
    t int,
    v text,
    s text static,
    PRIMARY KEY (pk, t)
);

ALTER TABLE prueba ADD x text;
```

Tal y como puede verse, a la hora de crear la tabla se está indicando también una serie de columnas predefinidas en el momento de creación de la tabla, pero también se pueden añadir o quitar posteriormente con la sentencia ALTER.

A la hora de definir las columnas, hay que definir el nombre de la misma y el tipo de dato que almacenará, que restringirá los valores que pueden insertarse en esa columna. Además, pueden declararse los siguientes modificadores sobre las columnas: **clave primaria**, que puede ser una única columna o un conjunto de varias columnas, o **columna estática**, que significa que los valores estáticos serán compartidos por todas las columnas que pertenezcan a la misma fila.

Para insertar datos en la tabla "prueba" creada en el anterior ejemplo y ver que esos valores efectivamente se han añadido, se emplearía la siguiente sintaxis:

```
INSERT INTO prueba (pk, t, v) VALUES (0, 0, 'val0');
INSERT INTO prueba (pk, t, v, s) VALUES (0, 1, 'val1', 'static1');
INSERT INTO prueba (pk, t, v, s,x) VALUES (1, 2, 'val2', 'static2','prueba');
```

```
SELECT * FROM prueba;
   pk | t | v      | s         | x
  ----+---+--------+-------------------
   0  | 0 | 'val0' | 'static1' |
   0  | 1 | 'val1' | 'static1' |
   1  | 2 | 'val2' | 'static2' | 'prueba'
```

Este ejemplo sirve para ilustrar algunos aspectos interesantes del modelo orientado a columnas. A diferencia del modelo relacional, si no se va a insertar un valor en una columna concreta no es necesario insertar un valor *null* en ella u otro valor arbitrario. La cláusula de inserción necesita como mínimo que se inserten datos en las columnas que componen la clave primaria, pero las demás no serían obligatorias.

Además este ejemplo ilustra el concepto de columna estática, ya que aunque en la primera inserción no se ha utilizado la columna "s" y no debería tener valor, como ambas comparten la misma clave de partición (pk) y la columna "s" es estática, comparte el valor de la segunda inserción.

De forma similar a las bases de datos relacionales, la cláusula SELECT de CQL permite especificar las columnas que se desean buscar en una única tabla o emplear funciones como *count(*)* sobre ellas. También permite especificar parámetros concretos de búsqueda a través de la cláusula WHERE, a la que opcionalmente se le pueden añadir cláusulas para agrupar (GROUP BY), ordenar (ORDER BY) o limitar (LIMIT) los resultados. La cláusula where de Cassandra puede usarse también en las operaciones UPDATE y DELETE, pero con la limitación de que sólo puede buscarse por columnas que formen parte de la clave primaria o que tengan un índice secundario.

Cassandra nunca lee antes de escribir. No comprueba si los datos ya existen al hacer un INSERT, por lo tanto las cláusulas UPDATE e INSERT pueden ser usadas para modificar los datos almacenados. UPDATE es la elección más habitual para realizar las modificaciones, y un ejemplo de uso en el ejemplo anterior sería el siguiente:

```
UPDATE prueba SET v = ‘val3‘ WHERE pk = 1;
```

La cláusula DELETE elimina columnas y filas. Si los nombres de las columnas se proporcionan directamente después de la palabra clave DELETE, solo esas columnas se eliminarán de la fila indicada por la cláusula WHERE. De lo contrario, se eliminan las filas enteras.

CQL permite la creación de índices secundarios en tablas, lo que a su vez posibilita que las consultas en la tabla usen esos índices. La sentencia CREATE INDEX se usa para crear un nuevo índice secundario para una columna existente en una tabla dada. El índice puede llevar asociado un nombre o no, según se desee, pero hay que tener en cuenta que para borrar un índice, hay que indicar su nombre.

Si a la hora de crear un índice ya existen datos para la columna elegida, se indexarán de forma asíncrona. Una vez creado el índice, los nuevos datos para la columna se indexan automáticamente en el momento de la inserción. Un ejemplo de creación de un índice secundario sería el siguiente:

```
CREATE INDEX indice ON prueba (v);
```

3.2.5. Naturaleza distribuida

La arquitectura distribuida y descentralizada de Cassandra está basada en una serie de nodos completamente iguales distribuidos en un clúster. Estos nodos se comunican entre ellos a través de un protocolo P2P, denominado *gossip* o "cotilleo" en castellano, por lo que la redundancia es máxima.

Gossip [48] es un protocolo de comunicación de igual a igual en el que los nodos intercambian periódicamente información de estado sobre ellos mismos y sobre otros nodos que conocen, por lo que todos los nodos aprenden rápidamente sobre los demás nodos del clúster. Este protocolo también es útil para la detección de fallos y el enrutamiento de peticiones, ya que a partir del estado y el historial de un nodo se puede saber si éste está inactivo o si se ha recuperado tras una caída, y también se puede usar esta información para evitar enviar las solicitudes de los clientes a nodos inalcanzables o a nodos con bajo rendimiento, proceso que en Cassandra se comoce como "Snitching" [49].

El hecho de que Cassandra esté descentralizada implica que no tiene un punto único de fallo. Todos los nodos en un clúster funcionan del mismo modo y son iguales entre sí, por lo que no existe un nodo administrador o maestro que coordine tareas entre los nodos, como es habitual en otras arquitecturas distribuidas.

Replicación y distribución de datos

En Cassandra la distribución y la replicación de los datos van juntos. Los datos se organizan por tablas y se identifican mediante la clave principal o primaria, que determinará en qué nodo se almacenan los datos. Las réplicas son copias de las filas, que se almacenan en varios nodos para garantizar la fiabilidad y la tolerancia a fallos.

Para la distribución de datos se utilizan técnicas de sharding empleando un *particionador*, que determina cómo se distribuyen los datos entre los nodos del clúster, incluyendo las réplicas.

Un particionador es, en esencia, una función hash que emplea *tokens* para determinar la ubicación de los datos almacenados en una tabla, de forma que cada fila de datos se distribuye a través del clúster en función del valor de su token. Estos tokens representan lo que se conoce como la **clave de partición**, que es la clave primaria de la tabla si ésta es única o el primer componente de la clave primaria si esta clave está compuesta por varias columnas. De esta forma, la base de datos asigna un valor hash a cada token, y cada nodo en el clúster es responsable de un rango de datos en base a los hashes.

Esta política de sharding trata de distribuir de forma uniforme los datos de todas las tablas a lo largo de todos los nodos que conforman el clúster, teniendo en cuenta que los datos que poseen la misma clave de partición estarán siempre juntos, en los mismos nodos. Esto también implica que las solicitudes de lectura y escritura al clúster se distribuyen uniformemente entre los nodos, simplificando el balanceo de la carga.

Por otra parte, tal y como se ha visto anteriormente, la política o estrategia de replicación es un parámetro necesario para crear un keyspace y se aplica a todos los datos contenidos en las tablas de ese keyspace. La estrategia de replicación concreta los nodos donde se colocarán las réplicas de los datos. El número total de réplicas en el clúster se conoce como el factor de replicación. Un factor de replicación de 1 significa que solo hay una copia de cada fila en el clúster, un factor de replicación de 2 significa que hay dos copias de cada fila y que cada copia está en un nodo diferente, y así sucesivamente.

Hay dos estrategias de replicación disponibles:

- *SimpleStrategy.* Es la estrategia más sencilla y se considera adecuada si únicamente hay un centro de datos. Con esta estrategia, la primera réplica se coloca en un nodo determinado por el particionador y las réplicas adicionales, si las hay, se colocarán en los siguientes nodos disponibles.

- *NetworkTopologyStrategy.* Es la estrategia recomendada si se dispone de múltiples centros de datos, y permite especificar cuantas réplicas se desea tener en cada centro de datos.

Al igual que ocurre con los nodos, todas las réplicas son igualmente importantes y no existe réplica primaria o maestra. Como regla general, se considera que el factor de replicación no debe exceder el número de nodos en el clúster.

Niveles de consistencia

Al estar hablando de una red peer-to-peer en la que los datos están replicados en diferentes nodos y que juega con las reglas de **consistencia eventual** presentes en NoSQL, se presenta la situación de que más de un nodo podría responder a la misma operación, cada uno con una respuesta diferente.

Para solventar esto, Cassandra permite ajustar el nivel de consistencia para operaciones de lectura y escritura, ya sea para operaciones concretas o de forma global en un clúster o centro de datos.

Esencialmente, para resolver una operación se elige un coordinador, de forma que el nivel de consistencia indica cuántas de las réplicas deben responder al coordinador para considerar la operación como un éxito. Si las réplicas contactadas tuviesen una versión diferente de los datos, se tomaría por válida la versión más reciente. Estos niveles de consistencia, que se listan a continuación, permiten a Cassandra alcanzar un equilibrio entre la consistencia y la disponibilidad, o priorizar uno de ellos, según sea necesario para las aplicaciones concretas que usen este SGBD.

- ONE, TWO or THREE: estos tres niveles de consistencia indican el número concreto de réplicas que deben responder.
- QUORUM: la mayoría de las réplicas debe responder. Esto significa que si tenemos un clúster con un factor de replicación 3, deben responder 2 de las réplicas.
- ALL: todas las réplicas deben responder.
- LOCAL_QUORUM y EACH_QUORUM: teniendo múltiples centros de datos, especifica si se debe consultar sólo al centro de datos en el que está el coordinador (local), evitando la latencia de comunicación entre centros de datos, o a todos los centros de datos (each).
- LOCAL_ONE: solo una réplica debe responder. En un clúster de centros de datos múltiples, esto garantiza que las solicitudes de lectura no se envíen a las réplicas en otro centro de datos.

Las operaciones de escritura siempre se envían a todas las réplicas, independientemente del nivel de consistencia elegido. El nivel de consistencia simplemente controla cuántas respuestas espera el coordinador antes de responder al cliente [50].

3.2.6. Seguridad

Para comenzar a utilizar Cassandra en un entorno con varios nodos, es necesario realizar configuraciones tras su instalación, lo que en principio debería reducir los problemas de seguridad asociados a una configuración por defecto insuficiente. Obligar al usuario a configurar algunos aspectos antes de empezar a funcionar aumenta la seguridad en comparación con otros SGBD, pero en el caso de Cassandra, también adolece de otros problemas de seguridad comunes, ya que configuraciones de seguridad como la autenticación o la autorización de acceso son parámetros considerados como “adicionales y opcionales”, lo cual puede dejar los datos almacenados completamente expuestos [51].

En cualquier caso, esta base de datos provee tres medidas de seguridad fundamentales [52]:

- Encriptación TLS/SSL para las comunicaciones y transferencias de datos, tanto desde el cliente al clúster, como entre los nodos del clúster.

- Autenticación. Permite establecer unos credenciales de usuario (usuario y contraseña) que se almacenan encriptados en una tabla del sistema. Por defecto, no se emplea ningún tipo de autenticación y

activar esta opción en la configuración tampoco es una solución instantánea, ya que el super usuario por defecto es bien conocido (usuario: cassandra, contraseña: cassandra), por lo que habría que crear uno nuevo y deshabilitar el por defecto.

- Roles y Autorización. Permite establecer diferentes roles de usuarios y diferentes permisos para cada rol (una lista blanca). Por defecto, se otorgan todos los permisos a todos los roles. Una vez que se ha configurado la base de datos para que la autorización esté activada, los permisos deben establecerse con la sentencia GRANT y se pueden eliminar con REVOKE.

3.3. Principales diferencias

Desde el segundo capítulo se ha podido empezar a ver las enormes diferencias que hay entre MongoDB y Cassandra, y son tantas que casi se podría decir que sólo coinciden en que son sistemas de gestión de bases de datos NoSQL de código abierto. Las diferencias entre ellos comienzan en su propia filosofía y concepción, ya no sólo como base de datos, si no en un nivel más general.

MongoDB posee una filosofía Plug-and-Play que resulta muy cómoda y atractiva para sus usuarios, ya que la instalación es muy sencilla en todas las plataformas y no necesita configuración para poder empezar a funcionar, aunque si es recomendable configurar algunos aspectos previamente. Por su parte, habitualmente Cassandra si necesita configuración tras su instalación, y como se ha comentado, la instalación en algunos sistemas operativos no resulta sencilla. También hay que decir que Cassandra presenta un modelo de datos y lenguaje de consulta más cercano a las bases de datos relacionales, mientras que en MongoDB son completamente diferentes.

Existe otra diferencia importante en este aspecto, y es la documentación. El desarrollo y el soporte de MongoDB siempre ha sido ofrecido por la misma empresa que lo creó, y tanto la documentación oficial disponible en la página web de MongoDB como los cursos de MongoDB University resultan bastante completos e intuitivos.

El desarrollo inicial de Cassandra estuvo a cargo de Facebook. Tras ser liberado, el proyecto fue desarrollado y mantenido por Apache, aunque durante un tiempo estuvo muy ligado a otras empresas como DataStax, que también ofrecía documentación de Cassandra (Planet Cassandra), hasta finales de 2016. El resultado de esto, es que actualmente la página web oficial de Cassandra tiene numerosos enlaces a Planet Cassandra, plataforma que está fuera de servicio y que a su vez redirige a la página principal de la web de Cassandra. Por si esto fuese poco, secciones completas de la documentación oficial de Cassandra, como el modelo de datos o cuestiones de arquitectura, están incompletas o simplemente vacías.

En cuestiones de seguridad, hay una diferencia notable, y es que realizando una exploración de ambas bases de datos a través de la plataforma *shodan.io*, para conocer los servidores de bases de datos que son directamente accesibles a través de internet (que no poseen ningún tipo de autenticación o emplean credenciales por defecto débiles como admin/admin), el resultado para Cassandra fue que tiene muchas menos máquinas accesibles (1.272) que MongoDB (62.326).

Esto no quiere decir que haya más de 60.000 bases de datos MongoDB y Cassandra cuyos datos están esperando a ser robados o borrados, simplemente indica que las bases de datos son accesibles. La naturaleza de la información expuesta puede ser muy diversa; es posible que algunos de esos sistemas fácilmente accesibles si estén exponiendo los datos almacenados por sus aplicaciones o usuarios, pero en muchos casos la información accesible es propia del servidor y su gestión, como ficheros logs.

En la siguiente tabla se resumen las principales diferencias entre MongoDB y Cassandra.

	MongoDB	Cassandra
Teorema CAP	CP	AP
Filosofía	Orientado a Documentos	Orientado a Columnas
Programado en	C++	Java
Almacenamiento de datos	Documentos BSON	Familias de columnas
Lenguaje de consulta	Objetos y métodos JavaScript	CQL
Distribuido	Maestro-Esclavo	Peer-to-Peer
Disponibilidad	Alta disponibilidad, basada en la recuperación ante fallos automática en el nodo maestro	Alta disponibilidad, basada en un modelo altamente distribuido y redundante
Tolerancia a fallos	Alta	Excepcional
Última versión	4.2	3.11.4

Tabla 3.1: Principales diferencias entre MongoDB y Cassandra.

Capítulo 4

Preparación del experimento

En este capítulo se entra en un terreno mucho más práctico, con el fin último de abordar el segundo objetivo principal de este trabajo: comparar el rendimiento de los dos sistemas de bases de datos NoSQL estudiados.

Además de realizar el paso evidente de instalar ambas bases de datos en las máquinas donde se harán las pruebas y realizar las configuraciones necesarias, en este capítulo también se especificarán las métricas de rendimiento elegidas y la manera en que se realizarán las mediciones.

Para poder comparar el rendimiento de estas dos bases de datos hace falta un elemento adicional: datos. Y no sólo hace falta tener datos que almacenar en las BD, si no que, para que las pruebas de rendimiento realizadas sean válidas, también se hace necesario que los "modelos o esquemas" para almacenarlos en ambas bases de datos sean lo más equivalentes posibles. Todo esto se tratará en las siguientes secciones.

4.1. Obtención de los datos

La adquisición de los datos que se emplearán en las pruebas es un proceso más delicado de lo que puede parecer a primera vista y requiere tener en cuenta varios factores que se enumeran a continuación:

1. Volumen de datos: la cantidad de datos o su *peso* es un factor importante a la hora de hablar de Big Data, aunque como se ha explicado anteriormente, no existe una cifra que separe lo que se considera Big Data de lo que no.

2. Estructura y formato: Los datos pueden ser completamente estructurados, completamente desestructurados o semi-estructurados. También el tipo de formato en el que estén almacenados puede suponer alguna diferencia a la hora de importar los datos.

3. Número de *filas y columnas*: si bien está relacionado con los dos factores anteriores, proporciona una forma mucho más real y cercana de ver el volumen que alcanzará la base de datos.

4. Relaciones entre los datos: la interconexión de los datos resulta importante, y lo sería aún más si la comparación se hiciese entre un sistema relacional y un sistema nosql, ya que la *integridad referencial* es muy importante en el mundo relacional, y sin embargo, en muchos sistemas nosql no hay ninguna forma real de implementarla.

La correcta elección de los datos que se manejarán en el experimento resulta vital para que este pueda ser considerado como exitoso. Elegir un conjunto de datos con unas características que puedan favorecer a uno de los dos sistemas en estudio podría llevar a unas conclusiones erróneas en cuanto al rendimiento del mismo.

En este caso, el volumen de datos está limitado por la capacidad de almacenamiento de las máquinas de pruebas. Desde luego, lo ideal sería obtener un gran conjunto de datos de cientos de GB, o incluso superar el TB. Pero siendo realistas, el volumen de datos tendrá que limitarse a unos pocos GB.

Ambos sistemas NoSQL están preparados para trabajar con datos no estructurados o semi-estructurados de una manera sencilla y eficaz así que en este caso por simple comodidad y por resultar más sencillo de obtener, se optará por un conjunto de datos estructurados o semi-estructurados con un elevado número de filas y un número más reducido de atributos o columnas. A su vez, se procurará que el formato en el que estén almacenados los datos sea *CSV*, por tratarse de un formato bien conocido y de fácil importación en ambos sistemas de bases de datos.

Los *joins* suponen un elemento muy diferencial, pues Cassandra no aporta ninguna utilidad para su implementación y uso, y sin embargo MongoDB, tras su acercamiento al mundo relacional incorporando ACID, sí permite realizar joins, aunque con ciertas limitaciones. Y si bien en MongoDB se permite referenciar otros documentos y realizar transacciones, Cassandra no soporta ninguna de estas opciones –aunque existe el concepto de *transacción ligera*– por lo cual se optará por un conjunto de datos sencillo, sin relaciones entre ellos.

Para realizar el proceso de obtención de los datos que se utilizarán en las pruebas posteriores, se han barajado varias maneras de conseguirlos que podrían resultar satisfactorias. La primera en ser valorada fue la opción de crear una aplicación (por ejemplo, una aplicación web funcionando en un servidor Apache) que obtuviese datos en tiempo real de alguna plataforma de red social, como Twitter. Esta opción se descartó al introducir más complejidad en las pruebas (diferentes APIs o interfaces para comunicar con las distintas bases de datos) y tiempos de latencia (de red, del servidor e incluso de la plataforma de red social).

Puesto que en el fondo el contenido de los datos es ciertamente irrelevante, se consideró la opción de generar datos aleatorios, en un formato elegido, hasta tener el volumen que se considere adecuado. En este orden, también se ha valorado la opción de obtener un conjunto de datos real desde un *banco de datos* o una fuente similar.

Finalmente se consideró que aunque el contenido de los datos no importase, el hecho de que los datos sean reales si podría añadir cierto valor a las pruebas, que no se tendría al hacerlas con datos arbitrariamente aleatorios, ya que aunque no sea el objetivo, se estaría empleando información real que podría resolver un problema real.

4.1.1. Elección del conjunto de datos

Al poner el foco en un conjunto de datos reales y en el contexto NoSQL, fue bastante natural pensar en datos obtenidos por sensores o aparatos IoT, de forma que se estaría empleando información de una naturaleza similar a la que se emplea habitualmente en casos reales de aplicaciones NoSQL. Por este motivo, se ha optado por recurrir a un banco de datos, concretamente un repositorio de datos relacionados con el Machine Learning de la Universidad de California [53].

Buscando en este repositorio, se encontró un conjunto de datos bastante interesante y que cumplía con los requisitos especificados anteriormente; datos reales pertenecientes a mediciones realizadas con sensores de *Smartphones* y *Smartwatches*, con un gran número de filas o instancias (43.930.257 exactamente) y un número adecuado de columnas o atributos (16). El peso de este conjunto de datos completo es de 3.07 GB, pero está fraccionado en 4 archivos de formato *.csv*. Además, se indica que existen valores faltantes o "perdidos", lo cual también se adapta bien a la estructura deseada.

Puesto que ese *data set* se ajusta bien a lo que se necesita para realizar las pruebas, se ha optado por emplearlo como el conjunto de datos que poblará las bases de datos.

4.1.2. Descripción del conjunto de datos elegido

El conjunto de datos elegido recibe el nombre de "Heterogeneity Activity Recognition Data Set" [54] y es un conjunto de datos diseñado para comparar algoritmos de reconocimiento de la actividad humana (clasificación, segmentación automática de datos, fusión de sensores, extracción de características, etc.) en un contexto perteneciente al mundo real.

Este data set contiene las lecturas de dos sensores de movimiento que se encuentran integrados comúnmente en los dispositivos inteligentes: acelerómetro y giroscopio. Las lecturas se registraron mientras los usuarios –llevando un smartphone o un smartwatch– realizaban diferentes actividades; andar, ir en bici y subir o bajar escaleras, así como estar sentado o de pie. Estos datos se han recogido en 4 archivos .csv según el tipo de dispositivo y el tipo de sensor, y tienen los siguientes nombres: Phones_accelerometer.csv, Phones_gyroscope.csv, Watch_accelerometer.csv, Watch_gyroscope.csv.

Los dispositivos que se han utilizado para recoger las muestras fueron 4 smartwatches (2 LG Watches y 2 Samsumg Galaxy Gears) y 8 smartphones (2 Samsung Galaxy S3 mini, 2 Samsumg Galaxy S3, 2 LG Nexus 4 y 2 Samsung Galaxy S+). Estos dispositivos fueron empleados por 9 usuarios diferentes, cada uno de ellos con un identificador distintivo asignado para poder diferenciarlos.

La información de este conjunto de datos está estructurada de la siguiente manera:

- Index: Un índice indicando el número de fila.
- Arrival_Time: Indica la hora a la que la medición llegó a la aplicación de monitorización.
- Creation_Time: Marca temporal que el Sistema Operativo asigna a la medición.
- X,y,z: Indica los valores proporcionados por los sensores en los 3 ejes; X, Y, Z.
- User: Nombre del usuario que generó la muestra, identificados por una letra de la "a a la "i.
- Model: Indica el modelo de teléfono o reloj que originó la muestra.
- Device: Especifica el dispositivo en concreto que originó la muestra. Se les nombra añadiendo al nombre del modelo, el número identificativo de dispositivo, por ejemplo nexus4_1 y nexus4_2.
- Gt: Señala la actividad que se estaba realizando; bike, sit, stand, walk, stairsup, stairsdown o null.

Adicionalmente, este conjunto de datos anexa otro conjunto de datos muy diferente con 6 atributos relativos a la posición del acelerómetro, sumando de esta forma 16 atributos totales con los 10 que se han explicado en la lista anterior. Para el caso de este experimento, se emplearán los archivos en formato csv comentados anteriormente, con los 10 atributos listados.

4.2. Obtención de los modelos de bases de datos

En el mundo de las bases de datos relacionales, las metodologías y buenas prácticas a la hora de obtener modelos han sido ampliamente estudiadas, probadas y refinadas tras décadas de investigación y uso. En la figura 4.1 se muestra un flujo conceptualmente simplificado del proceso habitual a la hora de diseñar una base de datos relacional.

Tal y como puede verse, tras analizar los requisitos y requerimientos de los datos, se realiza el modelado conceptual, normalmente en un diagrama de entidad-relación (DER) [55] para posteriormente, normalizar el modelo relacional obtenido a partir del modelo conceptual. En la figura 4.2 se puede ver un ejemplo prototípico de un Diagrama E-R.

4.2.1. Modelado de datos NoSQL

El proceso de modelado de los datos en el mundo NoSQL se afronta con un punto de vista muy diferente al modelado relacional. No existe un estándar ni unas normas bien definidas para la generación de los

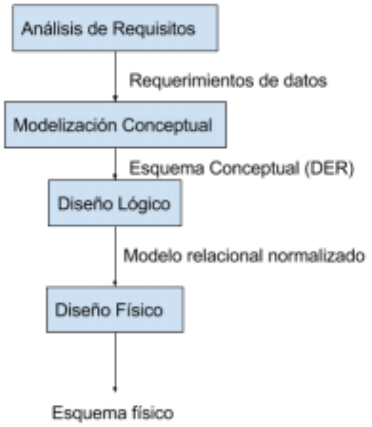

Figura 4.1: Diseño Bases de Datos Relacionales.

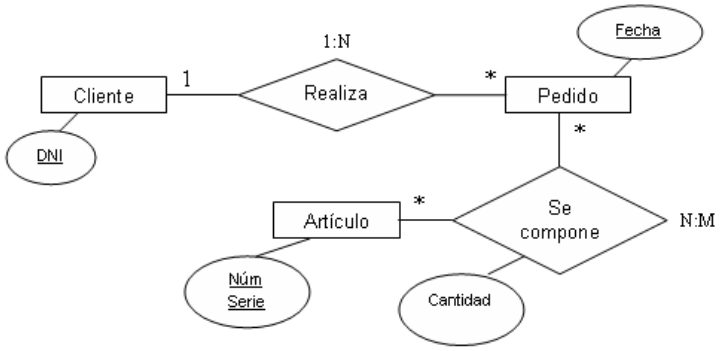

Figura 4.2: Ejemplo de diagrama Entidad Relación.

modelos de datos NoSQL, en parte, por que existen demasiados tipos de bases de datos NoSQL, y cada una trata los datos de maneras diferentes. Por otra parte, las técnicas aplicadas al diseño relacional no son aplicables (no eficazmente) en bases de datos no relacionales. Los diagramas de entidad-relación, pese a formar parte del proceso de Modelado Conceptual (y por tanto, ser técnicamente aplicables también al mundo no relacional), están muy ligados al Modelo Relacional, tanto que existe una cierta tendencia en el mundo NoSQL de evitar este tipo de diagramas a menos que sea estrictamente necesario (estar manejando datos que poseen relaciones importantes entre ellos y usar una base de datos no relacional que permita implementar dichas relaciones).

Las bases de datos NoSQL están diseñadas pensando en maximizar la eficiencia y en afrontar los problemas que el enfoque relacional no puede gestionar de manera eficaz, de esta forma, el modelado de datos NoSQL se pliega a los requisitos y estructuras de datos cambiantes, apostando por la flexibilidad y **enfocándose en qué datos se buscarán** y qué características tendrá dicha búsqueda. Así pues, al modelar datos orientados al mundo NoSQL, se tiende a realizar un proceso de *desnormalización* de los datos.

Las tendencias actuales a la hora de realizar un modelo de datos NoSQL según las relaciones de los mismos son las siguientes:

- **Embedding**: Consiste en almacenar datos relacionados (generalmente, con una relación *1 a 1*) en una única estructura (documento, tabla, keyspace...). De esta forma, a la hora de realizar búsquedas se evita realizar operaciones de JOIN y se maximiza el rendimiento.

- **Establecer relaciones**: Consiste en generar referencias y relaciones entre elementos existentes en la base de datos (normalmente cuando las relaciones son de tipo *1 a **). A diferencia de en los sistemas gestores de bases de datos relacionales, este tipo de modelado de datos no es aplicable a todas las

bases de datos NoSQL, y en las que lo es, queda naturalmente restringido por las características propias de cada sistema gestor de bases de datos. Con esta tendencia es habitual emplear diagramas DER para el modelado conceptual de los datos.

- **Gestionar las relaciones en la capa de aplicación**: Por último, es habitual en el mundo NoSQL almacenar los datos *en bruto*, sin importar en absoluto cuestiones como que sean no estructurados o semi-estructurados, que no estén normalizados, que haya cierta redundancia en ellos o que estén relacionados entre sí. De esta forma el sistema gestor de bases de datos nosql únicamente se encargaría de realizar las lecturas y escrituras de los datos, dejando las cuestiones relativas a su tratamiento a la capa de aplicación del servicio que los emplea.

En base a estas tres tendencias, se puede asumir que el enfoque de embeber datos en una única estructura sería una decisión acertada cuando las relaciones entre ellos sean simples y que el de establecer relaciones y referencias entre los datos lo será cuando sea necesario un enfoque más relacional de los mismos. Por su parte, el almacenamiento en bruto de los datos ignora casi por completo cuestiones relativas al modelado conceptual o lógico y se centra en la eficiencia máxima de la escritura y lectura de los datos. Esta tendencia a la hora de tratar la información es muy adecuada cuando hablamos en términos de Big Data e IoT, ya que son dos situaciones en las que, tanto la afluencia como el volumen de los datos que se acaban manejando resultan enormes.

En definitiva, se puede decir que el mundo NoSQL genera nuevos desafíos a la hora de modelar los datos de manera flexible y siempre ha de estar orientado a la eficacia de las consultas. Es por eso que la representación del modelo de datos no relacional es en ocasiones problemático, pudiéndose encontrar representaciones variopintas de los mismos; desde modelos de entidad-relación clásicos, representaciones en JSON, descripciones más o menos formales representadas en tablas o esquemas, etc...

Todo lo explicado en este apartado sobre la problemática del modelado de la información para bases de datos NoSQL lleva habitualmente a la conclusión lógica de que en realidad, el enfoque NoSQL no emplea ningún modelo de datos. Esta afirmación es en algunos casos un tanto arriesgada y en otros, errada. Lo que sí es cierto, es que sea cual sea el enfoque del SGBD NoSQL que se emplee, este empleará un esquema o modelo flexible que normalmente se irá adaptando y refinando con el tiempo.

4.2.2. Modelado de los datos elegidos

Siguiendo con lo explicado en el punto anterior, en este punto se pasará a comentar cómo se modelará concretamente el dataset elegido, así como las opciones que se han barajado y el razonamiento para aceptarlas o rechazarlas.
El conjunto de datos que se empleará está repartido entre 4 archivos en formato CSV que no tienen unas relaciones fuertes entre ellos. Naturalmente, desde un punto de vista relacional se podría realizar un proceso de modelado conceptual y normalización que probablemente generaría otras tablas adicionales, como “Usuarios”, “Modelo del dispositivo” o “Actividad” y que podrían ser empleadas para mantener la integridad referencial (claves foráneas) de las tablas ya existentes. Sin embargo, como se ha podido ver a lo largo del presente documento, emplear una base de datos no relacional utilizando un enfoque puramente relacional, si bien en ocasiones concretas podría resultar ser viable, no es recomendable.

Debido a esto, a continuación se enumeran las opciones que se han valorado a la hora de implementar las tablas de las bases de datos para MongoDB y Cassandra. Entiéndase que en este punto se está empleando el término “tabla” de manera genérica para referirse a las estructuras propias de cada SGBD NoSQL, y no en términos relacionales.

1. **Desnormalización total en una única tabla:** Esta opción sería completamente contraria al modelado relacional clásico, uniendo la información contenida en los 4 archivos .csv en una única

tabla. Para realizar este proceso de una forma clara, sería necesario incluir dos atributos más para indicar por cada *fila* si la información procede de un teléfono o un reloj y si fue generada por el acelerómetro o por el giroscopio. Con este enfoque se implementarían estructuras de índices en cada gestor de bases de datos que garanticen que la búsqueda por "tipo de dispositivo" o "sensor" sean posibles y eficientes.

2. **Embeber datos relacionados en dos tablas:** En términos conceptuales, existe una relación directa entre los datos que permitiría reducir el número de tablas a la mitad de una manera bastante natural. Los archivos están separados por el *tipo de dispositivo* y el *tipo de sensor* que lo realizó, de forma que podrían unirse en dos tablas, por ejemplo, "Acelerómetro" y "Giroscopio", añadiendo un atributo por cada tabla para indicar el tipo del dispositivo que realizó la muestra, o al revés, creando una tabla para "Teléfono" y otra para "Reloj" y añadiendo un atributo común para indicar el sensor.

3. **Importar los datos sin modificar:** De esta manera, los datos en formato .csv se importarían a las bases de datos sin realizar ninguna modificación, generándose 4 tablas diferentes con el contenido exacto de los archivos. Esta sería la opción más sencilla, pero también la más orientada al mundo relacional.

Con las opciones 1 y 2, se podría prescindir del atributo *index* que llevan todas las tablas, al ser un dato que carecería de utilidad una vez que varios archivos .csv hayan sido volcados en otro contenedor que los incluya. Aunque por simplicidad y por no restarle más "peso" al archivo resultante, se optará por no eliminarlo y mantenerlo, aunque sea información que no aporte nada relevante.

Por otra parte, realizar una implementación de los modelos correspondientes a ambas opciones, permitiría realizar una comparativa de rendimiento con diferentes volúmenes de datos, teniendo la opción 1 un volumen mucho mayor que el de que tendría la opción 2 si se escoge, por ejemplo, la tabla "Reloj" anteriormente comentada, para realizar las consultas.

Explorar las posibilidades y el rendimiento del enfoque más relacional en bases de datos NoSQL –la opción 3– sería interesante si ambos gestores de bases de datos pudiesen emplear JOINS, para comprobar el rendimiento al realizar ese tipo de operaciones. Como este tipo de comparativa de rendimiento entre MongoDB y Cassandra no sería posible y el volumen de cada tabla no aportaría una gran diferencia con respecto a la segunda opción, se ha optado por descartar la tercera opción.

A continuación se muestra el modelo conceptual, descrito a través de un ejemplo de los datos reales en notación JSON que se emplearán para la comparativa de rendimiento objeto de este capitulo.

Para la primera opción, en la que toda la información estaría almacenada en una única estructura, sería el modelo siguiente:


```
{
    "Opcion1": {
        "_id": 1,
        "type_device": "watch",
        "sensor": "accelerometer",
        "index": 0,
        "arrival_time": 1424696638740,
        "creation_time": 27920678471000,
        "x": -0.5650316,
        "y": -9.572019,
        "z": -0.61411273,
```

```
          "user": "a",
          "model": "gear",
          "device": "gear_1",
          "gt": "stand"
     }
}
```


Mientras que, para la segunda opción se separa la información en dos estructuras diferenciadas, siguiendo el siguiente esquema:


```
{
    "Opcion2": {
        "Phone":{
             "_id": 1,
             "sensor": "accelerometer",
             "index": 0,
             "arrival_time": 1424696638740,
             "creation_time": 27920678471000,
             "x": -0.5650316,
             "y": -9.572019,
             "z": -0.61411273,
             "user": "a",
             "model": "nexus4",
             "device": "nexus4_1",
             "gt": "stand"
             },
        "Watch":{
             "_id": 1,
             "sensor": "accelerometer",
             "index": 0,
             "arrival_time": 1474694538787,
             "creation_time": 27820678471500,
             "x": -0.5750311,
             "y": -8.672013,
             "z": -0.71411275,
             "user": "a",
             "model": "gear",
             "device": "gear_1",
             "gt": "stand"
             }
     }
}
```


4.3. Adecuación de los ficheros

Tal y como se ha comentado en la sección anterior, se van a realizar las pruebas de rendimiento siguiendo dos modelos diferentes. Ambos modelos requieren realizar una modificación sobre los datos originales antes de cargar las respectivas bases de datos con ellos. Para realizar el proceso de modificación

y adecuación de los datos, se ha empleado el comando de consola de linux *sed* empleando tuberías y redirecciones. Para mayor simplicidad y claridad, los comandos empleados en este punto, así como su función y observaciones sobre su comportamiento, están incluidos en el Anexo I

4.4. Implementación de las bases de datos

El primer paso para implementar ambas bases de datos, es realizar el proceso de instalación completo de los sistemas gestores de bases de datos en las máquinas donde se realizarán las pruebas. A la hora de realizar la instalación, se emplearán las **versiones estables** más recientes, descargadas desde la página oficial de cada producto. Para MongoDB la versión estable más reciente es la 4.2, mientras que para Apache Cassandra es la versión 3.11.4. Ambos procesos de instalación están bien documentados en las secciones correspondientes de la documentación oficial [56, 57], y el proceso de instalación se ha seguido al pie de la letra, por lo que se considera que no es necesario ni relevante comentar nada más sobre el mismo, ni incluir documentación adicional o imágenes al respecto.

En relación a la configuración base, una de las características más llamativas y que ya se ha comentado de MongoDB es su naturaleza *Plug And Play*, que implica que el servicio puede comenzar a funcionar inmediatamente después de instalarlo, sin realizar ningún tipo de configuración previa. Con Cassandra el escenario si es un poco diferente, pues si requiere un mínimo de configuración, aunque esta configuración está orientada a su despliegue en varios nodos, y para su ejecución **en un único nodo**, la configuración por defecto que se encuentra en el archivo *./conf/cassandra.yaml* resulta suficiente y no se recomienda cambiarla.

Naturalmente, se podrían realizar configuraciones relativas a incrementar la seguridad, cuestiones como el control de acceso y la protección de los datos mediante contraseña. Y de hecho, en un caso de uso real es un paso que se debería realizar obligatoriamente para evitar alguno de los problemas que ya se ha comentado anteriormente, referentes a la falta de seguridad por defecto en el mundo nosql [39]. No obstante, para el caso que nos ocupa no se hace necesario realizar ninguna configuración adicional y se optará por realizar el experimento con la configuración por defecto en ambas bases de datos.

4.4.1. Creación de las bases de datos

En lo referente a MongoDB, en este punto no se puede comentar demasiado, pues es una base de datos cuyo esquema se crea según se puebla la base de datos, siendo tan flexible como para permitir que se inserten documentos siguiendo otra estructura diferente a los ya existentes. Por tanto, el proceso de creación de la base de datos en MongoDB se realizará en tiempo de ejecución cuando se carguen los datos.

Por su parte, Cassandra si necesita definir el esquema que empleará y también necesita crear la base de datos previamente al volcado de los mismos. El esquema que seguirá Cassandra se puede ver claramente con el script de creación del mismo, que aparece en el Anexo II. En relación a este esquema, se ha de comentar la elección de la clave primaria, que es un aspecto fundamental en Cassandra y del que depende en gran medida su utilidad y rendimiento.

En el apartado 3.2.4 se comentó que para Cassandra, insertar los datos puede funcionar también como la sentencia UPDATE. Este comportamiento cobra importancia en este momento, pues si por ejemplo, tomamos la tabla *watch* del segundo modelo u opción y establecemos que su clave primaria será el campo *sensor*, después de importar y procesar todos los datos del correspondiente archivo, al realizar la sentencia *select * from watch;* nos encontraríamos únicamente dos resultados, uno con el valor de sensor *accelerometer* y otro con el valor *gyroscope* en lugar de los, aproximadamente, 6 millones que se habrían importado y procesado. Es por este motivo que se ha elegido una clave primaria compuesta, con un número de campos suficientes como para que se inserten los datos sin sobre-escribirse entre ellos. Esta

clave primaria sería, respectivamente para el primer modelo y el segundo:

```
PRIMARY KEY ((type_device,sensor,indice),user,gt)
PRIMARY KEY ((sensor,indice),user,gt)
```

Tras esto, la situación entre MongoDB y Cassandra queda desequilibrada. La clave primaria de Cassandra sirve a su vez de índice, mientras que MongoDB en estos momentos sólo tendría una clave primaria, que se genera automáticamente junto al campo _id para identificar cada documento, y que no serviría para la realización de las pruebas. Por esto, para volver a igualar la situación entre las bases de datos, se creará un índice secundario para MongoDB que incluya los mismos campos que la clave primaria de Cassandra. Las siguientes sentencias se ejecutarán antes de poblar las bases de datos de ambos modelos en MongoDB, siendo la primera sentencia para la opción 1 y la segunda para la opción 2.

```
.createIndex({"Type_device":1,"Sensor":1,"Index":1,"User":1,"gt":1})
.createIndex({"Sensor":1,"Index":1,"User":1,"gt":1})
```

Puesto que el esquema de Cassandra ha de crearse antes, y como para que las pruebas entre los dos programas estén equilibradas se necesita que se creen los índices de MongoDB antes de las inserciones de datos, se ha creado el script *inicializacion.sh*, que puede verse en el Anexo I.

4.5. Máquinas de prueba y métricas de rendimiento

Las máquinas sobre las que se realizarán las pruebas de rendimiento serán 3 y se corresponderán con 2 ordenadores portátiles personales y una máquina virtual cedida por la Universidad de Valladolid. La información más importante y que sirve para diferenciar las características de cada máquina está recogida en la tabla 4.1.

Nombre	Modelo	Procesador	Memoria RAM	Almacenamiento	Sistema Operativo
Portátil I5	Acer Aspire E1-572G	Intel Core i5 4200U (1.60 GHz)	6GB DDR3	Samsumg SSD 850 EVO 500GB Partición 50GB	Ubuntu 18.10
Portátil I3	ASUS TP300L	Intel Core i3 4030U (1.9GHz)	4GB DDR3	HDD 500GB Partición 50GB	Ubuntu 18.04.03 LTS
Máquina Virtual			8GB	50 GB	Ubuntu 18.04.02 LTS

Tabla 4.1: Máquinas sobre las que se realizarán las pruebas

Nótese que la información relativa a la máquina virtual no es completa, ya que no se conocen los detalles del hardware real sobre el que está ejecutándose. En cualquier caso, para tratar de completar esta información, se ha incluido en el anexo II la información que proporciona la máquina virtual al ejecutar diversos comandos de consulta sobre el hardware del sistema.

Por otra parte, se ha elegido como la principal métrica de rendimiento el **tiempo de ejecución** de las diferentes tareas (se concretarán en el siguiente capítulo) que se realizarán sobre cada base de datos y sobre cada máquina. En base a este tiempo obtenido, se podrá establecer una comparativa fiable del rendimiento entre cada gestor de bases de datos, tanto de forma individual para cada máquina –en función de su propio hardware– o considerando las 3 configuraciones hardware en conjunto y extrayendo conclusiones más generalistas, si es posible.

También se ha valorado recoger datos potencialmente útiles o relevantes para el estudio del rendimiento, tales como el uso de memoria o de disco durante las pruebas de rendimiento. No obstante, y puesto que el tiempo de ejecución o sus métricas derivadas, como operaciones por segundo, son las métrica

por excelencia al hablar de rendimiento en los documentos de referencia consultados, y son las que más relevancia aportan, se ha optado por centrar este estudio del rendimiento en los tiempos de ejecución en exclusiva.

Capítulo 5

Pruebas realizadas

En el capítulo anterior, se dejó el experimento preparado para su realización; tras la obtención del conjunto de datos, la creación de dos modelos de prueba para cada una de las bases de datos, la definición de las métricas de rendimiento que se emplearán y las máquinas que realizarán las pruebas, así como la instalación e implementación de los sistemas gestores de bases de datos MongoDB y Cassandra en cada una de las máquinas.

En el presente capítulo, se definirá con exactitud las pruebas que se realizarán sobre las bases de datos, la forma en que se ejecutarán, y la manera en que se recogerán los resultados, que serán presentados en el próximo capítulo para su análisis.

5.1. Operaciones sobre las bases de datos

Para la comparativa de rendimiento, se hace necesario que las operaciones que se realicen sean compatibles entre sí y estén disponibles en ambas bases de datos. Es por esto que se ha descartado la realización de operaciones más complejas, como podrían ser los JOINS u operaciones de agregación, en favor de las operaciones más básicas de las que dispone una base de datos: las operaciones CRUD. Por sus siglas en inglés CRUD se refiere a los procesos *Create, Read, Update y Delete*, a las que en adelante se hará referencia como inserciones, consultas, actualizaciones y borrado.

5.1.1. Inserciones

Tal y como pudo verse en el capítulo 3, para realizar inserciones de datos MongoDB emplea una estructura *insert()* basada en JavaScript y JSON, mientras que Cassandra emplea un lenguaje propio, CQL, bastante cercano a SQL a la hora de realizar las operaciones CRUD.

Estas sentencias *insert* son habituales a la hora de insertar datos de forma manual en ambas bases de datos, desde su propio *shell*, pero para este caso se ha optado por cargar los datos existentes en los diferentes ficheros .csv que se prepararon para tal fin en el capitulo anterior. Para cargar datos masivos que provienen de un fichero externo y que están en un formato concreto, ambas bases de datos ponen a disposición de sus usuarios varias utilidades. En el caso de MongoDB, tenemos el comando *mongoimport*, una herramienta para importar contenido de ficheros externos en formato JSON, CSV o TSV. Mientras que para Cassandra, se dispone del comando *COPY*. A continuación se muestra un ejemplo de uso de ambos comandos.

```
mongoimport --db opcion2 --collection watch --type csv --headerline
--file /home/opcion2_watch.csv
```

```
cqlsh -k opcion2 -e "COPY watch (sensor, indice, arrival_time, creation_time, x, y, z,
```

```
user, model, device, gt) FROM '/home/opcion2_watch.csv' WITH HEADER = TRUE;"
```

Al tener dos modelos bien diferenciados, con una gran diferencia de filas a insertar, se podrá establecer una comparativa entre ambas bases de datos teniendo en cuenta la cantidad de datos a importar. Es por esto que en este punto, se ha optado por realizar la carga de dos de los archivos, en lugar de los 3 preparados anteriormente. En las consultas sobre el modelo 2 no se realizarán JOINS entre las dos estructuras, por lo tanto cargar ambos archivos no ofrece nada demasiado relevante ni diferencial, pues las consultas se deberán hacer sobre las *tablas* separadas. Debido a esto, se importarán el archivo .csv correspondiente al primer modelo y el archivo .csv correspondiente a la los datos de los Smart Watchs en el modelo 2, por ser el que tiene un número menor de filas, y por tanto, presentar una mayor diferencia de tamaño con respecto al archivo que contiene todos los datos.

Es importante señalar en este punto que, la inserción masiva de datos se realizará de una forma más lenta al contar ambas bases de datos con estructuras de índices creadas previamente, pues se ordenarán las filas/documentos según se vayan importando. Si bien en cuanto a los tiempos de inserción no estaremos en un escenario del mejor caso, si estamos en un escenario más realista y equilibrado entre ambos sistemas de bases de datos. También cabe destacar que estas estructuras de índices serán útiles para la ejecución óptima del resto de consultas, y que como se ha comentado en el capítulo anterior, en el caso de Cassandra su uso se hace necesario para no perder contenido.

5.1.2. Consultas

Para la realización de consultas o búsquedas a la base de datos, MongoDB emplea la sentencia propia de JavaScript *find()* mientras que Cassandra emplea SELECT, igual que en SQL.

Con el objetivo de establecer una comparativa fiable entre ambas bases de datos, se ha optado por realizar una consulta relativamente sencilla, con varios operadores de búsqueda (igualdad, mayor que, menor que...), que afecten tanto a campos indexados como a no indexados y que además arrojen el mismo resultado tanto para el modelo 1, en el que todos los datos se han cargado desde un único archivo en una única estructura, como para el modelo 2 en el que los archivos estaban separados, y por tanto, los datos almacenados también.

Con esto en mente, se han seleccionado los campos *Sensor, User, Model, x, z y gt* como el objeto de las consultas y se han establecido los valores por los que se filtrarán:

- sensor = accelerometer
- user = a
- Model = lgwatch
- $x > -6{,}9$
- $z < 2{,}3$
- gt = sit

De esta forma, las consultas resultantes quedarían de la siguiente forma, respectivamente para MongoDB y para Cassandra:

```
db.coleccion.find({"Sensor":"accelerometer","gt":"sit","User":"a","Model":"lgwatch",
"x": {$gt: -6.9},"z":{$lt:2.3}}).forEach(printjson)
```

```
SELECT * FROM tabla WHERE sensor='accelerometer'and gt='sit' and user='a' and
model='lgwatch' and x >-6.9 and z<2.3 ALLOW FILTERING;
```

Las consultas sobre las bases de datos con estos filtros, ofrecen 81 documentos/filas como resultado para ambas bases de datos. Si bien, se hace necesario explicar las ordenes añadidas a cada una de los comandos de búsqueda.

En MongoDB, los datos por defecto se cargan en un cursor que permite iterarlos de 20 en 20, ya sea de forma manual en la shell de mongo o con un script que lo haga automático. Para evitar este comportamiento por defecto, se añade el método *forEach()*, indicándole que muestre todos los datos en formato JSON.

Por su parte, el comportamiento por defecto de Cassandra no permite realizar consultas que escapen del ámbito de la clave primaria o de los índices secundarios, sólo permitiendo dichas consultas si se incluye la cláusula *ALLOW FILTERING*. Por otra parte, Cassandra muestra los resultados por lotes, de forma análoga a MongoDB, con un *batch* de 100 filas iterables. La consulta elegida proporciona 81 resultados, y por tanto no se hace necesario evitar este comportamiento, pero de requerirlo, se podría abordar con la sentencia *PAGING OFF*.

5.1.3. Actualizaciones

La idea original al plantear las operaciones que se ejecutarían para las pruebas de rendimiento radicaba en pares simétricos: Que las operaciones de inserción y borrado, por un lado, y las de búsqueda y actualización, por otro, involucrasen a las mismas filas. De esta manera, el borrado se realizaría sobre toda la base de datos, y se realizaría un UPDATE sobre los 81 resultados obtenidos por la consulta anterior.

En MongoDB, esta idea se reflejaría con la siguiente consulta, donde se actualiza el campo *model*:

```
.updateMany({"Sensor":"accelerometer","gt":"sit","User":"a","Model":"lgwatch",
"x": {$gt: -6.9},"z":{$lt:2.3}}, {$set: {"model":"newwatch"}})
```

No obstante, las sentencias UPDATE en Cassandra tienen las siguientes limitaciones:

1. Tiene que actualizarse un campo que no forme parte de la clave primaria.
2. En la cláusula WHERE se tienen que incluir todos los campos que compongan la clave primaria.
3. Las comparaciones han de ser de igualdad (= ó *IN*).
4. No se pueden incluir comparaciones de campos que no pertenezcan a la clave primaria.

Al darse las limitaciones que se acaban de enumerar de manera conjunta, la idea original se ve truncada, pues estas limitaciones implican que las actualizaciones de datos están orientadas a realizarse sobre filas individuales y no sobre grandes conjuntos de filas. Por esto, se ha optado por adaptar la sentencia de actualización de datos para ambos sistemas gestores de bases de datos, de forma que actualicen el valor del campo de una única fila, tal y como puede verse a continuación:

```
UPDATE watch SET model ='newwatch' where sensor='accelerometer' and gt='sit' and
user='a' and indice =122214;
```

```
db.watch.updateOne({"Sensor":"accelerometer","gt":"sit","User":"a","Index":122214},
{$set: {"model":"newwatch"}})
```

Si bien, estas sentencias distan bastante del concepto original mostrado anteriormente, también resulta interesante ver como manejan las dos bases de datos este tipo de actualizaciones tan concretas, especialmente en el primer modelo, que maneja una gran cantidad de filas/documentos.

5.1.4. Borrado

A diferencia de la sentencia de actualización de Cassandra, que resultó ser muy poco flexible e intuitiva, la sentencia de borrado si permite un mayor grado de flexibilidad. *DELETE* permite borrar tanto filas como columnas, siendo necesario indicar condiciones en la cláusula *where*, a diferencia del lenguaje SQL. Estas condiciones de la sentencia where no están sujetas a las mismas restricciones que el UPDATE, pudiéndose emplear campos no pertenecientes a la clave primaria y comparaciones de mayor o menor, por ejemplo.

Cassandra también permite el borrado masivo de los datos contenidos en una tabla, al eliminar dicha tabla (*DROP TABLE*) o truncarla (*TRUNCATE TABLE*), siendo la única diferencia entre estas sentencias que drop table también elimina la estructura de la tabla y truncate table, únicamente el contenido.

Como la intención desde el principio fue borrar todo el contenido de la tabla, se optará por emplear la sentencia de truncado en Cassandra y una sentencia equivalente en MongoDB. Un ejemplo de ambas puede verse en las siguientes líneas:

```
TRUNCATE watch;
db.watch.deleteMany({})
```

En esta sección se han fijado las sentencias que se ejecutarán en ambas bases de datos, realizando un esfuerzo para que sean equivalentes y equitativas para las dos, de forma que las pruebas de rendimiento sean válidas y útiles. No obstante, esto también ha servido para hacer una comparativa a nivel conceptual y funcional sobre el uso y la flexibilidad de estas bases de datos, quedando patente que MongoDB resulta mucho más flexible y accesible que Cassandra.

5.2. Mediciones y Ejecución de comandos

El comando *time* es un comando del sistema operativo Unix y derivados que se utiliza para determinar la duración de ejecución de un determinado comando. Este comando informa de cuanto tiempo llevó ejecutar la orden indicada en términos de tiempo de CPU de usuario, tiempo de CPU del sistema y tiempo real, arrojando una salida similar a la siguiente:

```
# time ls -lh
total 8,0K
drwxr-xr-x 3 root root 4,0K abr 10 16:16 Desktop
drwxr-xr-x 5 root root 4,0K abr 10 22:09 ejemplos

real    0m0.020s
user    0m0.000s
sys     0m0.004s
```

Como este comando proporciona diferentes tiempos, se hace necesario explicar qué es cada tiempo y cuál se tomará como el valor más útil para la medición del rendimiento. A modo de ejemplo, cuando un programa recorre una matriz, se acumula tiempo de CPU de usuario. Por el contrario, cuando un programa ejecuta una llamada al sistema, como un exec o un fork, se está acumulando tiempo de CPU del sistema. Por su parte, el término tiempo real en este contexto se refiere al tiempo transcurrido, como si fuese un cronómetro. El tiempo de CPU total (tiempo de usuario + tiempo de sistema) puede ser mayor o menor que ese valor de tiempo real, según el comportamiento del sistema en el momento que se ejecuta el comando. [58]

Conociendo esto, se optará por tomar el valor de **tiempo real** que proporciona el comando time.

Una vez definida la manera en la que se tomará los tiempos, únicamente queda adaptar las sentencias fijadas en la sección anterior para poder lanzarlas desde la consola de linux, junto al comando time. Como se ha podido observar, la mayoría de estas sentencias se realizan desde el *bash* propio de cada gestor, accediendo a él con los comandos *mongo* y *cqlsh*, siendo la excepción el comando mongoimport, que se ejecuta desde la consola de linux.

Para lanzar consultas desde la consola de linux, en MongoDB encontramos la opción - *-eval 'sentencia javascript'* del comando mongo, teniendo que indicar la base de datos sobre la que se realizará la consulta. Mientras que en Cassandra, se dispone de la opción *-e "sentencia CQL"* del comando cqlsh, habiendo que indicarle con el comando -k el keyspace que se usará. Así pues, las sentencias completas que se han usado para la realización de las pruebas se pueden ver en el Anexo III.

Finalmente, la ejecución de estos comandos se ha automatizado mediante el uso de scripts bash sencillos, que recogen la salida de los comandos (incluyendo el tiempo proporcionado por "time") en archivos separados, para poder validar su correctitud.

Estos scripts, que pueden verse en los anexos III y IV realizan el flujo completo de pruebas, comenzando por la inserción masiva de datos, continuando con la búsqueda y la actualización de los mismos, y finalizan borrando todos los datos de las bases de datos. Debido a la forma en la que han sido diseñadas las pruebas, no es necesario repetir las sentencias de creación de las bases de datos: el truncado en Cassandra no elimina el esquema de la base de datos y el borrado de todos los documentos de la colección no elimina los índices creados en MongoDB sobre la misma, de forma que tras el borrado de todo el contenido, las bases de datos quedan preparadas para volver a repetir todo el proceso.

La forma en la que se lanzan los scripts para la realización de las pruebas es la siguiente:

```
sudo ./mongo_script_pruebas.sh >> salida_mongo.txt 2>> salida_mongo.txt
sudo ./cassandra_script_pruebas.sh >> salida_cassandra.txt 2>> salida_cassandra.txt
```

Capítulo 6

Análisis de los resultados

Tras haber realizado las pruebas de rendimiento y tomado las medidas de tiempo tal y como se indicó en el capítulo anterior, en este capítulo se presentarán los resultados obtenidos mediante tablas y gráficas, con el fin de poder analizar los resultados de una forma clara y eficaz.

Tras la presentación de los resultados obtenidos en las pruebas, se espera que el análisis de los mismos revele cual de estos motores de bases de datos tiene un mejor desempeño.

Antes de realizar dicho análisis y de ver los resultados de las pruebas realizadas, en función de lo visto en los anteriores capítulos, se realizan las siguientes **hipótesis** sobre el comportamiento de estas bases de datos ante las pruebas:

1. Las operaciones de inserción y borrado de todos los datos serán las más costosas en términos computacionales y de tiempo para ambas bases de datos.

2. Las operaciones de inserción tardarán aproximadamente lo mismo para ambas bases de datos, pues han de ordenarlos (indexarlos) según se insertan.

3. Las búsquedas en Cassandra, al emplear valores que están fuera de la clave primaria, deberían ser más lentas.

4. Debido a lo restrictiva que ha resultado ser Cassandra para las actualizaciones de los datos almacenados, se presupone que deberán ser más rápidas.

Como parte del análisis final, se comprobará si estas hipotéticas suposiciones se muestran como correctas, o si por el contrario, algunas de ellas –o todas– están equivocadas.

6.1. MongoDB

Los datos obtenidos al realizar las pruebas sobre MongoDB se encuentran reflejados en las 3 tablas siguientes:

Portátil I5	Modelo 1					Modelo 2				
	R1	R2	R3	R4	R5	R1	R2	R3	R4	R5
Inserciones	19m 14.36s	20m 8s	20m 54.6s	18m 27.24s	18m 25.65s	3m 4.09s	3m 9.78s	2m 47.71s	2m 47.27s	2m 47.63s
Búsqueda	35.780s	33.24s	29.44s	30.80s	29.99s	2.88s	3.08s	2.64s	2.65s	2.57s
Actualización	3.34s	2.97s	2.74 s	2.41s	2.62s	0.11s	0.107s	0.09s	0.1s	0.096s
Borrado	14m 34.38s	14m 14.89s	12m 8.95s	12m 8.20s	12m 11.72s	1m 55.46s	1m 59.95s	1m 41.2s	1m 42.06s	1m 41.92s

Tabla 6.1: Datos obtenidos en las pruebas de MongoDB sobre el portátil i5

Máquina Virtual	Modelo 1					Modelo 2				
	R1	R2	R3	R4	R5	R1	R2	R3	R4	R5
Inserciones	27m 18,83s	28m 20,19s	28m 30,71s	28m 43,3s	28m 38,21s	4m 10,60s	4m 12,36s	4m 17,68s	4m 13,47s	4m 14,47s
Búsqueda	1m 2,50s	1m 3,52s	1m 6,10s	1m 5s	1m 2,19s	4.80s	4,56s	4,87s	4,91s	4,74s
Actualización	5,12s	5,60s	5,91s	5,85s	6,13s	0.10s	0.11s	0,10s	0,12s	0,12s
Borrado	30m 4,43s	31m 0,21s	32m 8,72s	31m28,12s	31m 56,71	4m 50,22s	4m 56,89s	5m 10,19s	4m 54,26s	5m 1,3s

Tabla 6.2: Datos obtenidos en las pruebas de MongoDB sobre la máquina virtual

Portátil I3	Modelo 1					Modelo 2				
	R1	R2	R3	R4	R5	R1	R2	R3	R4	R5
Inserciones	43m 30,61s	54m 49,84s	41m 58,45s	49m 7,69s	54m 42,88s	5m 9,08s	4m 36,92s	4m 53,61s	4m 44,597ss	4m 50,81s
Búsqueda	1m 35,79s	1m 5,41s	58,16s	1m 5,02s	59,90s	11,46s	9,98s	10,88s	7,80s	8,88s
Actualización	19,21s	7,53s	6,37s	8,99s	7,40s	2,31s	0,18s	0,13s	0,23s	0,23s
Borrado	38m 53,28s	33m 37,68s	38m 57,14s	35m 21,64s	37m 19,09s	3m 25,19s	3m 14,14s	3m 16,46s	3m 23,74s	3m 26,85s

Tabla 6.3: Datos obtenidos en las pruebas de MongoDB sobre el portátil I3

6.2. Cassandra

Los datos obtenidos al realizar las pruebas sobre Cassandra se encuentran reflejados en las 3 tablas siguientes:

Portátil I5	Modelo 1					Modelo 2				
	R1	R2	R3	R4	R5	R1	R2	R3	R4	R5
Inserciones	43m 53,94s	43m 26,51s	43m 30,75s	43m 23,37s	43m 30,40s	7m 58,33s	7m 56,65s	7m 56,54s	7m55,91s	7m 58,88s
Búsqueda	error	error	error	error	error	error	error	error	error	error
Actualización	0,64s	0,62s	0,46s	0,53s	0,60s	0,63s	0,47s	0,48s	0,55s	0,61s
Borrado	1,53s	0,74s	0,73s	0,85s	0,71s	0,99s	1,01s	0,72s	0,80s	1,11s

Tabla 6.4: Datos obtenidos en las pruebas de Cassandra sobre el portátil I5

Máquina virtual	Modelo 1					Modelo 2				
	R1	R2	R3	R4	R5	R1	R2	R3	R4	R5
Inserciones	91m 16,04s	91m 53,86s	95m 38,51s	94m12,19s	95m 23,53s	16m 43,13s	17m 48,52s	18m 23,85s	17m 5,21s	17m 36,92s
Búsqueda	error	error	error	error	error	error	error	error	error	error
Actualización	0,60s	0,59s	0,64s	0,61s	0,60s	0,58s	0,61s	0,60s	0,61s	0,58s
Borrado	2,37s	2,24s	2,01s	1,23s	1,94s	1,03s	0,93s	1,88s	1,51s	1,54s

Tabla 6.5: Datos obtenidos en las pruebas de Cassandra sobre la máquina virtual

Portátil I3	Modelo 1					Modelo 2				
	R1	R2	R3	R4	R5	R1	R2	R3	R4	R5
Inserciones	57m 33,95s	57m 12,68s	57m 40,28s	59m 9,14s	57m 11s	10m 34,44s	10m 31,76s	10m 26,45s	10m 34,54s	10m 31,34
Búsqueda	error	error	error	error	error	error	error	error	error	error
Actualización	0,59s	0,53s	0,75s	0,52s	0,55s	0,56s	0,80s	0,74s	0,66s	0,72s
Borrado	2,98s	2,28s	3,07s	1,59s	4,51s	2,14s	4,08s	3,82s	3,30s	5,18s

Tabla 6.6: Datos obtenidos en las pruebas de Cassandra sobre el portátil I3

Los datos obtenidos en las pruebas de este gestor de bases de datos han sido anómalos en 2 de las 4 operaciones realizadas: búsqueda y borrado.

A la hora de realizar las búsquedas en Cassandra, en todas las ejecuciones del script, al llegar a la sección de búsqueda han arrojado el siguiente error:

ReadFailure: Error from server: code=1300. Replica(s) failed to execute read. message="Operation failed - received 0 responses and 1 failures" info='failures': 1, 'received_responses': 0, 'required_responses':1, 'consistency': 'ONE'

Error que al consultarlo en la web, resulta ser un error genérico muy habitual en Cassandra y puede estar originado por múltiples motivos. Tras indagar en los diferentes logs que genera la base de datos, se encontró en */var/log/cassandra/debug.log* la siguiente línea:

```
<SELECT * FROM opcion1.phone_watch WHERE sensor = accelerometer AND model = lgwatch
AND x > -6.9 AND z < 2.3 AND >, total time 5006 msec, timeout 5000 msec
```

Que viene a indicar que la operación se ha cancelado por alcanzar un estado de *timeout*. Así pues, para poder repetir las pruebas, se ha realizado la siguiente modificación en el fichero de configuración *cassandra.yaml*, modificando los valores para hacerlos lo suficientemente grandes como para que no puedan interferir con la operación de búsqueda.

```
# How long the coordinator should wait for read operations to complete
read_request_timeout_in_ms: 50000000
# How long the coordinator should wait for seq or index scans to complete
range_request_timeout_in_ms: 10000000
# The default timeout for other, miscellaneous operations
request_timeout_in_ms: 10000000
```

Y tras reiniciar el servicio de cassandra, se han repetido las pruebas, arrojando esta vez un resultado válido para el portátil I5, pero saltando otro error diferente para la máquina virtual y el portátil I3:

OperationTimedOut: errors=127.0.0.1: Client request timeout. See Session.execute[async](timeout), last_host=127.0.0.1

Este error se ha esquivado introduciendo un parámetro adicional en la sentencia cqlsh que se lanza por consola, siendo 12.000 un número arbitrario lo suficientemente alto como para asegurar que no haya ningún problema más de tipo *request timeout*:

```
time cqlsh --request-timeout 12000
```

	Modelo 1					Modelo 2				
	R1	R2	R3	R4	R5	R1	R2	R3	R4	R5
Búsqueda I5	8,00s	8,02s	8,06s	8,03s	8,11s	4,01s	3,83s	3,77s	3,71s	3,71s
Búsqueda MV	22,69s	20,81s	20,23s	19,86s	19,75s	11,35s	10,989s	10,91s	10,01s	9,91s
Búsqueda I3	16,75s	10,92s	10,37s	10,21s	10,27s	12,24s	6,24s	6,12s	6,12s	6,10s

Tabla 6.7: Datos obtenidos al repetir las pruebas de búsqueda sobre Cassandra.

Por su parte, el truncado/borrado de la base de datos en Cassandra es, en todo momento, un proceso extremadamente rápido según los datos obtenidos. Datos que no parecen tener ningún sentido tras observar que Cassandra es objetivamente más lenta que MongoDB en las operaciones de escritura –inserciones– realizadas previamente.

Esto es, al parecer, debido a la forma en que Cassandra realiza esta operación. En lugar de borrar del disco una a una todas las filas, marca el contenido de toda la tabla como *borrado* y posteriormente, según el comportamiento por defecto, realiza un *snapshot* de todos los datos almacenados, para posibilitar su posterior recuperación en caso de ser necesario. Este comportamiento puede deshabilitarse en el fichero de configuración de la base de datos.

En cualquier caso, los datos dejan de estar accesibles inmediatamente tras el truncado de la tabla, aunque el espacio en disco no se recupera inmediatamente.

También es interesante la salida del comando *time* para cassandra, pues en algunas de las inserciones se observa algo similar a esto:

```
*** -> Empezando repeticion: 5
Poblando base de datos Cassandra en opcion1

real 43m30,405s
user 97m12,845s
sys 4m2,408s
```

Siendo el tiempo de usuario mucho mayor que el tiempo real, que proporcionan tanto el comando time como el propio servicio de cassandra tras finalizar la inserción masiva de datos.

6.3. Comparación gráfica

Tras la presentación de los tiempos obtenidos en las 5 repeticiones establecidas para cada base de datos y cada modelo en cada máquina de pruebas, en esta sección se presenta una comparativa gráfica entre los **tiempos medios** obtenidos por cada gestor para cada modelo.

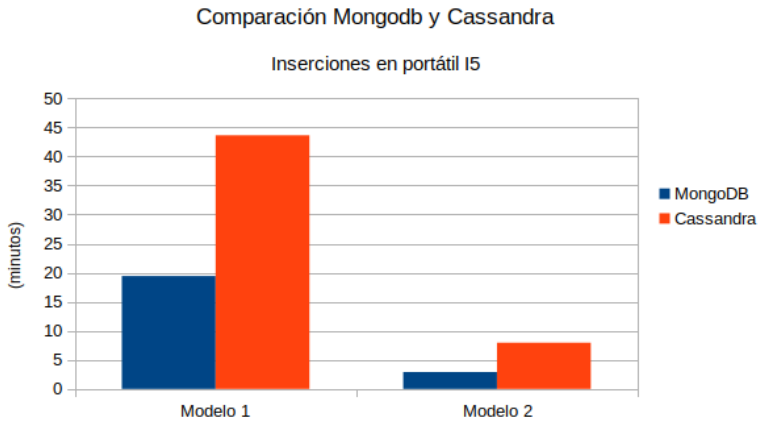

Figura 6.1: Comparación de tiempos de inserción entre MongoDB y Cassandra en I5.

En la gráfica 6.1 se puede ver la fuerte correlación existente entre el tiempo y el número de instancias a insertar. También es palpable el hecho de que Cassandra necesita aproximadamente el doble de tiempo que MongoDB para realizar el mismo trabajo de inserción masiva.

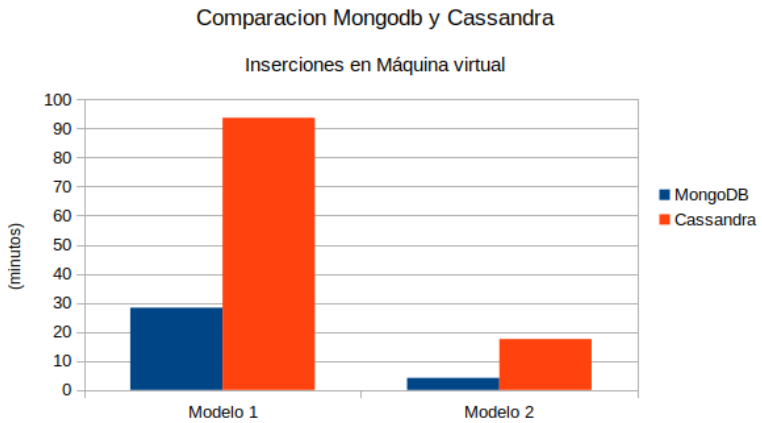

Figura 6.2: Comparación de tiempos de inserción entre MongoDB y Cassandra en MV.

En la gráfica 6.2, que muestra la comparación de tiempo de inserción en la máquina virtual, tenemos una situación curiosa, pues los tiempos para importar los datos en MongoDB son ligeramente mayores que a los vistos en la *máquina I5*, lo que simplemente indica que este portátil es algo más potente que

la máquina virtual, pero resulta un tanto extraño que los tiempos para las inserciones de Cassandra se disparen en la máquina virtual, siendo más de 3 veces superiores a los tiempos de MongoDB, y siendo más del doble de los tiempos de Cassandra en el portátil I5.

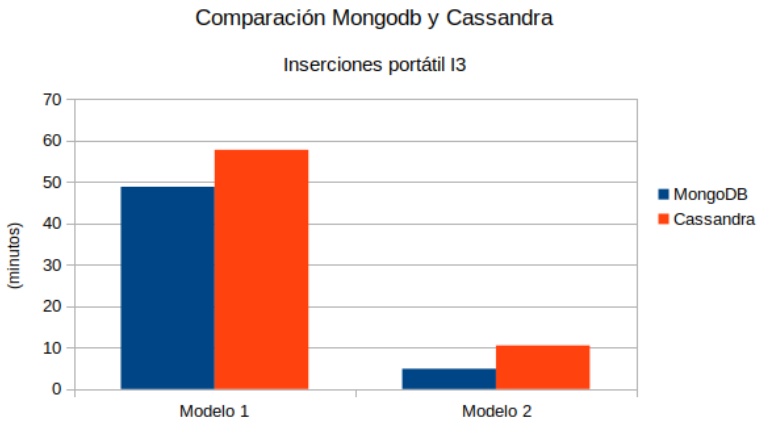

Figura 6.3: Comparación de tiempos de inserción entre MongoDB y Cassandra en I3.

Por su parte, la gráfica 6.3 muestra una situación más paritaria en la *máquina I3* entre ambas bases de datos y en ambos modelos. Los resultados para MongoDB son mayores que los vistos en las otras dos máquinas –lo cual resulta bastante normal, pues parece poseer el hardware más modesto– pero los de Cassandra son ostensiblemente menores que los encontrados en la máquina virtual.

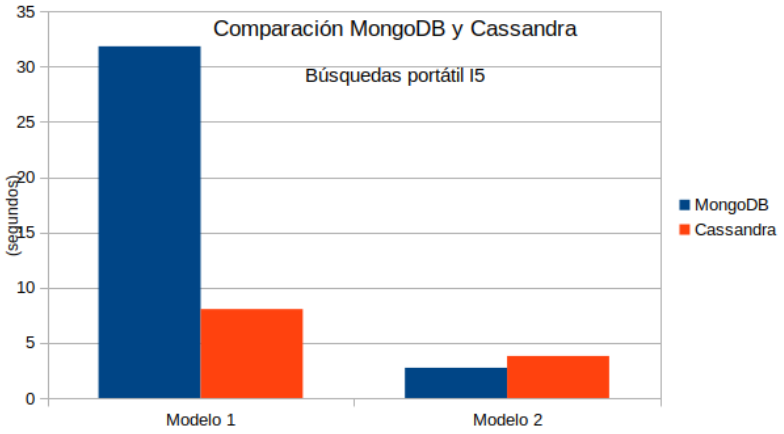

Figura 6.4: Comparación de tiempos de búsqueda entre MongoDB y Cassandra en I5.

En cuanto a la comparación de tiempos de búsqueda, los resultados encontrados son realmente curiosos e interesantes. Empezando por la máquina I5, en la gráfica 6.4 tenemos que las búsquedas en Cassandra para ambos modelos están muy próximas en tiempo, siendo algo menores para el segundo modelo, que contiene menos datos.

Lo que se hace evidente tanto en esta gráfica como en las gráficas 6.5 y 6.6 es que el tiempo de búsqueda en Cassandra no depende directamente de la cantidad de datos contenidos, a diferencia de MongoDB, donde se ve claramente que sí tiene esa dependencia directa.

También es destacable la brutal diferencia que esta dependencia sobre la cantidad de datos genera en ambos modelos, teniendo en cuenta el hardware. Para el modelo con más cantidad de datos, la búsqueda en MongoDB es aproximadamente tres veces más lenta tanto para el portátil I5 como para la máquina virtual, pero es 6 veces más lenta para el portátil I3. Por el otro lado, encontramos que para el segundo modelo, MongoDB es ostensiblemente más rápido en las dos máquinas más potentes, y ligeramente más lento en la máquina modesta.

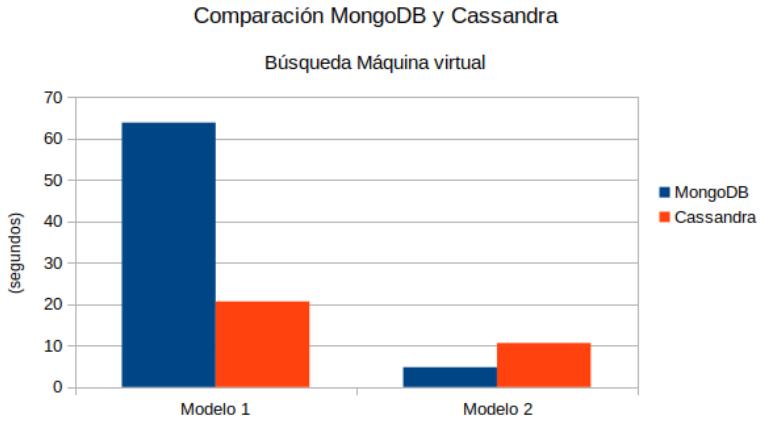

Figura 6.5: Comparación de tiempos de búsqueda entre MongoDB y Cassandra en MV.

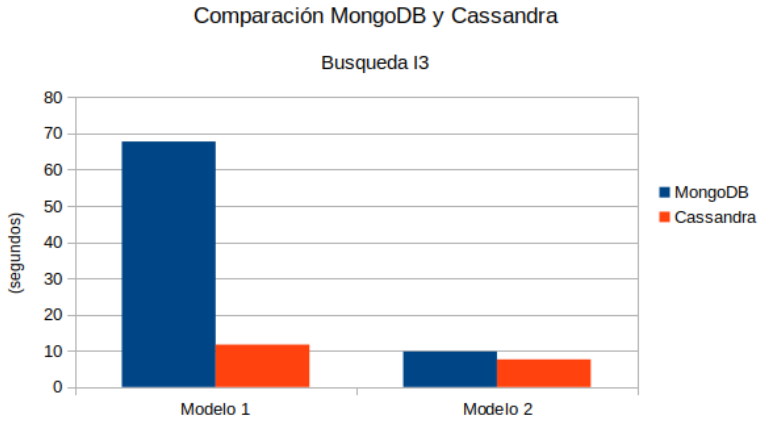

Figura 6.6: Comparación de tiempos de búsqueda entre MongoDB y Cassandra en I3.

Para finalizar la parte relativa a la búsqueda, se hace necesario señalar que nuevamente el desempeño de cassandra en la máquina virtual ha sido peor que en los dos portátiles.

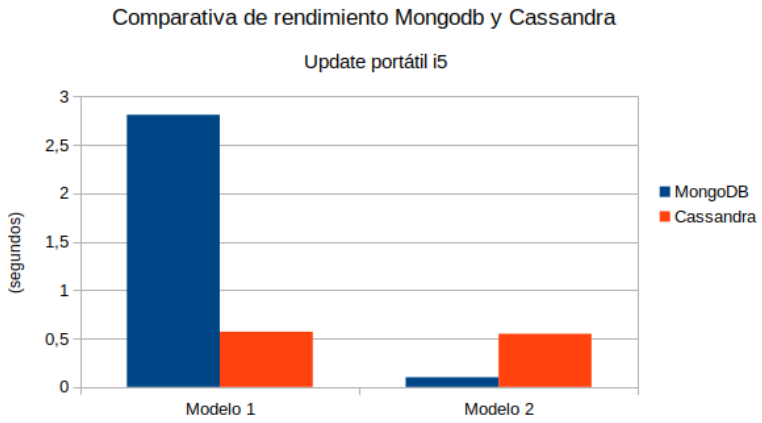

Figura 6.7: Comparación de tiempos de actualización entre MongoDB y Cassandra en I5.

En cuanto al tiempo necesario para realizar la operación de actualización, en las gráficas 6.7, 6.8 y 6.9 se ve con meridiana claridad que Cassandra resulta muy estable, teniendo unos tiempos de actualización casi idénticos para ambos modelos en las 3 máquinas, independientemente de la cantidad de datos manejada y de la potencia del hardware.

También se ve, al igual que con la búsqueda, que el esfuerzo que realiza MongoDB es proporcional-

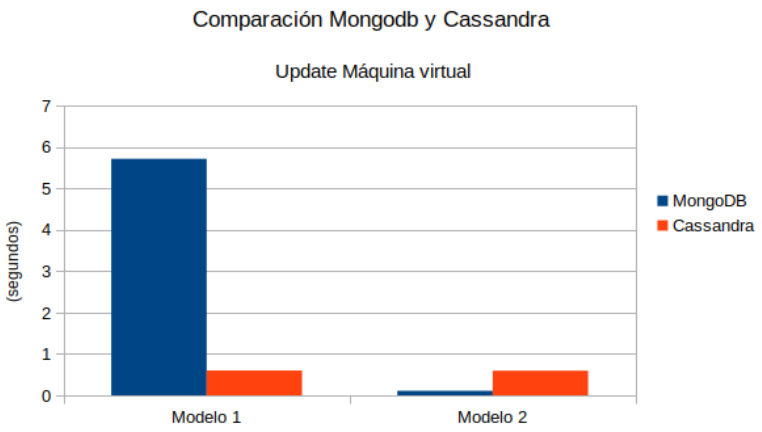

Figura 6.8: Comparación de tiempos de actualización entre MongoDB y Cassandra en MV.

mente lineal al volumen de datos, de forma que para las 3 máquinas, el tiempo obtenido por mongo es categóricamente mayor que el de cassandra cuando se trabaja sobre el primer modelo, pero bastante inferior cuando se utiliza el segundo modelo.

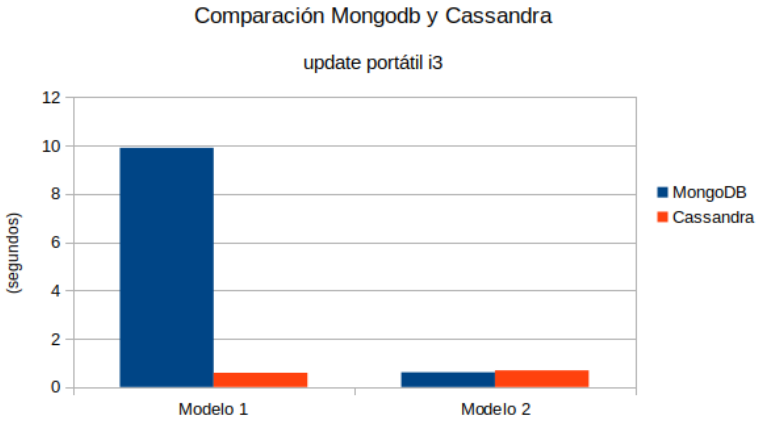

Figura 6.9: Comparación de tiempos de actualización entre MongoDB y Cassandra en I3.

En el caso del borrado completo de la base de datos, tal y como se pudo ver en las tablas que recogían la información de las ejecuciones, la diferencia entre las dos bases de datos es inmensa, pues en MongoDB se emplea un tiempo ligeramente inferior al de las inserciones de datos –salvo en la máquina virtual, que era un poco mayor–, mientras que en Cassandra la operación de truncado se realiza en unos pocos segundos.

Debido a esta abismal diferencia encontrada entre los tiempos obtenidos para borrar toda la base de datos, no se ha creado una gráfica comparativa, pues se considera que no aportaría nada relevante, ya que se hace más que evidente que los dos SGBD realizan la operación de borrado de una manera muy diferente, que no es comparable.

6.4. Valoración de las hipótesis formuladas y conclusiones extraídas del experimento

Al inicio de este capítulo se realizaron 4 hipótesis en base al conocimiento adquirido al realizar este trabajo, sobre el funcionamiento de MongoDB y Cassandra. Estas hipótesis valoraban la forma teórica en la que las dos bases de datos realizan las diferentes operaciones y *predecían su comportamiento* en un entorno real.

La primera hipótesis planteaba que la inserción y el borrado serían las operaciones que más esfuerzo requerirían, al tratarse de operaciones de escritura. Y al menos para MongoDB y la inserción en Cassandra sí resultó ser cierta. No obstante, como ya se ha comentado, el borrado en Cassandra no se realiza de manera directa e inmediata, si no que se marca la tabla entera como "vacía", borrando su estructura interna llamada *sstable* y por tanto, es un proceso muy rápido.

La segunda hipótesis indicaba que puesto que ambas bases de datos se verían obligadas a indexar los datos antes de realizar la escritura de los mismos, tendrían un desempeño aproximado. Nada más lejos de la realidad, pues el desempeño de Mongo en este aspecto ha sido claramente muy superior.

La tercera predicción resultó estar parcialmente errada, pues postulaba que para Cassandra resultaría demasiado costoso realizar operaciones de búsqueda que contuviesen valores fuera de su clave primaria. Lo que se ha podido ver con este experimento es que la búsqueda en Cassandra depende muy poco de la cantidad de información y es bastante estable. Siendo normalmente más lenta que mongodb cuando la cantidad de información es menor, pero mucho más rápida cuando el volumen de datos resulta mayor.

Un caso idéntico al de la tercera hipótesis resulta la cuarta, pues apostaba a que la actualización de datos en cassandra debería ser más rápida que en mongodb, debido a lo restrictiva que resultó ser esa operación. Y al igual que con la búsqueda, esto es cierto solamente para volúmenes de datos mayores.

Además, se han llegado a las siguientes conclusiones:

- El rendimiento de MongoDB depende linealmente del volumen de datos que maneja.
- El proceso de escritura en Cassandra es costoso y sí que se puede apreciar una relación directa entre el tiempo necesario para realizar las operaciones, el volumen de datos y el hardware de la máquina.
- Cassandra resulta un sistema gestor de bases de datos NoSQL muy estable. Los tiempos de lectura no varían demasiado entre sí, ni en función de la carga de datos ni del hardware que lo soporta, dotando a este sistema de una muy buena predictibilidad y estabilidad.
- De igual manera, la actualización de datos en Cassandra ha requerido un tiempo muy parecido en los 2 modelos de datos y en las 3 máquinas donde se han hecho las pruebas.
- El borrado completo de la base de datos no es realmente comparable, pues cada uno lo ejecuta de una manera muy diferente.
- Teniendo en cuenta las restricciones y los errores encontrados al realizar las pruebas sobre Cassandra, se puede afirmar que MongoDB resulta mucho más usable y flexible que Cassandra.
- En función de los datos obtenidos en el experimento, y que han sido reflejados en las tablas del inicio de este capítulo, se han encontrado unos *valores outliers* llamativos para las primeras sentencias de actualización ejecutadas por MongoDB y para las primeras búsquedas realizadas en Cassandra, ambos en el portátil I3 y para ambos modelos. Se desconoce si este valor anómalo se debe a un suceso ajeno ocurrido durante la ejecución de dicha prueba o si por el contrario tiene que ver con la existencia de estructuras de *cachés internas*, pero así lo parece.

Capítulo 7

Conclusiones

El mundo ha cambiado mucho en este último año. El mundo está cambiando en el momento de escribir estas últimas líneas. Una de las conclusiones más evidentes extraíbles de la realización de este trabajo es que el modelo relacional, que nació hace casi medio siglo, pese a su solvencia, refinamiento e implantación, se ve desbordado ante el *data deluge* –gran diluvio de los datos– que necesariamente implican las tecnologías más punteras y prometedoras hoy en día: Big Data, Inteligencia Aritificial, IoT, Visión Artificial...

Por si esto no fuese suficiente, el paradigma de consumo de los usuarios de internet está cada vez más centrado en la inmediatez y la disponibilidad más absoluta. Lo queremos todo ya, y disponible en todo momento –por si acaso–. Estas necesidades ponen de manifiesto las limitaciones del modelo ACID y presentan a BASE como una alternativa mucho más sólida y solvente de lo que puede parecer inicialmente.

Este nuevo mundo ha puesto unas grandes expectativas y ha generado un gran interés en la nueva idea que suponen las bases de datos No sólo SQL, y en concreto sus 4 tipos más conocidos: clave-valor, documental, orientadas a columnas y de grafos. Cada uno de estos tipos posee unas características especiales y concretas, que les hace especialmente aptos y solventes para problemas muy concretos, aunque siempre con el foco en la escalabilidad, la disponibilidad, la velocidad y el rendimiento.

Dentro de estas bases de datos NoSQL, hemos encontrado que los dos sistemas objeto de este estudio se han mantenido en los últimos años como los sistemas NoSQL más populares de sus tipos, y siendo además, de los más populares en la actualidad en relación a todas las bases de datos comerciales existentes. También vimos que MongoDB y Cassandra difieren en prácticamente todo, perteneciendo de entrada, cada uno, a una sección diferente del triángulo CAP.

Si bien se excluyó de la parte práctica de este trabajo, una de las cosas más relevantes que hemos encontrado sobre estas bases de datos es su gran componente distribuido. Naturalmente, es posible utilizarlas en un único nodo, pero están diseñadas y su potencia se revela al emplearlas trabajando con múltiples nodos, en un clúster que extraiga todo el poder del escalado horizontal que caracteriza a estos sistemas NoSQL.

Tras la conclusión de la parte de carácter más teórico, se pasó a trabajar directamente con ambas bases de datos, siendo esta la parte que me ha permitido extraer más conclusiones a nivel de sensaciones, además de las conclusiones analíticas extraídas a partir de los datos del experimento.

Así, puedo afirmar que la experiencia de usuario con MongoDB ha sido estupenda; el lenguaje javascript es sencillo de usar y se aprende a utilizar el modelo documental muy rápido. Inicialmente parecía una base de datos muy lejana a las bases de datos relacionales, pero al realizar este trabajo, he podido ver que Mongo ha ido pivotando en su propia concepción, acercándose más al mundo relacional, pero manteniendo la esencia de las bases de datos documentales. De esta forma, se puede afirmar que a diferencia de la

mayoría de bases de datos NoSQL, MongoDB se ha logrado postular como una base de datos para resolver problemas generales, y no nichos específicos.

Por contraparte, la experiencia de usuario con Cassandra ha sido bastante frustrante; comenzando por la problemática de la documentación incompleta oficial y la dificultad añadida de tener que distinguir si la información encontrada relativa a varias cuestiones, era propia de Apache Cassandra o de la versión propiedad de la empresa DataStax.

También se hace extraño el hecho de que inicialmente, Cassandra se parece mucho a las bases de datos SQL clásicas. Sin embargo, al poco de empezar a usarla se puede ver muy claramente hasta que punto se diferencia de ellas, quedándose el parecido en una cuestión estética.

Como se ha visto a la hora de realizar las pruebas, la sentencia de actualización se tuvo que adaptar debido a las grandes limitaciones que imponía esta base de datos, y las búsquedas arrojaron errores recurrentes. Todo esto muy alejado de la naturaleza Plug-And-Play que la que hace gala MongoDB.

En relación a las pruebas, se optó por trabajar en un escenario equitativo y equivalente para ambas bases de datos. Se consideró que el conjunto de datos elegido, quizás se pudiese modelar de una forma óptima para ambas bases de datos, pero eso incluía la dificultad añadida de *valorar la adecuación del modelado óptimo*, corriendo el riesgo de que, por muchas razones posibles, uno de ellos fuese mejor, y los resultados del rendimiento obtenidos realmente no sirviesen para nada.

Con los modelos creados, se ha evitado deliberadamente el escenario del mejor caso posible, y se ha tratado de encontrar un equilibrio con los recursos disponibles. Pese a disponer de un espacio en disco limitado, al fraccionar los datos en dos conjuntos de tamaño bien diferenciado (uno cercano a los 4GB y otro cercano a los 700MB), se ha podido ver como responden ambas bases de datos ante diferentes volúmenes, obteniéndose unos resultados realmente dispares e ilustrativos.

Tras el análisis de los dos dos sistemas de bases de datos y de los datos obtenidos por el experimento, hay que decir que si bien, en términos personales, MongoDB sería mi apuesta para usarlo como base de datos de propósito general, la tremenda estabilidad que ha mostrado Cassandra es más que digna de ser mencionada y valorada positivamente.

No cabe ninguna duda de que Cassandra puede ser una aplicación extraordinaria ante problemas específicos, que requieran un modelado adecuado para sus estructuras y formas de trabajar.

A modo de conclusión final, se hace necesario decir que este trabajo podría ser fácilmente continuado y expandido explorando la naturaleza distribuida de estos sistemas gestores de bases de datos, realizando un estudio teórico mucho más profundo sobre las técnicas de sharding y replicación, y un análisis de rendimiento sobre clústeres de varios nodos. Cabe decir que sería realmente interesante comparar los resultados obtenidos en este trabajo con las pruebas sobre un nodo y los que se podrían obtener al realizar los test sobre los nodos trabajando en conjunto.

Referencias

[1] (2018) La Era de la Información. Wikipedia.org. [Online]. Available: https://es.wikipedia.org/wiki/Era_de_la_informaci%C3%B3n

[2] (2017) ¿Se podría calcular el peso real de todo el contenido de internet? redeszone.net. [Online]. Available: https://www.redeszone.net/2017/11/26/se-podria-calcular-peso-real-contenido-internet/

[3] (2018) Edge Computing: qué es y por qué hay gente que piensa que es el futuro. Xataka. [Online]. Available: https://www.xataka.com/internet-of-things/edge-computing-que-es-y-por-que-hay-gente-que-piensa-que-es-el-futuro

[4] N. Leavitt, "Will nosql databases live up to their promise?" *Computer*, vol. 43, no. 2, pp. 12–14, Feb. 2010. [Online]. Available: https://doi.org/10.1109/MC.2010.58

[5] (2018) Presente y futuro de las bases de datos relacionales con Big Data. Hiberus Tecnología. [Online]. Available: https://www.hiberus.com/crecemos-contigo/presente-futuro-las-bases-datos-relacionales-big-data/

[6] P. Sharma and J. Prabha, "Performance Evaluation of Cloud Based Query Processing with MongoDB and Cassandra on Assorted Parameters," April 2018. [Online]. Available: https://ssrn.com/abstract=3166503

[7] A. Gupta, S. Tyagi, N. Panwar, S. Sachdeva, and U. Saxena, "Nosql databases: Critical analysis and comparison," in *2017 International Conference on Computing and Communication Technologies for Smart Nation (IC3TSN)*, Oct 2017, pp. 293–299. [Online]. Available: https://ieeexplore.ieee.org/document/8284494/

[8] K. Fraczek and M. Plechawska-Wojcik, "Comparative analysis of relational and non-relational databases in the context of performance in web applications," in *Beyond Databases, Architectures and Structures. Towards Efficient Solutions for Data Analysis and Knowledge Representation*, S. Kozielski, D. Mrozek, P. Kasprowski, B. Małysiak-Mrozek, and D. Kostrzewa, Eds. Cham: Springer International Publishing, 2017, pp. 153–164. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-319-58274-0_13

[9] K. Bhamra. (2017) A Comparative Analysis of MongoDB and Cassandra. University of Bergen. [Online]. Available: http://bora.uib.no/handle/1956/17228

[10] D. Escudero Romero. (2015) Estudio del SGDB Cassandra. Universidad de Valladolid. [Online]. Available: http://uvadoc.uva.es/handle/10324/15204

[11] N. Román Seneque. (2014) Administración en MongoDB. Universidad de Valladolid. [Online]. Available: http://uvadoc.uva.es/handle/10324/7898

[12] R. García Cebreiros. (2016-07-21) Análisis de bases de datos y tendencias tecnológicas. un caso de uso en twitter para aplicaciones de análisis de sentimientos y opiniones. Universidad de Alicante. [Online]. Available: http://hdl.handle.net/10045/57065

[13] V. J. BAS ABAD. (2015) Estudio comparativo de BBDD relacionales y NoSQL en un entorno industrial. Universidad Politécnica de Valencia. [Online]. Available: http://hdl.handle.net/10251/55530

[14] (2018) DB Engine. [Online]. Available: https://db-engines.com/en/

[15] (2018) Data Never Sleeps 6.0. DOMO.com. [Online]. Available: https://www.domo.com/learn/data-never-sleeps-6

[16] (2019) Data Never Sleeps 7.0. DOMO.com. [Online]. Available: https://www.domo.com/learn/data-never-sleeps-7

[17] Ecosistema Big Data. paradigmadigital.com. [Online]. Available: https://www.paradigmadigital.com/ecosistema-big-data/

[18] (2017) ¿Qué es NoSQL y qué Big Data? edgartec.com. [Online]. Available: http://edgartec.com/que-es-nosql-y-que-bigdata/

[19] J. Gray. (1981) The Transaction Concept: Virtues and Limitations. [Online]. Available: http://jimgray.azurewebsites.net/papers/thetransactionconcept.pdf

[20] T. Haerder and A. Reuter, "Principles of transaction-oriented database recovery," *ACM Comput. Surv.*, vol. 15, no. 4, pp. 287–317, Dec. 1983. [Online]. Available: http://doi.acm.org/10.1145/289.291

[21] B. Ippolito. (2009) Drop ACID and think about data. PyCOn 2009. [Online]. Available: https://bob.ippoli.to/python/pycon/archives/2009/04/01/pycon-2009-drop-acid-and-think-about-data

[22] C. Strauch. (2011) NoSQL Databases. Stuttgart Media University. [Online]. Available: http://www.christof-strauch.de/nosqldbs

[23] E. Brewer. (2000) Towards Robust Distributed Systems. [Online]. Available: http://www.cs.berkeley.edu/~brewer/cs262b-2004/PODC-keynote.pdf

[24] Bases de datos NoSQL. Qué son y tipos que nos podemos encontrar. Acens. [Online]. Available: https://www.acens.com/wp-content/images/2014/02/bbdd-nosql-wp-acens.pdf

[25] (2017) NoSQL y sus tipos. Smart Base. [Online]. Available: http://smartbasegroup.com/bds-nosql-tipos/

[26] (2018) NoSQL Databases. [Online]. Available: http://nosql-database.org/

[27] O. Blancarte. (2017) Escalabilidad Horizontal y Vertical. [Online]. Available: https://www.oscarblancarteblog.com/2017/03/07/escalabilidad-horizontal-y-vertical/

[28] (2015) NoSQL vs SQL: Principales diferencias y cuándo elegir cada una de ellas. PandoraFMS. [Online]. Available: https://blog.pandorafms.org/es/nosql-vs-sql-diferencias-y-cuando-elegir-cada-una/

[29] (2017) Bases de datos NoSQL : Guía definitiva. PandoraFMS. [Online]. Available: https://blog.pandorafms.org/es/bases-de-datos-nosql/

[30] (2018) Repositorio de MongoDB en GitHub. Github. [Online]. Available: https://github.com/mongodb/mongo

[31] (2009) 32-bit limitations. MongoDB. [Online]. Available: https://www.mongodb.com/blog/post/32-bit-limitations

[32] (2017) farewell, Solaris. MongoDB. [Online]. Available: https://engineering.mongodb.com/post/farewell-solaris

[33] (2018) Our Customers. MongoDB Inc. [Online]. Available: https://www.mongodb.com/who-uses-mongodb

[34] (2017) Empresas que usan MongoDB. Openwebinars. [Online]. Available: https://openwebinars.net/blog/empresas-que-usan-mongodb/

[35] (2018) What is MongoDB? MongoDB Inc. [Online]. Available: https://www.mongodb.com/what-is-mongodb

[36] BSON types. MongoDB Inc. [Online]. Available: https://docs.mongodb.com/manual/reference/bson-types/

[37] Aggregation. MongoDB Inc. [Online]. Available: https://docs.mongodb.com/manual/aggregation/

[38] (2017) Ransomware groups have deleted over 10,000 MongoDB databases. computerworld. [Online]. Available: https://www.computerworld.com/article/3155260/security/ransomware-groups-have-deleted-over-10000-mongodb-databases.html

[39] (2018) Its the data. Stupid! Shodan.io. [Online]. Available: https://blog.shodan.io/its-the-data-stupid/

[40] (2018) Security Checklist. MongoDb Inc. [Online]. Available: https://docs.mongodb.com/manual/administration/security-checklist/

[41] (2018) GitHub Cassandra. GitHub. [Online]. Available: https://github.com/apache/cassandra

[42] (2018) DataStax Enterprise. DataStax. [Online]. Available: https://www.datastax.com/products/datastax-enterprise

[43] (2018) DataStax Distribution of Apache Cassandra. DataStax Academy. [Online]. Available: https://academy.datastax.com/planet-cassandra//cassandra

[44] (2017) No Datastax - Apache Cassandra 3.11 - Windows installation. YouTube. [Online]. Available: https://www.youtube.com/watch?v=Mn1xOHAH_dc

[45] (2016) Apache Cassandra. Apache Cassandra. [Online]. Available: https://cassandra.apache.org/

[46] (2018) Cassandra System Properties. DB-ENgine. [Online]. Available: https://db-engines.com/en/system/Cassandra#a34

[47] (2016) CQL - Data definition. Apache Cassandra. [Online]. Available: https://cassandra.apache.org/doc/latest/cql/ddl.html

[48] (2018) Internode Communications (Gossip). DataStax. [Online]. Available: https://docs.datastax.com/en/dse/6.0/dse-arch/datastax_enterprise/dbArch/archGossipAbout.html

[49] (2016) Snitch. Apache Cassandra. [Online]. Available: https://cassandra.apache.org/doc/latest/operating/snitch.html

[50] (2016) Tunable Consistency. Apache Cassandra. [Online]. Available: https://cassandra.apache.org/doc/latest/architecture/dynamo.html#tunable-consistency

[51] (2016) Big Data Security Tales: MongoDB Cassandra (Level 101). Un Informático en el Lado del Mal. [Online]. Available: https://www.elladodelmal.com/2016/07/big-data-security-tales-mongodb.html

[52] (2016) Security. Apache Cassandra. [Online]. Available: https://cassandra.apache.org/doc/latest/operating/security.html

[53] D. Dua and C. Graff, "UCI machine learning repository," University of California, Irvine, School of Information and Computer Sciences, 2017. [Online]. Available: http://archive.ics.uci.edu/ml

[54] (2015) Heterogeneity Human Activity Recognition Dataset. University of California, Irvine, School of Information and Computer Sciences. [Online]. Available: https://archive.ics.uci.edu/ml/datasets/Heterogeneity+Activity+Recognition

[55] (2019) Modelo entidad-relación. Wikipedia. [Online]. Available: https://es.wikipedia.org/wiki/Modelo_entidad-relaci%C3%B3n

[56] (2019) Installing Cassandra. Apache Cassandra. [Online]. Available: https://cassandra.apache.org/doc/latest/getting_started/installing.html

[57] (2019) Install MongoDB. MongoDB Inc. [Online]. Available: https://docs.mongodb.com/manual/tutorial/install-mongodb-on-ubuntu/

[58] (2017) time (Unix). Wikipedia. [Online]. Available: https://es.wikipedia.org/wiki/Time_(Unix)

Anexos

Anexo A

Contenido del soporte digital

El CD contiene el siguiente material:

- Documento PDF que contiene la memoria del proyecto (memoria.pdf)
- Carpeta *resultados*, donde están almacenados las salidas de los comandos ejecutados para la realización de las pruebas de rendimiento y las tablas empleadas para su análisis y obtención de los gráficos.
- Carpeta *scripts* donde se encuentran los scripts empleados para automatizar las pruebas realizadas.

Anexo B

Comandos y logs

I. Comandos para la modificación de la fuente de datos

Generar la primera linea (cabecera) modificada para la primera opción.

```
sed -n '1p' Watch_gyroscope.csv | sed '1 s/^/Tipe_Device,Sensor,/' > opcion1.csv
```

Los siguientes comandos vuelcan el contenido de los 4 archivos de datos origen en el final del archivo cuya cabecera se generó con el comando anterior, eliminando la primera línea de cada archivo correspondiente a la cabecera original.

```
sed '1d' Watch_gyroscope.csv | sed 's/^/watch,gyroscope,/'>> opcion1.csv
sed '1d' Watch_accelerometer.csv | sed 's/^/watch,accelerometer,/'>> opcion1.csv
sed '1d' Phones_gyroscope.csv | sed 's/^/phone,gyroscope,/'>> opcion1.csv
sed '1d' Phones_accelerometer.csv | sed 's/^/phone,accelerometer,/'>> opcion1.csv
```

Al generar el nuevo archivo con los nuevos atributos, se incrementa el peso del mismo, de forma que finalizar el proceso, el archivo resultante tiene las siguientes características:

```
wc -l opcion1.csv
33741501 opcion1.csv
du -h opcion1.csv
3,7G opcion1.csv
```

La siguiente secuencia de comandos genera los archivos que se emplearán para la opción 2, en la que los datos se separan en función del tipo del dispositivo:

```
sed -n '1p' Watch_gyroscope.csv | sed '1 s/^/Sensor,/' > opcion2_watch.csv
sed '1d' Watch_gyroscope.csv | sed 's/^/gyroscope,/'>> opcion2_watch.csv
sed '1d' Watch_accelerometer.csv | sed 's/^/accelerometer,/'>> opcion2_watch.csv
sed -n '1p' Phones_gyroscope.csv | sed '1 s/^/Sensor,/' > opcion2_phone.csv
sed '1d' Phones_gyroscope.csv | sed 's/^/gyroscope,/'>> opcion2_phone.csv
sed '1d' Phones_accelerometer.csv | sed 's/^/accelerometer,/'>> opcion2_phone.csv
```

II. Hardware de la máquina virtual

```
usuario@virtual:~$ lscpu
Architecture:        x86_64
CPU op-mode(s):      32-bit, 64-bit
Byte Order:          Little Endian
CPU(s):              2
On-line CPU(s) list: 0,1
Thread(s) per core:  1
Core(s) per socket:  2
Socket(s):           1
NUMA node(s):        1
Vendor ID:           GenuineIntel
CPU family:          15
Model:               6
Model name:          Common KVM processor
Stepping:            1
CPU MHz:             2394.268
BogoMIPS:            4788.53
Hypervisor vendor:   KVM
Virtualization type: full
L1d cache:           32K
L1i cache:           32K
L2 cache:            4096K
L3 cache:            16384K
NUMA node0 CPU(s):   0,1
```

```
    usuario@virtual:~$ free
                total        used        free      shared  buff/cache   available
Mem:         8168212      102448     7057104         680     1008660     7777556
Swap:         131068           0      131068
```

```
usuario@virtual:~$ sudo lsblk
NAME   MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
sr0     11:0    1 1024M  0 rom
vda    252:0    0   50G  0 disk
vda1 252:1    0    1M  0 part
vda2 252:2    0   50G  0 part /
```

III. Comandos para la realización de las pruebas

Para la realización de las pruebas, se ejecutan los comandos que se indican a continuación, desde la consola de comandos de Linux.
Poblar las bases de datos:

```
>>MongoDB
mongoimport --db opcion1 --collection phone_watch --type csv --headerline
--file /home/datos/opcion1.csv
mongoimport --db opcion2 --collection swatch --type csv --headerline
--file /home/datos/opcion2_watch.csv

>>Cassandra
cqlsh -k opcion1 -e "COPY phone_watch (type_device, sensor, indice, arrival_time,
creation_time, x, y, z, user, model, device, gt) FROM '/home/datos/opcion1.csv'
WITH HEADER = TRUE;"
cqlsh -k opcion2 -e "COPY watch (sensor, indice, arrival_time, creation_time, x, y, z,
user, model, device, gt) FROM '/home/datos/opcion2_watch.csv' WITH HEADER = TRUE;"
```

Ejecución de las búsquedas:

```
>>MongoDB
mongo opcion1 --eval 'db.phone_watch.find({"Sensor":"accelerometer","gt":"sit",
"User":"a","Model":"lgwatch","x": {$gt: -6.9},"z":{$lt:2.3}}).forEach(printjson)'
mongo opcion2 --eval 'db.swatch.find({"Sensor":"accelerometer","gt":"sit",
"User":"a","Model":"lgwatch","x": {$gt: -6.9},"z":{$lt:2.3}}).forEach(printjson)'

>>Cassandra
cqlsh localhost -k opcion1 -e "select * from phone_watch where sensor='accelerometer'
and gt='sit' and user='a' and model='lgwatch' and x >-6.9 and z<2.3 ALLOW FILTERING;"
cqlsh localhost -k opcion2 -e "select * from watch where sensor='accelerometer'
and gt='sit' and user='a' and model='lgwatch' and x >-6.9 and z<2.3 ALLOW FILTERING;"
```

Actualización de la fila elegida:

```
>>MongoDB
mongo opcion1 --eval 'db.phone_watch.updateOne({"Sensor":"accelerometer","gt":"sit",
"User":"a","Index":122214}, {$set: {"model":"newwatch"}})'
mongo opcion2 --eval 'db.swatch.updateOne({"Sensor":"accelerometer","gt":"sit",
"User":"a","Index":122214}, {$set: {"model":"newwatch"}})'

>>Cassandra
cqlsh localhost -k opcion1 -e "UPDATE phone_watch SET model ='newwatch'
where type_device='watch' and sensor='accelerometer' and gt='sit' and user='a'
and indice =122214;"
cqlsh localhost -k opcion2 -e "UPDATE watch SET model ='newwatch'
where sensor='accelerometer' and gt='sit' and user='a' and indice =122214;"
```

Borrado de la base de datos

```
>>MongoDB
mongo opcion1 --eval 'db.phone_watch.deleteMany({})'
mongo opcion2 --eval 'db.swatch.deleteMany({})'

>>Cassandra
cqlsh localhost -k opcion1 -e "TRUNCATE phone_watch;"
cqlsh localhost -k opcion2 -e "TRUNCATE watch;"
```

Anexo C

Scripts

I. inicializacion.sh

```
#!/bin/bash
#Creación de las bases de datos en MongoDB al crear las estructuras de índices
mongo opcion1 --eval 'db.phone_watch.createIndex({"Type_device":1,"Sensor":1,"Index":1,
"User":1,"gt":1})'
mongo opcion2 --eval 'db.swatch.createIndex({"Sensor":1,"Index":1,"User":1,"gt":1})'
#Creación de los keyspaces y las tablas de cassandra ejecutado el script para ello
cqlsh -f /home/scripts/creacion.cql
```

II. creacion.cql

Las siguientes instrucciones son cargadas para asegurar la correcta creación de los keyspaces y tablas de Cassandra.

```
CREATE KEYSPACE if not exists Opcion1
with replication= { 'class': 'SimpleStrategy', 'replication_factor': 1};
USE Opcion1;

CREATE TABLE if not exists phone_watch (
    type_device text,
    sensor text,
    indice int,
    arrival_time bigint,
    creation_time bigint,
    x double,
    y double,
    z double,
    user text,
    model text,
    device text,
    gt text,
    PRIMARY KEY((type_device,sensor,indice),user,gt));

CREATE KEYSPACE if not exists Opcion2
```

```
    with replication= { 'class': 'SimpleStrategy', 'replication_factor': 1};
    USE Opcion2;

    CREATE TABLE if not exists phone(
        sensor text,
        indice int,
        arrival_time bigint,
        creation_time bigint,
        x double,
        y double,
        z double,
        user text,
        model text,
        device text,
        gt text,
        PRIMARY KEY((sensor,indice),user,gt));

    CREATE TABLE if not exists watch (
        sensor text,
        indice int,
        arrival_time bigint,
        creation_time bigint,
        x double,
        y double,
        z double,
        user text,
        model text,
        device text,
        gt text,
        PRIMARY KEY((sensor,indice),user,gt));
```

III. mongo_script_pruebas.sh

```
#!/bin/bash
#Script para la realización de las pruebas en MongoDB
repeticiones=5

echo "   -> Numero de repeticiones: $repeticiones"
for nro in $( seq 1 $repeticiones)
do
opcion="opcion1"
# Fichero donde se almacena la información de las pruebas
fich_resultados="/home/resultados/mongo_$opcion.$nro.txt"
echo "     *** -> Empezando repeticion: $nro"
echo " Poblando base de datos MongoDB en $opcion"
time mongoimport --db opcion1 --collection phone_watch --type csv --headerline
```

```
--file /home/datos/opcion1.csv >> $fich_resultados
echo " Realizando búsqueda MongoDB en $opcion"
time mongo opcion1 --eval 'db.phone_watch.find({"Sensor":"accelerometer","gt":"sit",
"User":"a","Model":"lgwatch","x": {$gt: -6.9},"z":{$lt:2.3}}).forEach(printjson)'
>> $fich_resultados
echo " Realizando update MongoDB en $opcion"
time mongo opcion1 --eval 'db.phone_watch.updateOne({"Sensor":"accelerometer","gt":"sit",
"User":"a","Index":122214}, {$set: {"model":"newwatch"}})' >> $fich_resultados
echo "Borrando base de datos MongoDB en $opcion"
time mongo opcion1 --eval 'db.phone_watch.deleteMany({})' >> $fich_resultados
#Se pasa a emplear el segundo modelo
opcion="opcion2"
# Fichero donde se almacena la información de las pruebas
fich_resultados="/home/resultados/mongo_$opcion.$nro.txt"
echo " Poblando base de datos MongoDB en $opcion"
time mongoimport --db opcion2 --collection swatch --type csv --headerline
--file /home/datos/opcion2_watch.csv  >> $fich_resultados
echo " Realizando búsqueda MongoDB en $opcion"
time mongo opcion2 --eval 'db.swatch.find({"Sensor":"accelerometer","gt":"sit","User":"a",
"Model":"lgwatch","x": {$gt: -6.9},"z":{$lt:2.3}}).forEach(printjson)' >> $fich_resultados
echo " Realizando update MongoDB en $opcion"
time mongo opcion2 --eval 'db.swatch.updateOne({"Sensor":"accelerometer","gt":"sit",
"User":"a", "Index":122214}, {$set: {"model":"newwatch"}})' >> $fich_resultados
echo "Borrando base de datos MongoDB en $opcion"
time mongo opcion2 --eval 'db.swatch.deleteMany({})' >> $fich_resultados

# Se espera un tiempo antes de empezar la siguiente prueba
echo "     Tiempo de espera: 10 seg..."
sleep 10
done
exit 0
```

IV. cassandra_script_pruebas.sh

```
#!/bin/bash
#Script para la realización de las pruebas en Cassandra
repeticiones=5

echo "   -> Numero de repeticiones: $repeticiones"
for nro in $( seq 1 $repeticiones)
do
opcion="opcion1"
# Fichero donde se almacena la información de las pruebas
fich_resultados="/home/resultados/cassandra_$opcion.$nro.txt"
echo "     *** -> Empezando repeticion: $nro"
echo " Poblando base de datos Cassandra en $opcion"
```

```
time cqlsh -k opcion1 -e "COPY phone_watch (type_device, sensor, indice, arrival_time,
creation_time, x, y, z, user, model, device, gt) FROM '/home/datos/opcion1.csv'
WITH HEADER = TRUE;" >> $fich_resultados

echo " Realizando búsqueda Cassandra en $opcion"
time cqlsh localhost -k opcion1 -e "select * from phone_watch where sensor='accelerometer'
and gt='sit' and user='a' and model='lgwatch' and x >-6.9 and z<2.3
ALLOW FILTERING;" >> $fich_resultados

echo " Realizando update Cassandra en $opcion"
time cqlsh localhost -k opcion1 -e "UPDATE phone_watch SET model ='newwatch'
where type_device='watch' and sensor='accelerometer' and gt='sit' and user='a'
and indice =122214;" >> $fich_resultados

echo "Borrando base de datos Cassandra en $opcion"
time cqlsh localhost -k opcion1 -e "TRUNCATE phone_watch;" >> $fich_resultados

#Se pasa a emplear el segundo modelo
opcion="opcion2"
# Fichero donde se almacena la información de las pruebas
fich_resultados="/home/resultados/cassandra_$opcion.$nro.txt"
echo " Poblando base de datos Cassandra en $opcion"
time cqlsh -k opcion2 -e "COPY watch (sensor, indice, arrival_time, creation_time, x, y, z,
user, model, device, gt) FROM '/home/datos/opcion2_watch.csv'
WITH HEADER = TRUE;" >> $fich_resultados

echo " Realizando búsqueda Cassandra en $opcion"
time cqlsh localhost -k opcion2 -e "select * from watch where sensor='accelerometer'
and gt='sit' and user='a' and model='lgwatch' and x >-6.9 and z<2.3
ALLOW FILTERING;" >> $fich_resultados
echo " Realizando update Cassandra en $opcion"
time cqlsh localhost -k opcion2 -e "UPDATE watch SET model ='newwatch'
where sensor='accelerometer' and gt='sit' and user='a' and indice =122214;" >> $fich_resultados
echo "Borrando base de datos Cassandra en $opcion"
time cqlsh localhost -k opcion2 -e "TRUNCATE watch;" >> $fich_resultados

# Se espera un tiempo antes de empezar la siguiente prueba
echo "     Tiempo de espera: 10 seg..."
sleep 10
done
exit 0
```