eman ta zabal zazu
Universidad del País Vasco
Euskal Herriko Unibertsitatea
INFORMATIKA FAKULTATEA
FACULTAD DE INFORMÁTICA

Grado en Ingeniería Informática

Ingeniería del Software

Trabajo de Fin de Grado

Bases de Datos NoSQL para la aplicación TicketBai

Autor

Gorka Álvarez Marlasca

2021

eman ta zabal zazu
Universidad del País Vasco
Euskal Herriko Unibertsitatea
INFORMATIKA FAKULTATEA
FACULTAD DE INFORMÁTICA

Grado en Ingeniería Informática

Ingeniería del Software

Trabajo de Fin de Grado

Bases de Datos NoSQL para la aplicación TicketBai

Autor

Gorka Álvarez Marlasca

Directora

Arantza Illarramendi Echave

Resumen

Abstract of the project at the beginning of the document.

Agradecimientos

Agradecimientos del proyecto.

Índice general

Índice de figuras

Índice de tablas

1. CAPÍTULO

Introducción

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Integer id sem orci. Fusce vel consectetur tortor. Donec a aliquam magna, eu posuere ligula. Donec ut ligula quam. Integer non sem vitae odio molestie efficitur eu et erat. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Sed elit lectus, blandit pretium ullamcorper quis, iaculis ac ipsum. Sed bibendum imperdiet eros, vel blandit eros auctor id. Integer commodo in ipsum sed tincidunt. Aenean mattis mi eget ex mollis, vitae aliquet ex maximus. Nullam dictum risus sit amet ornare tincidunt. Pellentesque venenatis massa elit, in pellentesque velit pulvinar at. Nunc id tortor ornare, pretium ex vitae, mollis mauris.

Pellentesque consectetur pretium arcu et tempor. Suspendisse id turpis quis dolor aliquam accumsan vel a arcu. Nulla bibendum risus ut elit ultrices, sit amet placerat ante euismod. Aliquam erat volutpat. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos. Morbi non sodales elit, ac posuere felis. Suspendisse condimentum malesuada rhoncus. Sed aliquet ex a tellus hendrerit suscipit. Nulla suscipit erat nec maximus hendrerit. Cras pulvinar, metus et congue semper, enim tortor posuere turpis, ut commodo neque dui et urna. Quisque interdum sem nibh, at blandit diam ornare lacinia. Morbi ultrices, tortor ut finibus porttitor, augue metus congue ligula, id hendrerit nunc dui nec nibh. Duis euismod velit lorem, in gravida lectus vulputate at. Ut scelerisque volutpat urna et consequat.

Nunc pretium vel sem sit amet porta. In sollicitudin quam quis est fermentum gravida. Ut fringilla maximus nunc sagittis scelerisque. Proin varius pretium leo, a pharetra felis

iaculis vitae. Praesent cursus ex lorem, sit amet tincidunt ex aliquam nec. Ut dignissim cursus justo nec porttitor. Pellentesque maximus massa accumsan nibh lacinia volutpat. Nunc scelerisque dolor id tortor eleifend accumsan. Mauris faucibus lacus non metus rutrum feugiat. Vestibulum et ex nec sem volutpat gravida. Integer consectetur suscipit arcu gravida imperdiet. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Quisque iaculis fringilla enim, ut dictum nisl ullamcorper ac.

2. CAPÍTULO

Contexto del proyecto

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Integer id sem orci. Fusce vel consectetur tortor. Donec a aliquam magna, eu posuere ligula. Donec ut ligula quam. Integer non sem vitae odio molestie efficitur eu et erat. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Sed elit lectus, blandit pretium ullamcorper quis, iaculis ac ipsum. Sed bibendum imperdiet eros, vel blandit eros auctor id. Integer commodo in ipsum sed tincidunt. Aenean mattis mi eget ex mollis, vitae aliquet ex maximus. Nullam dictum risus sit amet ornare tincidunt. Pellentesque venenatis massa elit, in pellentesque velit pulvinar at. Nunc id tortor ornare, pretium ex vitae, mollis mauris.

Pellentesque consectetur pretium arcu et tempor. Suspendisse id turpis quis dolor aliquam accumsan vel a arcu. Nulla bibendum risus ut elit ultrices, sit amet placerat ante euismod. Aliquam erat volutpat. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos. Morbi non sodales elit, ac posuere felis. Suspendisse condimentum malesuada rhoncus. Sed aliquet ex a tellus hendrerit suscipit. Nulla suscipit erat nec maximus hendrerit. Cras pulvinar, metus et congue semper, enim tortor posuere turpis, ut commodo neque dui et urna. Quisque interdum sem nibh, at blandit diam ornare lacinia. Morbi ultrices, tortor ut finibus porttitor, augue metus congue ligula, id hendrerit nunc dui nec nibh. Duis euismod velit lorem, in gravida lectus vulputate at. Ut scelerisque volutpat urna et consequat.

Nunc pretium vel sem sit amet porta. In sollicitudin quam quis est fermentum gravida. Ut fringilla maximus nunc sagittis scelerisque. Proin varius pretium leo, a pharetra felis

iaculis vitae. Praesent cursus ex lorem, sit amet tincidunt ex aliquam nec. Ut dignissim cursus justo nec porttitor. Pellentesque maximus massa accumsan nibh lacinia volutpat. Nunc scelerisque dolor id tortor eleifend accumsan. Mauris faucibus lacus non metus rutrum feugiat. Vestibulum et ex nec sem volutpat gravida. Integer consectetur suscipit arcu gravida imperdiet. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Quisque iaculis fringilla enim, ut dictum nisl ullamcorper ac.

3. CAPÍTULO

Objetivos y planificación del proyecto

3.1. Objetivos

El objetivo del proyecto consiste en **diseñar y crear** una base de datos, utilizando distintos Sistemas de Gestión de Bases de Datos, que permitan almacenar las facturas emitidas por las distintas empresas y locales del País Vasco.

3.2. Alcance

3.2.1. Objetivos

3.2.2. Exclusiones

3.2.3. Estructura de Desglose de Trabajo (E.D.T)

Una vez se han definido los objetivos y las exclusiones del proyecto se ha desarrollado el diagrama EDT (figura 3.1), el cual contiene las diferentes fases del trabajo junto con los paquetes de trabajo a realizar en cada uno de ellos.

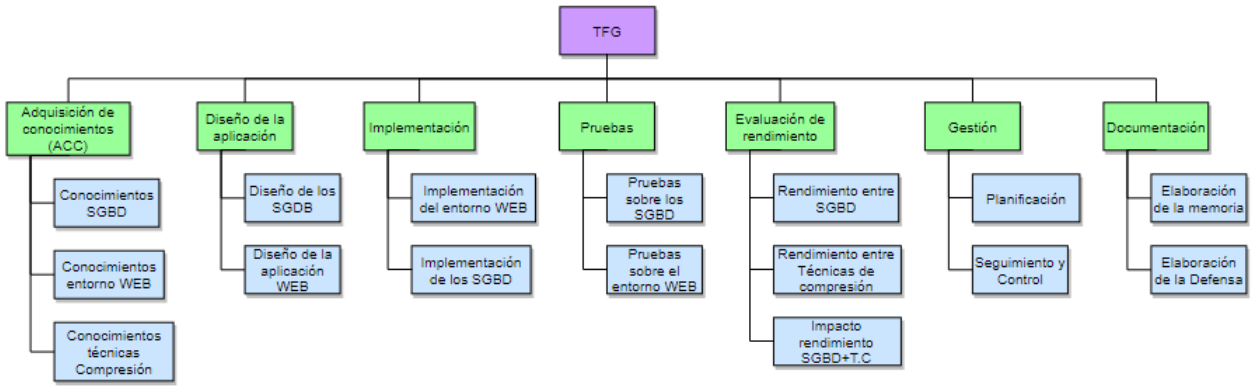

Figura 3.1: Diagrama EDT con los paquetes de trabajo relativos al proyecto

A continuación se presentan todos los paquetes de trabajo, estructurados por fases, junto con las tareas a realizar en cada uno de los paquetes.

Fase de Adquisición de Conocimientos

El fase de trabajo de ***Adquisición de Conocimientos*** se agrupan todas las tareas y actividades relacionadas con la adquisición de competencias necesarias para el correcto desarrollo del proyecto.

Conocimientos sobre los SGBD (CCSGBD)

Este paquete agrupa la adquisición de conocimientos sobre los tres SGBD que se quieren utilizar. Por tanto, las tareas que encontramos en este paquete son:

- **CCSGBD.1:** Adquisición de conocimientos sobre el SGBD No-SQL Neo4J.
- **CCSGBD.2:** Adquisición de conocimientos sobre el SGBD No-SQL MongoDB.
- **CCSGBD.3:** Adquisición de conocimientos sobre el SGBD No-SQL Cassandra.

Conocimientos sobre el entorno WEB (CCW)

- **CCW.1:** Adquisición de conocimientos sobre el entorno web (NodeJS) que permitirá el uso de los SGBD.
- **CCW.2:** Elección del servicio web en el que se alojará el entorno que permita hacer las pruebas.

Conocimientos sobre las técnicas de compresión (CCTC)

En este paquete de trabajo se agrupan las tareas relacionadas con la adquisición de conocimientos sobre las distintas técnicas de compresión sin pérdidas que se utilizarán durante el desarrollo de las bases de datos.

- **CCTC.1:** Adquisición de conocimientos sobre las técnicas de compresión sin pérdida que se utilizarán, *gzip, lzma, lzma2, lzss* y *brotli*.
- **CCTC.2:** Adquisición de conocimientos sobre técnicas de compresión con pérdida.
- **CCTC.3:** Comprender las técnicas de compresión que utilizan cada uno de los SGBD.

Diseño de la aplicación

En la fase de trabajo de ***Diseño de la aplicación*** se recogen todas las tareas relacionadas con el diseño de los distintos esquemas en los diferentes SGBD, así como el diseño de la aplicación WEB.

Diseño de las BD (DABD)

- **DABD.1:** Diseño genérico del sistema. Se diseñara un esquema genérico, es decir, no pensado para ningún SGBD concreto, con el fin de tener una estructura base para los tres sistemas.
- **DABD.2:** Diseño del sistema para Neo4J. Se realizará la adaptación del diseño genérico para poder funcionar en el SGBD Neo4J.
- **DABD.3:** Diseño del sistema para MongoDB. Se realizará la adaptación del diseño genérico para poder funcionar en el SGBD MongoDB.
- **DABD.4:** Diseño del sistema para Cassandra. Se realizará la adaptación del diseño genérico para poder funcionar en el SGBD Cassandra.

Diseño de la aplicación WEB (DAW)

- **DAW.1:** Preparación el dominio en el que se almacenará la aplicación web.
- **DAW.2:** Análisis de requisitos de la aplicación web.
- **DAW.3:** Diseño de la aplicación web utilizando NodeJS.

Implementación

Durante la fase de trabajo de ***Implementación*** se realizarán todos los paquetes de trabajo relacionados con la implementación de los sistemas.

Implementación de las BD (IBD)

- **IBD.1:** En este paquete de trabajo se realizará la implementación del esquema diseñado para Neo4J.
- **IBD.2:** Durante el desarrollo de este paquete de trabajo se realizará la implementación del esquema diseñado para MongoDB.
- **IBD.3:** Durante el transcurso de este paquete de trabajo se realizará la implementación del esquema diseñado para el SGBD Cassandra.

Implementación del entorno web (IWEB)

- **IWEB.1:** Este paquete de trabajo engloba la implementación del entorno web.
- **IWEB.2:** Durante este paquete de trabajo se realizarán las tareas relacionadas con el alojamiento del entorno implementado en el servicio de hosting escogido.

Pruebas

Durante la fase de ***Pruebas*** se realizarán todas las pruebas necesarias a los SGBD y al entorno web, de manera que se pueda asegurar un correcto funcionamiento de todos los sistemas desarrollados.

Pruebas sobre las BD (PRBD)

- **PRBD.1:** Tarea enfocada al desarrollo de las pruebas para cada uno de los SGBD del sistema.
- **PRBD.2:** Durante esta tarea se realizarán las pruebas necesarias para asegurar el correcto funcionamiento de la BD implementada en Neo4J.
- **PRBD.3:** En esta tarea se desarrollarán y ejecutarán las pruebas que permitan consolidar el correcto funcionamiento de la BD desarrollada en MongoDB.

- **PRBD.4:** Mediante esta tarea se quiere afianzar el correcto funcionamiento de la BD implementada en Cassandra.
- **PRBD.5:** Correcciones, si fuesen necesarias, de las implementaciones de las BD.

Pruebas sobre el entorno WEB (PRWEB)

- **PRWEB.1:** Desarrollo de las pruebas que puedan garantizar el correcto funcionamiento del entorno web.
- **PRWEB.2:** Ejecución de las pruebas desarrolladas sobre el entorno Web.
- **PRWEB.3:** Correcciones, si fuesen necesarias, a realizar el entorno debido a comportamientos erróneos detectados durante las pruebas.

Fase de evaluación de rendimiento

Durante la fase de ***Evaluación de Rendimiento*** se recogen todas las tareas relacionadas con la comparativa de rendimiento entre las distintas BD implementadas, las técnicas de compresión y el impacto de cada una de las técnicas en los distintos BD.

Evaluación de rendimiento entre bases de datos (ERBD)

- **ERBD.1:** Planificación de las consultas que evaluarán el rendimiento entre las distintas BD.
- **ERBD.2:** Ejecución y anotación de resultados de las pruebas.
- **ERBD.3:** Elaboración de un documento que recoja el resultado de cada una de las pruebas ejecutas en los distintos BD.

Evaluación de rendimiento entre las técnicas de compresión (ERTC)

- **ERTC.1:** Planificación de las pruebas que evaluarán el rendimiento entre todas las técnicas de compresión abordadas.
- **ERTC.2:** Ejecución y anotación de resultados de las pruebas ejecutas.
- **ERTC.3:** Elaboración de un documenta que recoja los resultados obtenidos en las pruebas para poder realizar un análisis exhaustivo de las mismas.

Evaluación conjunta del rendimiento entre las BD y las Técnicas de Compresión (ERC)

- **ERC.1:** Planificación de las pruebas que evaluarán el rendimiento conjunto de las BD junto con las técnicas de compresión.
- **ERC.2:** Ejecución de las pruebas con las distintas combinaciones de BD + TC[1].
- **ERC.3:** Elaboración de un documento que recoja las pruebas realizadas y que permita un posterior análisis de los resultados.

Fase de Gestión

La fase de ***Gestión*** recoge todas las tareas relacionadas con la planificación y gestión del proyecto.

Planificación (P)

- **P.1:** Análisis inicial de requisitos del proyecto.
- **P.2:** Elaboración de la planificación del proyecto.
- **P.3:** Actualización, si fuera necesaria, de la planificación.
- **P.4:** Lectura detallada de la normativa de elaboración de los Trabajos Fin de Grado, así como de los criterios de evaluación, con el objetivo de desarrollar un TFG de calidad.

Seguimiento y Control (SyC)

- **SyC.1:** Reuniones periódicas con la directora durante el transcurso del proyecto.
- **SyC.2:** Elaboración de un documento que recoja la dedicación realizada cada día así como el desarrollo o problemas surgidos en cada una de las tareas.
- **SyC.3:** Seguimiento continuo para evitar riesgos o posibles problemas no contemplados durante la fase de planificación.
- **SyC.4:** Mantenimiento continuo del entorno web.

[1] Técnica de Compresión

Fase de elaboración de la documentación

Elaboración de la memoria (EM)

- **EM.1:** Preparación del entorno en el que se redactará la memoria.
- **EM.2:** Redacción de la memoria.
- **EM.3:** Revisión periódica de la memoria con el objetivo de detectar errores no contemplados durante la redacción.

Elaboración y Preparación de la defensa (EDEF)

- **EDEF.1:** Análisis de los contenidos a incluir en la presentación.
- **EDEF.2:** Preparación del entorno de elaboración de la presentación.
- **EDEF.3:** Elaboración de la presentación.
- **EDEF.4:** Elaboración de un documento que sirva de apoyo durante la presentación.
- **EDEF.5:** Práctica de la presentación a fin de mejorar la calidad de la misma.
- **EDEF.6:** Correcciones, si fuesen necesarias, de la presentación.

3.3. Diagrama de Dependencias entre tareas

3.4. Diagrama Gantt

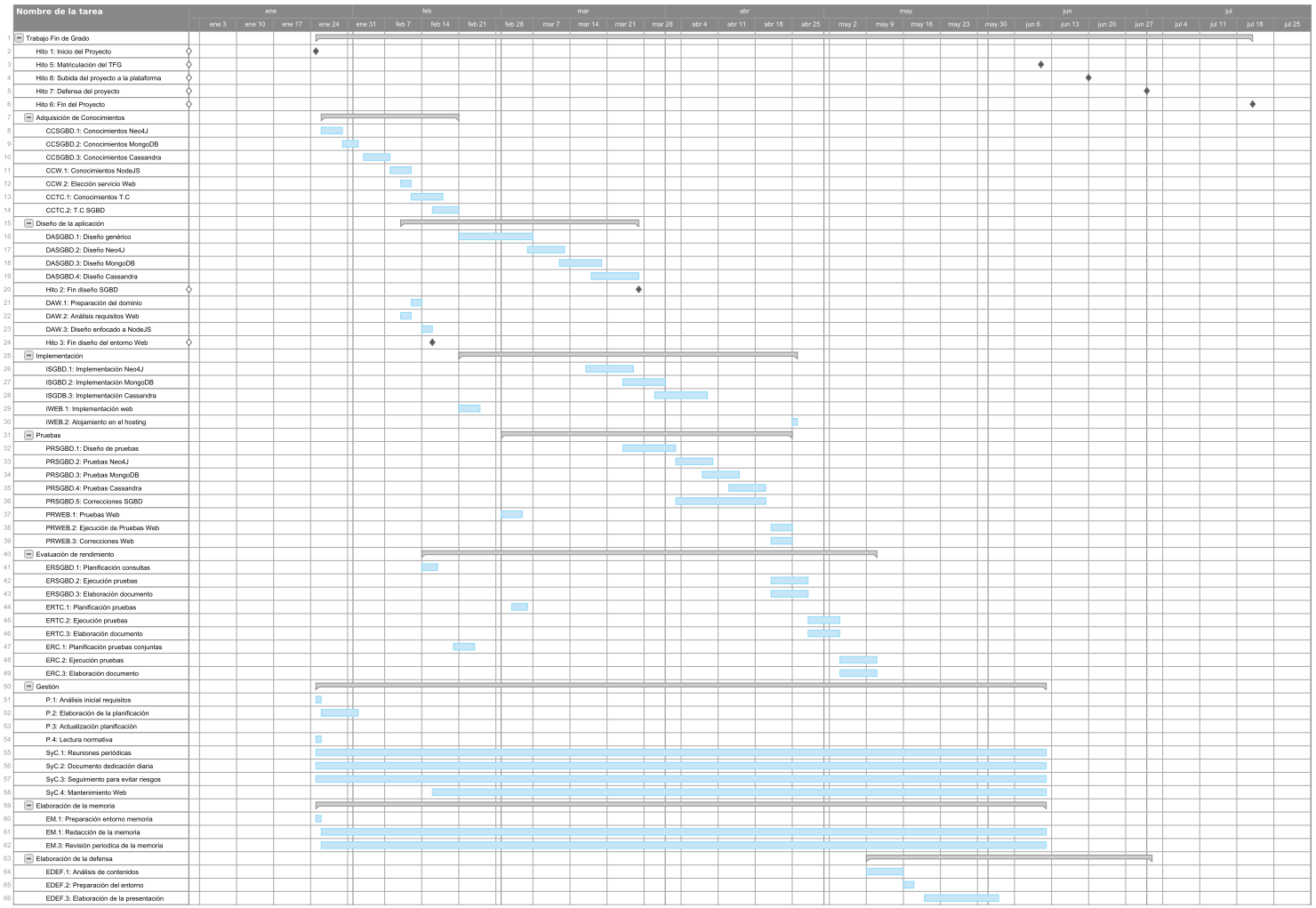

Figura 3.2: Caption

	Nombre de la tarea	ene				feb				mar					abr				may				jun				jul			
		ene 3	ene 10	ene 17	ene 24	ene 31	feb 7	feb 14	feb 21	feb 28	mar 7	mar 14	mar 21	mar 28	abr 4	abr 11	abr 18	abr 25	may 2	may 9	may 16	may 23	may 30	jun 6	jun 13	jun 20	jun 27	jul 4	jul 11	jul 18	jul 25
67	EDEF.4: Elaboración de documento de apoyo																													
68	EDEF.5: Práctica de la presentación																													
69	EDEF.6: Correcciones presentación																													

Figura 3.3: Caption

3.4.1. Hitos del Proyecto

En esta sección se describen los hitos a alcanzar durante el desarrollo del proyecto.

- **Hito 1:** Inicio del proyecto.
- **Hito 2:** Fin de diseño de los SGBD.
- **Hito 3:** Fin de diseño del entorno web.
- **Hito 4:** Fin de la implementación de los SGBD.
- **Hito 5:** Matriculación del proyecto en G.A.U.R. (11/06/2021)
- **Hito 6:** Fin del proyecto. (21/07/2021)
- **Hito 7:** Defensa del proyecto. (01/07/2021[2])
- **Hito 8:** Subida del proyecto a la plataforma. (20/06/2021)

3.5. Gestión de riesgos

En todo proyecto es necesario realizar un buen análisis y gestión de riesgos que permitan que el proyecto se ejecute sin ningún problema. Una buena mitigación de riesgos evitará costos extras en el proyecto.

Durante esta sección, se detallan los riesgos identificados que puedan afectar negativamente al desarrollo del proyecto, junto con las acciones a tomar en cada uno de ellos.

3.5.1. Problemas de compaginación con las asignaturas

Dado que las fechas de realización del TFG coinciden con el desarrollo del último cuatrimestre del cuarto curso del grado en Ingeniería Informática, gran parte del desarrollo del mismo deberá ser compaginado con las asignaturas a realizar durante ese cuatrimestre.

- **Probabilidad:** Baja

[2]Fecha estimada

- **Impacto:** Alto
- **Responsabilidad:** Propia
- **Consecuencias:** Dificultad de cumplimiento de los plazos establecidos para el proyecto.
- **Prevención:** Realizar una planificación en la que se tengan en cuenta los hitos y dedicaciones estimadas de las asignaturas.
- **Plan de mitigación:** Replanificación de las tareas para tratar de compaginar ambas actividades de la mejor manera posible.

3.5.2. Errores en la planificación

En todo proyecto con una planificación de gran envergadura existe el riesgo de que dicha planificación contenga errores o que no contenga aspectos importantes no contemplados al comienzo

- **Probabilidad:** Media
- **Impacto:** Alto
- **Responsabilidad:** Propia
- **Consecuencias:** Retraso en el cumplimiento de los plazos establecidos.
- **Prevención:** Realizar un amplio estudio previo para evaluar los posibles errores e imprevistos. Así mismo, se ampliarán los plazos lo máximo posible dejando margen para posibles errores.
- **Plan de mitigación:** Replanificación de los plazos establecidos intentando garantizar el menor impacto posible a largo plazo.

3.5.3. Problemas con la captura de requisitos

Desde el comienzo del desarrollo del proyecto, el grupo BDI no tenía información sobre las principales consultas que se realizarían sobre el sistema, por lo que, principalmente, el diseño del SGBD Cassandra se vería afectado.

- **Probabilidad:** Media
- **Impacto:** Alto
- **Responsabilidad:** Ajena
- **Consecuencias:** Retraso en el desarrollo de los SGBD, principalmente Cassandra.
- **Prevención:** Para evitar que este riesgo afecte drásticamente al desarrollo del proyecto, si no se ha obtenido información suficiente que permita un correcto desarrollo del SGBD antes del 1 de Abril, se descartará el desarrollo del SGBD Cassandra.
- **Plan de mitigación:**

3.5.4. Problemas con el aprendizaje de las tecnologías

Dado que muchas de las tecnologías abordadas durante el proyecto son desconocidas o no han sido utilizadas nunca, pueden darse problemas en el aprendizaje de las mismas.

- **Probabilidad:** Baja
- **Impacto:** Alto
- **Responsabilidad:** Propia
- **Consecuencias:** Retraso en el desarrollo de algunas de las tareas del proyecto.
- **Prevención:** Realizar una planificación en la que se establezca un tiempo considerable a la adquisición de conocimientos, asegurando de esta manera la correcta consolidación de los mismos.
- **Plan de mitigación:** Adelantar la realización de algunas de las tareas no dependientes de la herramienta o tecnología en cuestión.

3.5.5. Problemas personales

Durante el desarrollo del proyecto existe la posibilidad de que se den problemas personales que no se pueden prever.

- **Probabilidad:** Baja

- **Impacto:** Medio
- **Responsabilidad:** Propia
- **Consecuencias:** Retraso en el desarrollo y entrega de algunas de las tareas del proyecto.
- **Prevención:** Es imposible prever este tipo de problemas ya que habra variaciones en función del problema surgido.
- **Plan de mitigación:** Replanificación de las tareas con el objetivo de reducir las consecuencias a corto / medio plazo.

3.6. Herramientas y Tecnologías

3.6.1. Overleaf y LaTeX

Overleaf[3] es un editor de Latex *online*. Se trata de un editor gratuito que no necesita instalación y que permite editar, de forma rápida y sencilla, el lenguaje antes mencionado. En cuanto al ya mencionado Latex, es un lenguaje de composición de textos enfocado a crear documentos de alta calidad tipográfica. Tanto Overleaf como Latex, se han utilizado para redactar esta misma memoria.

3.6.2. Google Drive

Google Drive[4] es una de las más conocidas aplicaciones de almacenamiento en la nube. Permite acceder a tus datos desde cualquier dispositivo y en cualquier momento. Se ha utilizado para almacenar copias de respaldo de los distintos documentos y archivos utilizados en el proyecto.

[3] https://www.overleaf.com/
[4] https://www.google.es/drive/apps.html

3.6.3. GitHub y GitKraken

GitHub[5] es una plataforma *online* que implementa el control de versiones Git[6] y que permite almacenar cualquier tipo de proyectos, principalmente proyectos *software*. En cuanto a GitKraken[7], se trata de un cliente Git multiplataforma que permite realizar todas las funciones de Git sin necesidad de conocer exhaustivamente los comandos. Tanto GitHub como GitKraken, se han utilizado para llevar un control de versiones del proyecto.

3.6.4. Smartsheet

Smartsheet[8] es un servicio enfocado a la organización y gestión de trabajo de equipos de trabajo. Smartsheet se ha utilizado para desarrollar el Diagrama Gantt que recoge los periodos de realización de las tareas del proyecto.

3.6.5. Visual Paradigm

Visual Paradigm[9] es una herramienta que permite realizar múltiples diseños de modelado en UML. En concreto, se ha utilizado la versión *online* de esta plataforma[10] para crear el diagrama EDT que contiene los paquetes de trabajo del proyecto.

3.6.6. MongoDB

MongoDB[11] es un Sistema de Gestión de Base de Datos No-SQL orientado a documentos. Estos documentos son almacenados en formato JSON, por lo que le da una compatibilidad con multitud de lenguajes de forma nativa.

[5] https://github.com/
[6] https://git-scm.com/
[7] https://www.gitkraken.com/
[8] https://es.smartsheet.com/
[9] https://www.visual-paradigm.com/
[10] https://online.visual-paradigm.com/es/
[11] https://mongodb.com/

3.6.7. Cassandra

Cassandra [12] es un Sistema de Gestión de Base de Datos No-SQL orientado a tablas. Desarrollado por Apache en 2008 permite manejar grandes volúmenes de datos a través de varios servidores evitando de esta manera la existencia de un único punto de fallo.

3.6.8. Neo4J

Neo4J[13] es un Sistema de Gestión de Bases de Datos No-SQL orientada a grafos. El objetivo de ser un SGBD orientado a grafos es darle igual importancia a los datos como a las relaciones entre ellos.

3.6.9. Visual Studio Code

Visual Studio Code [14] es un editor de código abierto desarrollado por Microsoft[15]. Incluye características como finalización inteligente de código, *debugging* o resaltado de sintaxis. Estás características y otras muchas más le han hecho convertirse en uno de los editores de código más utilizado por los desarrolladores.

3.6.10. Node.JS

Node.JS [16] es un entorno de ejecución para la capa servidor, de código abierto, basado en el lenguaje de programación JavaScript. Este entorno se ha utilizado para crear todo el sistema que simulará el envío y recepción de facturas a las Bases de Datos implementadas, además también se utilizará para crear el entorno web que simulará el cliente final de la aplicación.

[12] https://cassandra.apache.org/
[13] https://neo4j.com/
[14] https://code.visualstudio.com/
[15] https://www.microsoft.com/
[16] https://nodejs.org/en/

3.6.11. PostMan

PostMan [17] es una plataforma enfocada para el desarrollo de APIs. Junto con Node.JS, se ha utilizado para realizar peticiones de testeo que permitan probar el correcto funcionamiento del sistema.

[17] https://www.postman.com/

4. CAPÍTULO

Descripción de las técnicas de compresión

La RAE[1] [ASALE and RAE, 2021], define comprimir como la acción de oprimir, apretar, estrechar o reducir a menor volumen algo. Así, los algoritmos de compresión, se encargan de reducir el volumen de datos que ocupa un fichero o archivo.

A la hora de recuperar el formato original existen dos técnicas, las técnicas sin pérdida, que como su nombre indica permite recuperar el archivo en el estado que tenía originalmente, y las técnicas con pérdida, que al contrario que las anteriores, juegan con la "pérdida de datos" para restablecer su estado original.

Para poder medir la "efectividad" de los algoritmos de compresión, se utiliza el ratio de compresión, que se define como $1 - \frac{B_0}{B_1}$, donde B_0 representa el número de total de bits del archivo después de la compresión y B_1 representa el número total de bits del archivo antes de la compresión. Por tanto, cuanto más cercano a 1 sea el ratio de compresión mejor será el algoritmo, aunque más adelante veremos que hay más factores a tener en cuenta, como puede ser la velocidad de compresión.

[1] Real Academia Española

4.1. Técnicas de compresión sin pérdida

Los algoritmos de compresión sin pérdida (*lossless algorithms*) son capaces de reconstruir el formato original a partir del mensaje comprimido [Hosseini, 2012]. Por ello, estos algoritmos, se utilizan principalmente en tareas relacionadas con el almacenamiento de textos o programas.

A continuación, se describen 5 algoritmos de compresión sin pérdida.[Gupta et al., 2017] Estos algoritmos son los que el grupo BDI concluyó que se ajustaban mejor a la aplicación a desarrollar.

4.1.1. GZip

El compresor GZip [loup Gailly and Adler, 2018] es el compresor utilizado y desarrollado por GNU[2]. Apareció por primera vez en 1993 con el objetivo de sustituir el anterior programa de compresión de GNU, *compress*, y sigue en uso, siendo su última versión estable, la 1.10, que se liberó en 2018.

GZip es capaz de reducir el tamaño de los archivos utilizando el algoritmo Lempel-Ziv (LZ77). Este algoritmo es la base de todos los algoritmos que se van a describir en los siguientes apartados, así como de prácticamente todos los algoritmos basados en el uso de un diccionario.

La idea básica del algoritmo es ir iterando sobre el texto de entrada y reemplazando los datos repetidos por la referencia a una única copia de esos datos los cuales se han tratado previamente. Cada una de estas coincidencias se codifica como la siguiente tripleta *(offset, length, character)*, donde el *offset* representa el número de pasos hacia atrás, es decir, hacia el flujo de datos sin comprimir, que se deben realizar para encontrar el comienzo de la cadena que representa. *Length* simplemente representa la longitud de la cadena. Y *character* representa el siguiente carácter después de la cadena.

Por ejemplo, si se quiere codificar la siguiente cadena.[Budhrani, 2019]

```
1   A B A B C B A B A B A A
```

El algoritmo, avanzará de la siguiente manera. Primero, leerá el carácter “A”, al no estar registrado en el diccionario se guardará la tripleta (0,0,A), ya que para obtener el primer

[2] https://gnu.org

carácter se tienen que ir 0 pasos hacia atrás con un *string* de longitud 0 y encontrando el carácter “A”. El próximo carácter a tratar se representa entre ‘[ ]’.

```
A [B] A B C B A B A B A A
(0,0,A)
```

A continuación, se leerá el carácter “B”, como su situación es la misma que en el carácter anterior la codificación es similar, (0,0,B).

```
A B [A] B C B A B A B A A
(0,0,A) (0,0,B)
```

Seguidamente, se leerá el carácter “A”, el cual ya esta en el diccionario, el primer carácter del flujo a representar, por ello, se leerá el conjunto “AB”, el cual también esta en el diccionario, los dos primeros caracteres del flujo. Sin embargo, no es ese el caso del conjunto “ABC”, por ello, para poder formar este conjunto, y tomando como referencia el carácter “A”, que era el carácter que tocaba tratar, se retrocederá dos posiciones, donde esta la representación más cercana, hasta la primera “A”, se formará un *string* de longitud 2, “AB” y se añadirá la “C” al final. De esta manera, se formaría la tripleta (2,2,C).

```
A B A B C [B] A B A B A A
(0,0,A) (0,0,B) (2,2,C)
```

A continuación se tratará la “B”, la cual ya se ha tratado con anterioridad, lo mismo pasa con “BA” y “BAB”, pero no con “BABA”, así que para formar este conjunto se retrocederá, partiendo desde el carácter “B” que nos tocaba tratar en este paso, 4 posiciones hacía atrás, formar un *string* de longitud 3, “BAB", y añadirle la “A” al final. Por tanto, la tripleta que se forma será (4,3,A).

```
A B A B C B A B A [B] A A
(0,0,A) (0,0,B) (2,2,C) (4,3,A)
```

Es el turno de la última codificación, el carácter “B” ya se ha tratado, y el conjunto “BA” también, siendo la representación más cercana justo 2 caracteres hacia atrás. El conjunto “BAA” no esta representado, por lo que, partiendo desde “B”, se retrocederá 2 pasos y se formará, con longitud 2, el *string* “BA” y se añadirá el carácter “A” al final, formando la tripleta (2,2,A). Es decir, la codificación del flujo de caracteres inicial sería.

```
(0,0,A) (0,0,B) (2,2,C) (4,3,A) (2,2,A)
```

Teniendo la codificación, la descompresión es “sencilla”. Tomando como referencia la primera tripleta, (0,0,A), la cual nos indica que se deben dar 0 pasos hacia atrás, coger el conjunto de longitud 0, es decir, vacío, y añadir el carácter A.

```
A _
(0,0,B) (2,2,C) (4,3,A) (2,2,A)
```

Siendo el “_” la siguiente posición a tratar, se cogerá la siguiente tripleta (0,0,B). Al igual que en la tripleta anterior, simplemente será añadir el carácter que se indica en la posición correspondiente.

```
A B _
(2,2,C) (4,3,A) (2,2,A)
```

Se cogerá la siguiente tripleta, (2,2,C), la cual representa “El *string* de longitud 2 que comienza 2 posiciones hacia la izquierda y añadiendo el carácter C al final”.

```
A B (A B C) _
(4,3,A) (2,2,A)
```

La siguiente tripleta (4,3,A), nos indica que se debe desplazar 4 posiciones hacia la izquierda, leer el *string* de longitud 3, y añadirle una “A” al final. Por tanto, el resultado sería el siguiente.

```
A B A B C (B A B A) _
(2,2,A)
```

Por último, la siguiente tripleta, (2,2,A), indica un desplazamiento de 4 posiciones hacia la izquierda, lectura de dos caracteres (“BA”), y añadirle una “A” al final. Por tanto, el resultado final de la descompresión, que es igual al flujo original, es.

```
A B A B C B A B A (B A A)
```

4.1.2. Lzma

LZMA[3] es uno de los algoritmos de compresión que utiliza el conocido programa de (des)compresión de datos 7-zip[4]. LZMA es uno de los muchos algoritmos que usan un diccionario basados en el algoritmo LZ77 que implementa GZip. Las principales diferencias entre ambos algoritmos son, el superior radio de compresión y la implementación de un tamaño variable de diccionario, siendo el tamaño máximo hasta 4GB. [Wikipedia, 2021]

A pesar de estar basado en el algoritmo LZ77, LZMA no es un algoritmo cuyo diccionario se basa únicamente en la agrupación de bytes, sino que especifica bits que indican el contexto en el que se encuentran esos bytes. De esta manera, se obtiene un mejor rendimiento en la compresión, ya que los bytes no están todos juntos sin mucha relación entre sí.

Los algoritmos LZMA y LZMA2, posteriormente descritos, no tienen una documentación clara sobre su funcionamiento, es decir, la mayoría de las implementaciones de estos algoritmos se han deducido a partir de otras implementaciones, siendo la utilizada por 7-zip la original.

El principal problema con el que tiene que lidiar este algoritmo es también una de sus mayores virtudes, el tamaño de diccionario. Un tamaño de diccionario tan grande, hasta 4 GB, permite comprimir gran cantidad de archivos, sin embargo, la búsqueda dentro de este diccionario puede ser algo tediosa. Al ser de un tamaño tan grande una búsqueda secuencial sería muy ineficiente, ya que volvería el algoritmo muy lento y poco práctico para la mayoría de usos. Por ello, LZMA utiliza su propio codificador, el cual puede decidir libremente con que *string* emparejar el flujo leído o simplemente ignorarlo.

4.1.3. Lzma2

LZMA2 es simplemente, una versión mejorada del algoritmo Lzma antes descrito. El creador de ambos algoritmos de compresión, Igor Pavlov, describe las siguientes virtudes principales de este algoritmo frente a LZMA [Pavlov, 2011].

- Mejora en la velocidad de compresión de archivos de más de 256 MB al utilizar 4 hilos de CPU o más.
- Mejora en la compresión de datos previamente comprimidos.

[3] Lempel-Ziv-Markov chain algorithm
[4] https://www.7-zip.org/

- El algoritmo fue creado para funcionar con el formato XZ, por lo que incluye modificaciones que mejoran la compresión de datos en ese formato.

Cabe mencionar que estas mejoras del algoritmo LZMA2 se obtienen utilizando mayor cantidad de memoria.

4.1.4. Lzss

LZSS[5] es un algoritmo de compresión sin pérdida creado en 1982. Este algoritmo, como los dos descritos anteriormente, está basado en el algoritmo LZ77, por tanto, es también un algoritmo basado en el uso de un diccionario. [Dipperstein, 2019]

Para poder comprimir los archivos o cadenas de caracteres, LZSS, trata de reemplazar un *string* por la referencia en el diccionario de la representación de ese *string*, similar a LZ77. Sin embargo, una de las diferencias frente a este algoritmo es el "tamaño" de esta referencia, el cual siempre será inferior al tamaño de la cadena a representar, en el algoritmo LZ77, esa condición no siempre se daba.

Por tanto, las principales diferencias entre LZSS y LZ77 se dan en la forma de almacenamiento del diccionario que ambos algoritmos utilizan para comprimir y descomprimir archivos, siendo la característica más significativa, de este primer algoritmo, la forma en la que se guardan los caracteres individuales, es decir, los *string* de un único elemento. Si se recuerda, el algoritmo LZ77, cuando tenía que guardar una cadena de un solo elemento, almacenaba, **(0,0,a)**, el algoritmo LZSS guardaría únicamente el carácter "a", de manera que se ahorrase el espacio ocupado por el *offset* y el *length*.

Así mismo, otras mejoras que ofrece este algoritmo respecto a su predecesor son, la adición de un bit de control que permita determinar si la entrada del diccionario esta codificada o no, es decir, si se almacenó en el diccionario simplemente un carácter o si se almaceno en texto plano, cuyo segundo caso no tendría un coste computacional.

4.1.5. Brotli

Brotli es el algoritmo más moderno de los que se describen en este documento. Fue desarrollado en 2013 por Jyrki Alakuijala y Zoltán Szabadka y publicado en el estándar

[5]Lempel Ziv Storer Szymanski

RFC[6] de la IETF[7]en 2016 [Alakuijala and Szabadka, 2016].

Brotli, es un algoritmo de compresión sin pérdida desarrollado para uso genérico y que está basado en el algoritmo LZ77 y la codificación Huffman y promete una eficiencia de (des)compresión equiparable a la de los algoritmos antes vistos. Su gran eficiencia sobre archivos relacionados con la WEB (HTML[8], CSS[9]) lo hacen perfecto para usarlo con navegadores como Mozilla Firefox[Fir, 2016] o Google Chrome[Acc, 2016].

La codificación de *Huffman* se basa en la asignación de códigos a los caracteres de entrada basados en la frecuencia de aparición de los caracteres anteriores. Cuanto más frecuente sea el carácter más pequeño será el código que se le asigne a ese carácter. [GeeksforGeeks, 2012] Por tanto, el objetivo de este algoritmo es, a partir de la tabla de relaciones carácter-repeticiones, crear un árbol estructurado por el número de repeticiones de los caracteres, los más frecuentes serán hojas cerca de la raíz y los menos frecuentes serán las hojas de mayor profundidad.

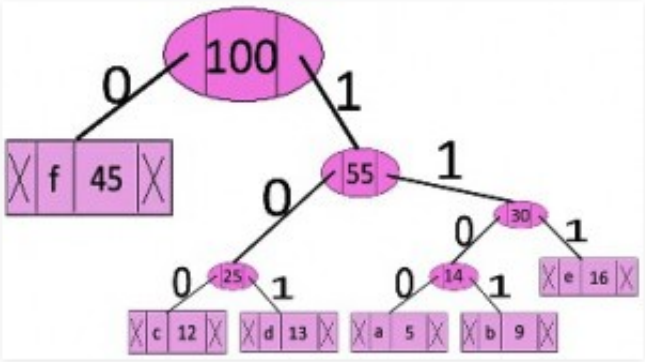

Figura 4.1: Árbol resultante de la codificación Huffman

[6]Request For Comments: https://www.ietf.org/standards/rfcs/
[7]Internet Engineering Task Force: https://www.ietf.org
[8]HyperText markup Language
[9]Cascading Style Sheet

Carácter	Codificación
f	0
c	100
d	101
a	1100
b	1101
e	111

Tabla 4.1: Codificación *Huffman* de los caracteres en base al árbol obtenido

En lo que al funcionamiento del algoritmo se refiere, los datos comprimidos se componen de una cabecera y una serie de meta-bloques. Cada uno de estos meta-bloques se descompone en una secuencia de 0 a 16MB sin comprimir. Los datos finales, sin comprimir, son la concatenación de las secuencias sin comprimir de cada meta-bloque. La cabecera, contiene los datos de la parte del *sliding window* utilizado.

Volviendo a los meta-bloques, cada uno de ellos esta comprimido mediante una combinación entre el algoritmo LZ77 y la codificación Huffman. Al resultado que se obtiene de la ultima codificación se le denomina "prefijo" y representa unívocamente a cada uno de los meta-bloques, semejante a la representación de clave primaria de una tabla de una BD. En cuanto al uso del algoritmo LZ77, se utiliza para codificar los contenidos de los meta-bloques, de manera que las cadenas sean referencias a otros meta-bloques.

4.2. Algoritmos de compresión con pérdida

En esta sección se describen algunos de los algoritmos de compresión con pérdida (*lossy algorithms*) más utilizados. Estos algoritmos, generalmente utilizados en tareas relacionadas con la compresión de vídeos o imágenes, son capaces de obtener una aproximación parcial del archivo original.

Figura 4.2: Ejemplo de compresión de una imagen con pérdida

4.2.1. JPEG

JPEG (Joint Photographic Experts Group)[Wallace, 1992] es uno de los algoritmos más conocidos de compresión de imágenes, aunque muchos lo conocen por la extensión de archivo ".jpeg". Fue creado en 1992 por el grupo que lleva su nombre y se basa en la Transformada del Coseno Discreto (DCT) propuesta por Nasir Ahmed en 1972. El funcionamiento del algoritmo es el siguiente.

En primer lugar, se divide la imagen en bloques de 8x8 píxeles. Puesto que cada píxel está compuesto por los tres colores del modelo RGB[10], el siguiente proceso se aplica para cada uno de las 64 representaciones de color del RGB dentro del bloque de 8x8.

Una vez obtenido el bloque de intensidades del color rojo por ejemplo, se le aplica la Transformada del Coseno Discreto de dos dimensiones(TCD), antes mencionada, de manera que se obtengan los coeficientes que representan el bloque de 8x8 anterior. Con la tabla de coeficientes se pasa al proceso denominado "Cuantización", aquí, la tabla de coeficientes se cuantiza uniformemente con una tabla especificada por el usuario o aplicación. A esta tabla se la denomina "Tabla de Cuantización" y contiene 64 valores (8x8) entre 1 y 255. En función de los valores contenidos en esta tabla la imagen resultado tendrá una compresión diferente (Cuanto mayores los números mayor compresión y menor calidad.)

[10]Red Green Blue

Figura 4.3: Niveles de compresión de JPEG

4.2.2. Algoritmos de Compresión de Texto

En lo que ha compresión de texto se refiere, en general, los esfuerzos se han centrado en utilizar técnicas de compresión sin pérdidas. Esto se debe a que generalmente, el texto suele ser parte de áreas críticas del sistema. Sin embargo, ¿Que pasa cuando el texto no pertenece a un área crítica?, ¿Es rentable utilizar algoritmos de compresión sin pérdida, los cuales pueden tener un mayor coste computacional, para estos campos? A continuación, se presentan tres sencillas técnicas capaces de comprimir un texto y cuya descompresión conlleva a una aproximación del texto original. [Palaniappan and Latifi, 2007]

La técnica **Letter Mapping (LMP)** consiste en el reemplazamiento de caracteres en todo el texto. El objetivo es reemplazar los caracteres con menos frecuencia de aparición, por otros con mayor frecuencia. Por ejemplo, en el alfabeto español, las consonantes con mayor porcentaje de aparición son *S, R,N, D* y *L*, mientras que las menos frecuentes son *J, Ñ, X, K* y *W*. La finalidad, por tanto, no es otra que sustituir los caracteres menos frecuentes por los más frecuentes, de manera que se reduzca el tamaño del alfabeto, en este caso por 5. De esta manera, si se aplica este método junto con la codificación de *Huffman*, ver 4.1.5, los caracteres que inicialmente eran menos probables, estarán ahora codificados con bits más pequeños. Tras descomprimir el texto, nos encontraremos con un gran número de palabras mal escritas, nada que no pueda solucionar un corrector ortográfico, aunque algunas palabras no serán las que se tenían originalmente.

La técnica **Dropped Vowels (DOV)** consiste en reemplazar las apariciones de todas las vocales que encontramos en un texto. En el alfabeto español las vocales son de las letras más frecuentes que nos encontramos, por lo que si eliminásemos las vocales se conseguiría una

gran compresión, aunque el problema vendría al tratar de recuperar el texto original. Una posible solución es sustituir todas las vocales por una concreta, es decir, sustituir las vocales *e, i , o, u* por la vocal *a*, reduciendo de esta manera el tamaño de compresión y pudiendo recuperar la gran mayoría de palabras del texto utilizando un corrector ortográfico.

La última técnica **Replacement of Characters (ROC)**, consiste en sustituir grupos de caracteres por un único carácter que denote el mismo sonido o significado, exactamente igual que la taquigrafía. De esta manera, el texto descomprimido presentaría una gran variedad de errores, pero quedaría legible por cualquier persona.

4.3. Algoritmos utilizados por los SGBD

Conocer los algoritmos de compresión es importante, sin embargo, también hay que tener en cuenta que la gran mayoría de SGBD existentes tienen implementados algoritmos de compresión, pudiendo elegir el algoritmo a utilizar desde los parámetros de configuración de los SGBD.

En esta sección se explicarán los algoritmos que implementan los SGBD tratados en este proyecto (MongoDB y Cassandra).

4.3.1. MongoDB

El SGBD **MongoDB** utiliza tres librerías de compresión de datos, *Snappy*[11], *zlib*[12] y *zstd*[13], siendo la primera la utilizada por defecto.

Snappy

Snappy es un algoritmo de (des)compresión basado en el algoritmo LZ77 desarrollado por Google y liberado en 2011.[Google, 2021]. La idea de este algoritmo es conseguir una compresión aceptable en el menor tiempo posible. Es más, comparado con *zlib*, otro de los algoritmos que utiliza MongoDB, es mucho más rápido a la hora de comprimir pero con resultados de compresión más grandes.

[11] https://google.github.io/snappy/
[12] http://www.zlib.net/
[13] https://facebook.github.io/zstd/

Snappy, es también utilizado en otros SGBD, como puede ser Cassandra, el cual se verá a continuación, o Couchbase[14], un SGBD No-SQL enfocado a documentos y que utiliza formato JSON para almacenamiento.

ZLib

En cuanto a **zlib**, es una variación del algoritmo de (des)compresión *deflate* desarrollado por Phil Katz y basado en una combinación del algoritmo LZ77 y la codificación Huffman.

Hay una gran cantidad de aplicaciones conocidas que utilizan *zlib* como (des)compresor de datos[Gailly and Adler, 2002], entre ellas, el lenguaje de programación *Python* o la aplicación de comunicación *openSSH*.

En cuanto a su funcionamiento, básicamente es el mismo que el del algoritmo *deflate* antes mencionado y posteriormente explicado. En términos generales, el algoritmo divide el flujo de datos de entrada en bloques, y cada uno de estos bloques se comprime individualmente. Para realizar la compresión, *zlib* ofrece 3 formas distintas.[Gally and Adler, 2002]

- **Sin comprimir.** Se utiliza esta opción cuando los datos de entrada ya están comprimidos de manera que no aumente mucho el tamaño del archivo.
- **Comprimido**. Primero se comprime con LZ77 y después se le aplica la codificación *Huffman*. En este caso, los árboles utilizados en la codificación *Huffman* son generados automáticamente por las especificaciones del algoritmo, por lo que no es necesario almacenarlos con los datos.
- **Comprimido**. Primero con LZ77 y luego se aplica la codificación *Huffman*. Los arboles son creados por el compresor, por tanto, son almacenados junto con los datos.

ZStd

En lo que a **zstd**[15] se refiere, es un algoritmo de compresión basado en el más que mencionado algoritmo LZ77, desarrollado por Yann Collet en 2016, trabajador de Facebook [Collet, 2016]. El objetivo, era crear un algoritmo que no fuese dependiente de la potencia de la CPU, del sistema operativo o de los caracteres utilizados.

[14] https://www.couchbase.com/
[15] Zstandard

Los datos comprimidos por el algoritmo se componen de lo que denominan *Zstandard Frame*, siendo su formato el que se describe en la siguiente tabla.

Magic_Number	Frame_Header	Data_Block	Content_Checksum
4 bytes	2-14 bytes	n bytes	0-4 bytes

Tabla 4.2: Formato de los *frames* de ZSTD

El contenido de estos campos de los *frames* es el siguiente:

- **Magic_Number.** Contiene el identificador del bloque, por ejemplo, 0xFD2FB528, Sigue el formato *little-endian*[16] evitando además patrones triviales, como 0x00 y valores fuera del espacio de UTF-8, de esta manera se reduce considerablemente la posibilidad de que un archivo de texto represente este valor por accidente.

- **Frame_Header.** Este conjunto de *bytes* contiene diferentes *flags* de información sobre el *frame*, así como otros campos opcionales, como la ID del diccionario a utilizar para ese *frame*.

- **Data_Block.** Todos los *frames* se componen de uno o más bloques de datos, cada uno se compone de una subestructura concreta que contiene, los datos y otro tipo de información como el tamaño del bloque o el tipo de bloque (Información descomprimida, información comprimida, bloque reservado...).

- **Content_Checksum.** Contiene el resultado de aplicar la función hash **xxh64**[17] a los datos decodificados y con una semilla fijada en 0. Este campo es opcional y para utilizarlo se debe habilitar el *flag* correspondiente.

De estos campos, destaca el campo *Data_Block* y cuya codificación de los bloques se realiza de la siguiente forma. Se utiliza la denominada *Entropy Encoding* o Codificación de Entropía, en concreto dos tipos, la FSE[18] y la codificación *Huffman*. Esta última se utiliza para codificar literales, mientras que la FSE se utiliza para codificar todos los otros símbolos.

[16] Forma de almacenamiento de los bits en memoria, en este formato los bits menos significativos se almacenan en las posiciones más bajas de la memoria.

[17] http://cyan4973.github.io/xxHash/

[18] *Finite State Entropy* o Entropía de Estado Finito

4.3.2. Cassandra

De los tres SGBD que se tratan en este proyecto, **Cassandra** es el que más opciones de compresión ofrece. [Apache, 2016] Algunas de ellas ya se han comentado en apartados anteriores, este es el caso de *zlib*, *Snappy* y *zstd*. Sin embargo, Cassandra también ofrece otras opciones de compresión, como son los algoritmos **Deflate**, **LZ4**[19] y **LZ4HC**, la cual es simplemente una versión de LZ4 con una mayor compresión (*High Compression*).

DEFLATE

DEFLATE es un algoritmo de compresión sin pérdida basado en el algoritmo LZ77. Fue creado por Phil Katz y publicado en el RFC en 1996 por Peter Deutsch [Deutsch and Katz, 1996].

El algoritmo comprime el flujo de datos entrante en una serie de bloques. Cada uno de los bloques se comprime mediante una combinación del algoritmo LZ77 y la codificación *Huffman*.

Los arboles de *Huffman* de cada uno de los bloques son independientes entre sí, mientras que el algoritmo LZ77 puede hacer referencia a una cadena duplicada en un bloque anterior. De esta manera, se realiza una compresión eficiente de los datos de entrada.

LZ4

El algoritmo LZ4 tiene un gran parecido al algoritmo ZSTD antes descrito, habiendo sido ambos algoritmos desarrollados por Yann Collet. El algoritmo funciona con una estructura de *frames* los cuales se representan de la siguiente forma.[Collet, 2020]

Magic Number	Frame Descriptor	Block	End Mark	Content Checksum
4 bytes	3-15 bytes	n bytes	4 bytes	0-4 bytes

Tabla 4.3: Estructura de los *frames* del algoritmo LZ4

- **Magic Number.** Representación, utilizando el formato *little-endian*, de los bytes de identificación de *frame*.
- **Frame Descriptor.** Contiene información relativa a la configuración del *frame*. Entre esta información están los *flags* que indican si los bloques son independientes

[19] http://lz4.github.io/lz4/

entre sí o deben decodificarse de forma secuencial, o el *flag* de identificación del diccionario a utilizar para la descompresión.

- **Data Blocks.** Contenedor de los datos. Los 4 primeros bytes representan el tamaño del bloque, además a través del valor del bit más significativo se indica el estado de los datos (comprimidos o descomprimidos).
- **End Mark.** Indica el final del flujo de bloques mediante 32 bits fijados a 0 (0x00000000).
- **Content Checksum.** Contiene el resultado de aplicar la función hash, xxHash-32, a los datos iniciales.

5. CAPÍTULO

Sistemas de Gestión de Bases de Datos

De cara a comprender mejor los esquemas definidos para cada uno de los SGBDs, durante este capítulo se realizará una breve descripción de las características y funcionamiento de los dos SGBDs NoSQL tratados en este proyecto, MongoDB y Cassandra.

5.1. SQL frente a NoSQL

5.1.1. Bases de Datos SQL

SQL (*Structured Query Language*) [W3Schools, 2021] es un lenguaje definido para acceder y modificar Bases de Datos relacionales. Estas Bases de Datos, están basadas en el modelo Entidad Relación, propuesto originalmente por E.F. Cold en 1970. [Codd, 2002]

Este modelo, organiza los datos en tablas formadas por filas y columnas, a cada una de las filas se la denomina "tupla". A una de las columnas de las tuplas se la denomina clave primaria o *Primary Key* y contiene un identificador inequívoco de esa tupla dentro de la tabla. Las columnas restantes, representan todas las características a guardar de cada una de las tuplas de la tabla.

Los SGBD más conocidos de este tipo de Bases de Datos son Oracle o MySQL.

5.1.2. Bases de Datos NoSQL

Las Bases de Datos NoSQL (*Non SQL* o *Not Only SQL*) surgieron durante el año 2000, con el objetivo de crear Bases de Datos escalables, con *queries* rápidas y fácil de amoldar a los constantes cambios en las aplicaciones [Schaefer, 2021].

En general hay diferentes tipos de Bases de Datos NoSQL, siendo las más conocidas las siguientes.

- **Documentos.** Se enfocan a un uso genérico, la idea es almacenar los datos en documentos, por ejemplo de tipo *JSON*. Este es el caso del popular SGBD MongoDB.
- **Clave-Valor.** Tiene un funcionamiento similar a las tablas Hash o los diccionarios, en los que una clave única se asocia a una colección de objetos. Un SGBD que sigue esta implementación es DynamoDB.
- **Tabular.** En lo que al almacenamiento y visualización de datos se refiere, los SGBD de este tipo tienen una estructura parecida a las clásicas SQL. Almacenan los datos en tablas que tienen filas y columnas dinámicas. Este tipo de Bases de Datos se enfoca principalmente a aplicaciones con gran volumen de datos con patrones de *queries* conocidos. En este tipo entra el SGBD Cassandra.
- **Orientadas a Grafos**. En general, son Bases de Datos enfocadas en las relaciones entre los datos, por tanto, sirven para analizar datos conectados entre sí. En ese grupo de Bases de Datos destaca el SGBD Neo4J.

5.1.3. Diferencias entre ambas

Cuando se habla de las diferencias entre las Bases de Datos SQL y las NoSQL, la principal diferencia que nos viene a la mente es la escalabilidad en los datos. A pesar de que las Bases de Datos relacionales llevan ya mucho tiempo en el mercado, en la mayoría de los casos, cuando se habla del almacenamiento y gestión de un volumen de datos muy grande, las Bases de Datos relacionales no son capaces de superar a las NoSQL. [Schaefer, 2021]

Otra de las diferencias principales entre las Bases de Datos Relacionales y las NoSQL es el no mantenimiento de los principios *ACID* por parte de las segundas. [Dave, 2012]. El principio *ACID* (**A**tomicidad, **C**onsistencia, a**I**slamiento y **D**urabilidad) es el principio en

el que se basan las Bases de Datos Relacionales y que garantizan, que tras realizar una serie de transacciones, el estado de la Base de Datos siga siendo consistente.

En relación al principio *ACID* existe el no tan conocido teorema *CAP* (**C**onsistencia, *A*vailability o Disponibilidad y **P**artition Tolerance o Tolerancia a Fallos). Este teorema defiende que todo Sistema de Gestión de Bases de Datos cumple un máximo de dos y nunca las tres propiedades del teorema. No obstante, esto no significa que se tenga que elegir entre todo o nada, sino que se puede "sacrificar" algo de disponibilidad para poder obtener una mayor consistencia. En general, las Bases de Datos NoSQL no son muy amigas de la Consistencia perfecta, por lo que no cumplen el principio *ACID*, antes comentado, ni tampoco la *C* (Consistencia) del teorema *CAP* o no al 100 %.

5.2. MongoDB

MongoDB[1] es un sistema de bases de datos NoSQL enfocado a documentos y de código abierto. Al ser una base de datos enfocada a documentos, los datos que se guardan en ella no son en tablas, como en las bases de datos SQL tradicionales, sino que se guardan en documentos de tipo JSON.

```
{
  "_id": "5cf0029caff5056591b0ce7d",
  "firstname": "Jane",
  "lastname": "Wu",
  "address": {
    "street": "1 Circle Rd",
    "city": "Los Angeles",
    "state": "CA",
    "zip": "90404"
  },
  "hobbies": ["surfing", "coding"]
}
```

Listing 5.1: Documento de ejemplo JSON almacenado en MongoDB

Al guardar los documentos en este formato permite [MongoDB, 2021a], entre otras características.

- **Flexibilidad en los esquemas de los documentos**. El modelado enfocado a documentos de MongoDB permite a los desarrolladores modelar y manipular fácilmente

[1] https://www.mongodb.com/

los datos. Así mismo, al utilizar el formato JSON, permite tener diferentes campos en distintas entradas de una misma colección.

- **Acceso a datos de código nativo**. La gran mayoría de Bases de Datos relacionales utilizan librerías de mapeo, las cuales convierten los datos de las tablas de la BD a objetos del propio lenguaje, como es el caso de *Hibernate*[2] para Java. Al ser una BD de documentos MongoDB no necesita estas librerías, ya que sus estructuras de datos son accesibles mediante las estructuras propias de cada lenguaje de forma nativa.

- **Consultas y analíticas potentes**. MongoDB esta diseñada para ser una BD en la que el acceso a datos sea fácil y sencillo y que rara vez se precise realizar operaciones como los costosos *joins* de SQL.

5.3. Cassandra

Cassandra[3] es un sistema de bases de datos NoSQL tabular, es decir, estructura los datos de forma similar a las bases de datos relaciónales, utilizando tablas, filas y columnas, aunque el formato y nombre de las columnas puede variar en cada una de las filas.

A continuación se listan algunas de las características que han hecho a Cassandra uno de los SGBD NoSQL más utilizados. [DataFlair, 2018]

- **Open Source.** Uno de los principales atractivos de este SGBD. Se trata de un proyecto de código abierto de Apache, gracias a ello, Cassandra cuenta con una enorme comunidad detrás.

- **Arquitectura *Peer-to-peer***. Cassandra almacena los datos en nodos, los cuales están distribuidos en forma de anillo. Los datos son replicados en cada uno de los nodos, de manera que no existe un nodo "maestro". Además, esto hace que no exista un único punto de fallo.

- **Gran escalabilidad.** Esta es una de las grandes características de Cassandra. Cada uno de los *clusters* ofrece una gran escalabilidad, en cualquier momento se pueden añadir o eliminar cualquier número de nodos.

[2] https://hibernate.org/
[3] https://cassandra.apache.org/

- **Alta disponibilidad y tolerancia a fallos.** Gracias a la replicación de los datos, Cassandra ofrece una alta disponibilidad y una fuerte tolerancia a fallos. Si un nodo falla los datos estarán disponibles en todo momento en cualquiera de los otros nodos. Al número de copias de los datos dentro de un mismo *cluster* se le denomina *replication factor* o factor de replicación. Para un factor de replicación de 2, por ejemplo, existirán des copias por cada una de las filas de datos, estando cada copia en un nodo diferente.

- **Consistencia modificable**. Cassandra ofrece distintos niveles de consistencia, [DataStax, 2021] aunque se pueden reducir a dos tipos generales. Un primer tipo denominado consistencia eventual, el cual, en cuanto el *cluster* acepta la escritura del dato, el sistema se asegura de recibir la aprobación del cliente. El segundo tipo, consistencia fuerte, se asegura de que las escrituras en un nodo se notifican al resto de nodos en los que los datos están replicados. No obstante, no es necesario elegir un único modo, sino que se puede realizar una mezcla de ambos.

6. CAPÍTULO

Diseño de la aplicación

En el siguiente capítulo se detallarán el contenido que debe ser almacenado en cada una de las bases de datos a diseñar, así mismo, también se especificará una primera aproximación del diseño en ambos Sistemas de Gestión de Bases de Datos.

6.1. Formato de las facturas

Como ya se ha comentado, el objetivo del proyecto consiste en diseñar e implementar una base de datos, en diferentes SGBD, capaz de almacenar, de forma eficiente, los datos recibidos. Por ello, una buena forma de orientar el diseño de la Base de Datos que utilizará la aplicación es conociendo el formato en el que se reciben los datos. En el caso de TicketBai, los datos pertenecientes a una factura se reciben en formato *XML*[1].

Estas facturas *XML* siguen el formato, en forma de *XMLSchema*, proporcionado por el cliente. Cada factura, esta formada por cinco bloques, **Cabecera**, **Sujetos, Factura, HuellaTBAI** y **Signature** [Diputación Foral, 2021].

6.1.1. Cabecera

El bloque Cabecera de la factura contiene, simplemente, un número identificativo de la versión de la estructura del fichero TicketBAI utilizado.

[1] eXtensible Markup Language

6.1.2. Sujetos

Este bloque contiene los datos relativos al emisor y destinatario(s) de las factura. Del emisor, se guardan valores como el NIF o la denominación social del emisor.

Para cada uno de los posibles destinatarios de la factura, se guarda el NIF en caso de que sea un destinatario nacional o el país y si número de identificación correspondiente en caso de que sea de un país extranjero.

6.1.3. Factura

El bloque Factura contiene todos los datos relacionados con la fecha de expedición o los productos adquiridos, entre otros, es por ello, que es el bloque más extenso de toda la factura. Por tanto, el bloque se divide en tres partes.

Cabecera Factura

Este primer sub-bloque contiene los datos que identifican una factura concreta, estos campos son, el número de factura y la fecha y hora de expedición de la factura. Así mismo, también se almacenan en este bloque otros valores que indican si una factura es simplificada o si es una factura rectificativa.

Datos Factura

Este sub-bloque contiene, principalmente, los detalles de la factura, es decir, los bienes o servicios adquiridos y que quedan recogidos en la factura. Por cada uno de ellos, se almacena información relativa al importe con y sin IVA aplicado, o una descripción del producto.

Además, también se almacena el importe total de la factura o valores que indican el tipo de IVA aplicado a la factura.

Tipo Desglose

Cada una de las facturas tiene la opción de ofrecer un desglose del IVA de cada uno de los detalles que conforman la mencionada factura. El desglose que se ofrece puede ser de

dos tipos diferentes, *Desglose de factura* o *Desglose por tipo de operación*. El primero se enfoca a un desglose más genérico, mientras que el segundo diferencia el desglose para servicios prestados y productos materiales.

Independientemente del tipo de desglose se diferencian dos casos, el primero, el caso menos común, que el producto o productos no estén sujetos al impuesto por la causa posteriormente indicada, el segundo, el caso quizás más común que los productos de la factura estén sujetos al pago de IVA. Aún así, en este segundo caso existe la posibilidad de que algunos de los productos o servicios estén exentos del pago de IVA, indicando en su caso la causa y la base imponible exenta del pago de este impuesto. Para los productos o servicios que no están exentos del pago del IVA se agruparán por su tipo de operación y posteriormente por el tipo impositivo que se aplica a los productos.

6.1.4. HuellaTBAI

En este bloque se guarda la información relacionada con el *software* que ha emitido la factura, entre estos datos se encuentran campos como el número de alta-inscripción en el registro del Software TicketBAI o la versión del *software* utilizada.

6.1.5. Signature

El bloque *Signature* contiene la firma electrónica de las facturas TicketBAI. Dada la normativa vigente, las facturas XML deben ser firmadas en el el formato XAdES[2]. Este formato es el conjunto de extensiones a las recomendaciones *XML-DSig* de manera que sean adecuadas para firmar documentos XML electrónicamente según el reglamento Nº 910/2014 del Parlamento Europeo [Parlamento Europeo, 2014].

XML-DSig o *XML Digital Signature* define la sintaxis y reglas de procesamiento para firmar documentos XML digitalmente. Estas reglas y sintaxis están definidas en el documento correspondiente de la W3C [W3C, 2013]. La estructura básica del campo de firma de un documento es la siguiente.

```
<Signature>
    <SignedInfo>
        <CanonicalizationMethod />
        <SignatureMethod />
        <Reference>
```

[2] ***X**ML **Ad**vanced **E**lectronic **S**ignatures*

```
            <Transforms />
            <DigestMethod />
            <DigestValue />
        </Reference>
        <Reference /> etc.
    </SignedInfo>
    <SignatureValue />
    <KeyInfo />
    <Object />
</Signature>
```

Listing 6.1: Estructura básica del campo *Signature* de un documento XML

El listing 6.1 contiene la estructura básica de la firma. No obstante, el tipo de firma XAdES ofrece diferentes tipos de formato de firma. Estos tres son, *Detached*, *Enveloping* y *Enveloped*. De estos tres, el formato solicitado por el cliente y a utilizar es el último, siendo las diferencias entre los tres las siguientes. [Administración Electrónica, 2021]

- **Detached.** Una firma XML en este formato permite tener la firma de forma separada e independiente del contenido firmado, de manera que la firma se relaciona con el contenido firmado mediante una referencia de URI. El uso más común de este tipo de firma es en la descarga de ficheros.
- **Enveloping.** En este formato de firma la única estructura existente es la de la propia firma. El contenido firmado se encuentra dentro de un elemento denominado *Object*.
- **Enveloped**. Este formato permite a un documento XML contener su propia firma digital. Por tanto, mientras que en los casos anteriores se podía firmar cualquier tipo de documento, mediante este formato de firma solo se podrán firmar documentos XML.

6.2. *Queries* más significativas

Una de las características de los SGBDs NoSQL es la dependencia de las consultas que se van a realizar. Conocer con anterioridad algunas de las consultas, o al menos las más significativas, ayuda mucho en la definición del esquema que seguirá el SGBD utilizado, MongoDB y Cassandra.

Por ello, tras una serie de reuniones con el cliente, en las que se obtuvo la información necesaria para realizar el diseño, se definieron las siguientes consultas.

Obtener una factura mediante el código QR

Frecuencia de consulta estimada: Muy frecuente. Cientos de miles de consultas diarias.

Todas las facturas emitidas mediante el software TicketBAI vienen con un código QR. Según las especificaciones establecidas, el código QR contiene el siguiente enlace `https://tbai.egoitza.gipuzkoa.eus/gr/`[3], junto con los siguientes parámetros.

- **id**. Identificador TicketBAI, este identificador determina inequívocamente a cada una de las facturas. El identificador se forma de la siguiente manera.
  - El texto **TBAI**.
  - Los nueve caracteres del NIF de la persona o entidad emisora de la factura.
  - La fecha de expedición de la factura. Esta se representará en formato DDMMAA.
  - Los trece primeros caracteres del campo *SignatureValue* del fichero XML, es decir, el campo que contiene la firma de la factura.
  - Tres caracteres que se corresponen con un código de detección de errores.

  Cada uno de los elementos listados del identificador va separado por el carácter "-", siendo el esquema genérico:
  TBAI-NNNNNNNNN-DDMMAA-FFFFFFFFFFFFF-CRC
- **s**. Representa la serie que sigue la factura. El contenido de este parámetro vendrá determinado por el campo *SerieFactura* del fichero XML que contiene los datos de la factura.
- **nf**. Representa el número de la factura emitida. Se corresponde con el campo *NumFactura* del fichero XML que contiene los datos de la factura.
- **i**. Contiene el importe total de la factura

Obtener las facturas emitidas por un NIF

Frecuencia de consulta estimada: Frecuente. Cientos de miles de consultas mensuales.

Como es de esperar, uno de los principales tipos de consultas es la obtención de las facturas por su NIF. Entre este tipo de consultas la que más destaca es la obtención de facturas emitidas por un NIF en unas determinadas fechas.

[3]Este es el enlace establecido para Gipuzkoa, en Bizkaia el enlace será `https://batuz.eus/QRTBAI/`, mientras que en Araba será `https://ticketbai.araba.eus/TBAI/QRTBAI`.

Obtención de datos estadísticos

Frecuencia de Consulta estimada: Poco frecuente. Pocas consultas mensuales pero realizando cálculos con un gran volumen de datos.

Una de las características que deben permitir realizar los diseños, es la obtención de datos estadísticos. Uno de los propósitos de TicketBAI es permitir a los emisores de facturas ver datos estadísticos que permitan comparar su negocio con otros negocios del mismo. A pesar de que estas consultas sean poco frecuentes el diseño realizado debe permitir el cálculo de las diferentes estadísticas a realizar. Algunos ejemplos de este tipo de consultas serían, "*¿A qué países exportan principalmente las empresas de mi sector?*" o "*¿Cual es el importe medio de los tickets que he emitido esta semana?*.

Agrupación de facturas

Frecuencia de consulta estimada: Media. Miles de consultas mensuales.

A pesar de que no es una consulta como tal, de cara a obtener un mejor almacenamiento de las facturas se ha decido realizar una agrupación de la mismas. Es decir, pasado un periodo de tiempo preestablecido, por ejemplo un año, las facturas almacenadas en la base de datos se moverán a una nueva tabla o colección en la que se agruparán todas las facturas emitidas en ese mismo año. De esta manera, tras comprimir el conjunto anual de facturas se reduce considerablemente el espacio ocupado en la base de datos.

6.3. Esquema de la aplicación en Cassandra

Partition Key		**Clustering Key**	**Data**		
Nif Emisor	Fecha Exp	Ident-TBAI	SerieFact	Importe	Num Fact

Tabla 6.1: Tabla 1 Cassandra

La tabla 6.1 muestra los datos que contendrá la tabla a responder a la consulta relacionada con el QR. Debido a la alta frecuencia estimada sobre estos datos se ha decidido crear una tabla que gestione la carga.

La clave de partición (*Partition Key*) estará compuesta por el NIF del emisor de la factura y la fecha en la que está expedida. Dentro de esta agrupación, se ordenarán las facturas

(*Cluestering key*) mediante el Identificador TicketBai. En cuanto a los datos guardados en esta tabla, son aquellos que se mostrarán al realizar la consulta sobre el QR.

Partition Key		**Clustering Key**		**Data**		
Nif Emisor	Fecha Exp	Hora Exp	Importe	Descripción	Detalles	xml

Tabla 6.2: Tabla 2 Cassandra

Esta segunda tabla viene a responder a las otras dos consultas principales. La tabla permitirá, no solo obtener los datos más significativos de las facturas emitidas por un NIF en una fecha determinada, sino que también contiene los datos principales que permitirán realizar los estudios estadísticos deseados por el cliente. A destacar el campo *xml* el cual contiene la factura completa comprimida a la que corresponden los datos.

Partition Key		**Clustering Key**	**Data**	
Nif Emisor	Fecha Inicio	Fecha Inicio	Fecha Fin	Agrupación

Tabla 6.3: Tabla 3 Cassandra

Esta tercera y última tabla permitirá el almacenamiento de las facturas agrupadas. La tabla contendrá el NIF emisor de las facturas, la fecha en la comienza la agrupación, la fecha en la que finaliza y el conjunto de facturas emitidas en ese rango de fechas comprimidas. De esta manera, para obtener una factura "antigua", se accedería a la fila correspondiente de la tabla y se buscaría la factura en la agrupación de facturas correspondiente. Esta última búsqueda se realizará mediante una característica común de todos los ficheros XML, su cabecera.

```
1 <?xml version="1.0" encoding="UTF-8" standalone="no" ?>
```

Listing 6.2: Cabecera común a todos los ficheros xml

6.4. Esquema de la aplicación en MongoDB

Mientras que Cassandra es un SGBD muy dependiente de las consultas que se realizarán sobre la base de datos, MongoDB ofrece una mayor flexibilidad. Por ello, en base a las consultas más significativas definidas en la sección 6.2, se ha decidido crear una única

colección que contiene los datos a los que se accederán con mayor frecuencia junto con la factura original comprimida. El esquema que seguiría una factura será el que se muestra a continuación.

```
{
    _id : {
        type: Schema.Types.String,
        required: true
    },
    NIF : {
        type: Schema.Types.String,
        required : true
    },
    FechaExpedicionFactura : {
        type: Schema.Types.Date,
        required : true
    },
    HoraExpedicionFactura : {
        type: Schema.Types.Date,
        required : true
    },
    ImporteTotalFactura : {
        type : Schema.Types.Number,
        required: true
    },
    SerieFactura : {
        type: Schema.Types.String,
        required : true
    },
    NumFactura : {
        type: Schema.Types.String,
        required : true
    },
    Descripcion :{
        type: Schema.Types.String,
        required : true
    },
    DetallesFactura: {
        required : true,
        type : [
            {
                DescripcionDetalle : {
                    type: Schema.Types.String,
                    required : true
                },
                Cantidad : {
                    type: Schema.Types.Decimal128,
                    required : true
                },
                ImporteUnitario : {
```

```
                    type: Schema.Types.Number,
                    required : true
                },
                ImporteTotal : {
                    type: Schema.Types.Number,
                    required : true
                }
            }
        ]
    },
    FacturaComprimida : {
        required : true,
        type : Schema.Types.String
    }
}
```

Listing 6.3: Esquema de las facturas en MongoDB

De todos los campos mostrados, los únicos a destacar son el campo *_id* y *FacturaComprimida*. El primero es un campo identificador propio de MongoDB, el cual identifica inequivocamente al objeto en cuestión, en este caso, el valor de este campo será el identificador TicketBai, el cual tenía el formato *TBAI-NNNNNNNNN-DDMMAA-FFFFFFFFFFFFF-CRC*.

El segundo campo, *FacturaComprimida*, contendrá el resultado de comprimir el factura utilizando el algoritmo de compresión GZip. De esta manera, se tendrá la factura completa almacenada, de forma eficiente, preparada para ser utilizada en caso de tener que realizar consultas sobre algunos de los campos no establecidos como prioritarios.

No obstante, para poder suplir la *query* correspondiente a la agrupación de las facturas, se ha decido crear una nueva colección dedicada únicamente a ello. El esquema de esta colección es el siguiente.

```
{
    _id : {
        type: Schema.Types.String,
        required: true
    },
    idents : {
        type: Schema.Types.Array,
        required : true
    },
    agrupacion : {
        type: Schema.Types.String,
        required : true
    }
}
```

Listing 6.4: Esquema de la colección de agrupación de facturas

Esta colección se compone de 3 campos. El primero, el campo identificador, que debe ser único, contiene el NIF emisor de las facturas, la fecha de inicio de la agrupación y la fecha fin. El formato, quedaría de la siguiente forma “NIF/DD-MM-AAAA/DD-MM-AAAA”.

Para facilitar la búsqueda de una factura concreta dentro de la agrupación de facturas se utilizará el campo *idents*, el cual contiene los identificadores TicketBai, ordenados, de cada una de las facturas que se han agrupado.

7. CAPÍTULO

Pruebas sobre el diseño

Durante este capitulo se detallarán las pruebas de rendimiento diseñadas de cara a probar el correcto funcionamiento del sistema. Así mismo, también se mostrarán los resultados obtenidos en la ejecución de las pruebas junto con los cambios realizados, en caso de que fuese necesario.

En cuanto al tipo de pruebas de rendimiento, únicamente se han realizado pruebas relacionadas con los tiempos de respuesta a la hora de recuperar los datos de diferentes formas, y la escalabilidad de estos datos. El motivo de esta elección ha sido que el sistema en el que se han hecho las pruebas no es el sistema final, por tanto, los resultados de tipo de pruebas relacionadas con la evaluación de la carga máxima que soporta el sistema o tests de resistencia no serían relevantes.

Tras analizar diferentes metodologías de *testing* de rendimiento de los sistemas, el esquema utilizado, el cual es el más extendido, ha sido el siguiente. [Guru99, 2021]

1. **Identificar del entorno de testeo.** Identificar las capacidades y limitaciones de la máquina en la que se ejecutarán las pruebas.

2. **Determinar los criterios de desempeño.** Identificar los criterios en los que se considerará válida la prueba o pruebas a realizar.

3. **Planificar y diseñar los tests de rendimiento.** Planificación del campo a probar y posterior diseño de los datos a insertar para probar el campo antes mencionado.

4. **Configurar el entorno de pruebas.** Preparación de las herramientas que se emplearán para ejecutar y analizar las pruebas.

5. **Implementar los tests.** Configuración del generador de facturas para que genere facturas con las características planificadas.

6. **Ejecutar los tests.** Ejecución de las pruebas, en este caso se realizarán las consultan e inserciones diseñadas a cada una de las bases de datos.

7. **Analizar, realizar cambios y retestear.** Analisis de los resultados obtenidos en los tests, en caso no obtener un resultado satisfactorio se realizarán los cambios pertinentes y se ejecutarán las pruebas de nuevo.

7.1. Descripción y ejecución de las Pruebas a realizar

Tal como se comentaba anteriormente, las pruebas definidas a continuación se han diseñado con el objetivo de evaluar el rendimiento de las bases de datos diseñadas y de evaluar posibles modificaciones del diseño que mejoren el rendimiento.

En términos generales, se han distinguido dos tipos de pruebas. Un primer grupo, encargado de probar el funcionamiento de la obtención de facturas, y datos de las mismas, individuales. Y un segundo grupo, encargado de probar la obtención de una factura, o un campo de la misma, de un conjunto de facturas agrupadas. Dentro de cada uno de estos grupo se han realizado dos tipos de pruebas principales, una con facturas “pequeñas”, es decir, no contienen muchos datos y su tamaño en bytes en pequeño. Las otras facturas son “grandes”, con la mayor cantidad de datos posibles, siendo su tamaño máximo aproximado de 500KB.

Cabe destacar que todas las pruebas se han realizado sobre la misma máquina y con aproximadamente 30GB de datos ya insertados en las bases de datos, por lo que se han podido ver las diferencias y similitudes de rendimiento entre ambos SGBDs. En cuanto a los gráficos mostrados, la unidad de todos los tiempos que se muestran es *milisegundos*, en caso contrario, se indicará en la leyenda del gráfico.

7.1.1. Pruebas para seleccionar el algoritmo de compresión

Dado que una de las principales características del sistema es el almacenamiento de los datos comprimidos para así ahorrar espacio, el primer paso antes de realizar pruebas de

inserción y consultas es elegir el algoritmo de compresión. En el capitulo 4, Descripción de las técnicas de compresión, se había definido la teoría detrás de cada una de las técnicas de compresión a probar en el proyecto.

A continuación, se muestran las pruebas a las que se han sometido los algoritmos y que han servido para decantarse por uno de ellos.

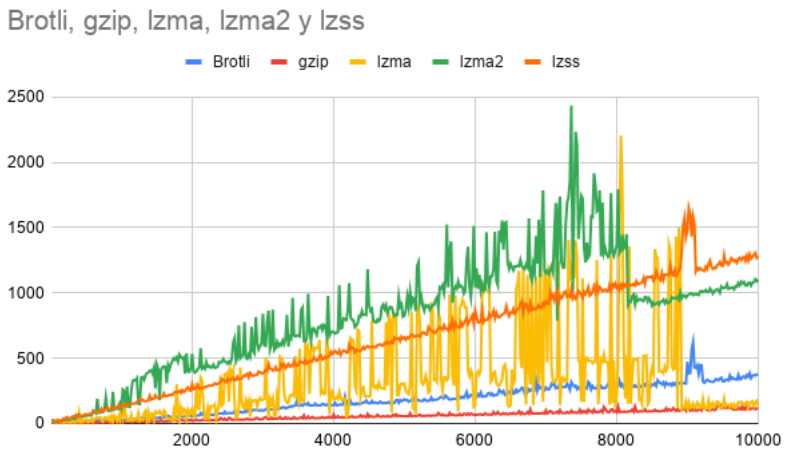

Figura 7.1: Comparación del tiempo de compresión entre algoritmos

La figura 7.1, contiene el gráfico resultante de comprimir agrupaciones de hasta 10000 facturas. El eje horizontal representa el número de facturas agrupadas, mientras que el eje vertical, representa el tiempo empleado, en milisegundos, por cada uno de los algoritmos a comprimir las agrupaciones de facturas. Tal como se puede observar en el gráfico, la técnica que ofrece mejores resultados es GZip, la cual mantiene un tiempo inferior a los 250 ms durante todos los casos

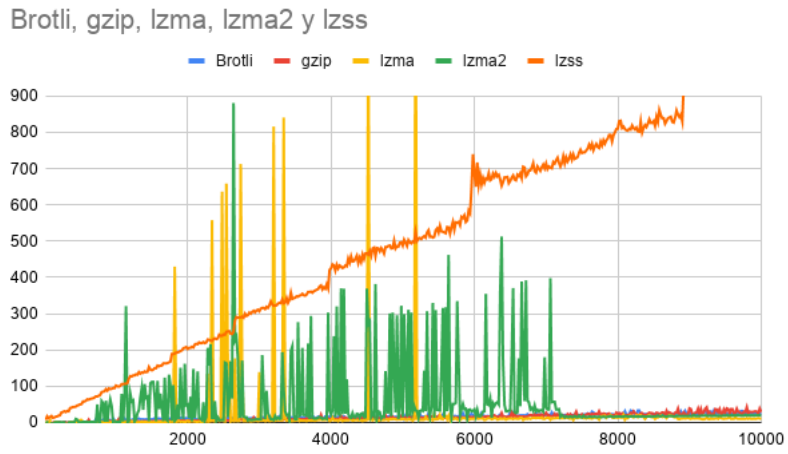

Figura 7.2: Comparación del tiempo de descompresión entre algoritmos

La figura 7.2, muestra el gráfico resultante de descomprimir la agrupación de 10000 facturas anterior. El eje horizontal, muestra de nuevo el número de facturas agrupadas en cada momento, mientras que el eje vertical muestra el tiempo, en milisegundos, de descomprimir cada una de las agrupaciones. Como se ve en el gráfico, las técnicas Brotli y GZip tienen un tiempo de descompresión casi constante, el cual se mantiene inferior a los 50 milisegundos.

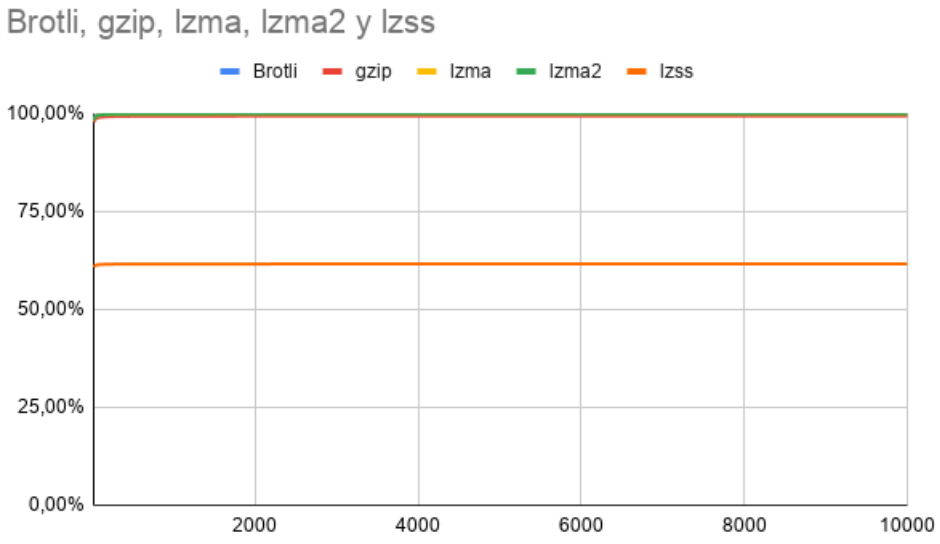

Figura 7.3: Ratio de compresión de cada una de las técnicas de compresión abordadas

Por último, la figura 7.3, muestra el ratio de compresión de cada una de las técnicas tratadas.

El eje horizontal representa el número de facturas agrupadas, mientras que el eje vertical representa el ratio de compresión de las técnicas, en tanto por ciento. Tal como se ve, todas las técnicas, a excepción de Lzss, ofrecen un ratio de compresión cercano al 100 %, por lo que en ese aspecto todas serían una buena opción.

Teniendo en cuenta los datos anteriores y dadas las necesidades de la aplicación, se ha decidido utilizar el algoritmo Gzip. Esto se debe a que Gzip es la técnica que ofrece una mejor compresión y descompresión de los datos. Así mismo, tiene un ratio de compresión muy alto, por lo que reducirá considerablemente el tamaño de las facturas originales.

7.1.2. Pruebas sobre facturas individuales

En general, las pruebas a las que se han sometido los diseños de las BDs, tanto en MongoDB como en Cassandra, están estrechamente relacionadas con la obtención de los datos de las bases de datos, ya que es el aspecto más crítico. No obstante, dado que algunos de los cambios en los diseños están relacionados con los tiempos de inserción de los datos se ha decidido documentar esta parte junto con los problemas surgidos.

Tiempo de inserción de facturas

Los gráficos que se muestran a continuación representan los tiempos en realizar los pasos de obtención de datos a guardar en RAW, compresión de factura e inserción en la base de datos correspondiente.

Pruebas sobre MongoDB

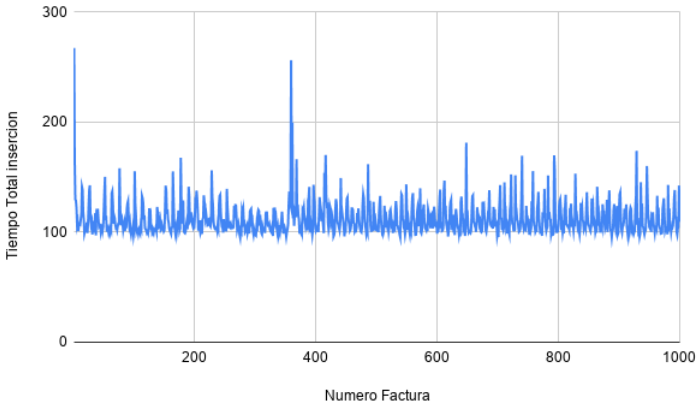

Figura 7.4: Tiempo total de inserción de facturas individuales con pocas lineas de detalle

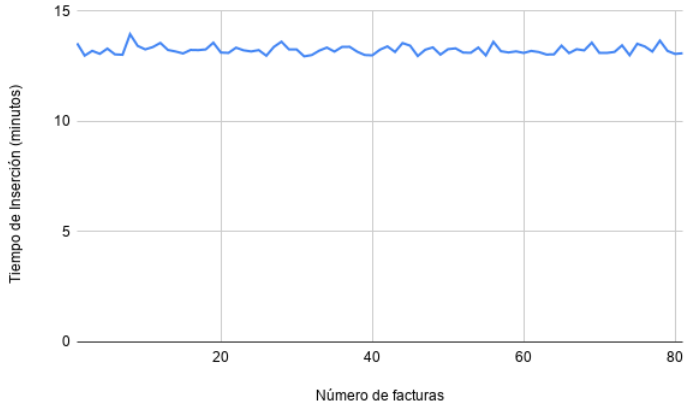

Figura 7.5: Tiempo total de inserción de facturas individuales con muchas lineas de detalle

Las figuras 7.4 y 7.5 muestran la diferencia del tiempo de inserción entre facturas "pequeñas" y "grandes", respectivamente. Tal como se ve en las figuras la diferencia de tiempo entre ambas es considerable. Para realizar una inserción de una factura con poco contenido, con una tamaño medio de 10KB, el tiempo de inserción esta entre 100 y 200 milisegundos, mientras que para una factura con mucho contenido, un tamaño medio de 500KB, el tiempo de inserción se encuentra sobre los 13 minutos. Este enorme tiempo de espera entre factura y factura es el motivo de que la muestra de facturas grandes no llegue a las 100 facturas, no obstante, es de un tamaño suficiente como para ser una muestra representativa.

Dado que esta situación es insostenible, ya que para un sistema con un gran número de peticiones diarias no es permisible una espera de 13 minutos entre petición y petición. Tras realizar las pruebas necesarias para averiguar el punto en el que se pierde tanto tiempo el resultado ha sido el siguiente.

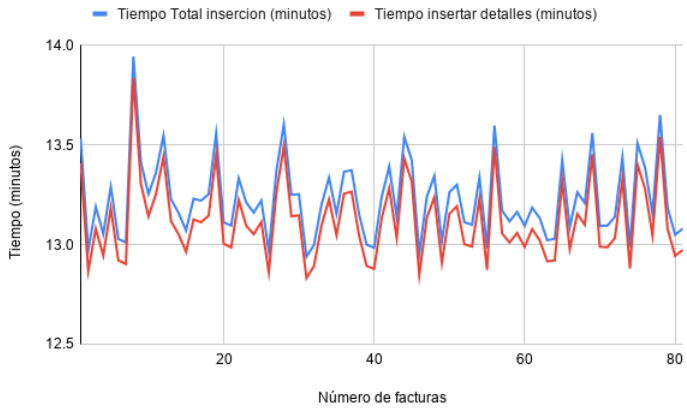

Figura 7.6: Tiempo de inserción total frente al tiempo de inserción de los detalles

Tal como se puede observar en la figura 7.6, a la hora de realizar la inserción de facturas de un gran tamaño, la gran mayoría del tiempo se emplea en la lectura y posterior inserción de los detalles de la factura en la base de datos. Para ver la corrección sobre el diseño correspondiente junto con los nuevos resultados obtenidos tras la corrección ver la sección 7.2.1, Corrección M.1.

Pruebas sobre Cassandra

Anteriormente se ha visto el problema surgido en MongoDB con el almacenamiento de los datos en RAW. No obstante, en Cassandra, desde el inicio no se planteó el almacenamiento de los detalles de la factura en RAW. Es por ello que en este SGBD no se ha tenido ningún problema relacionado con la inserción de las facturas, siendo los resultados de inserción y compresión los siguientes.

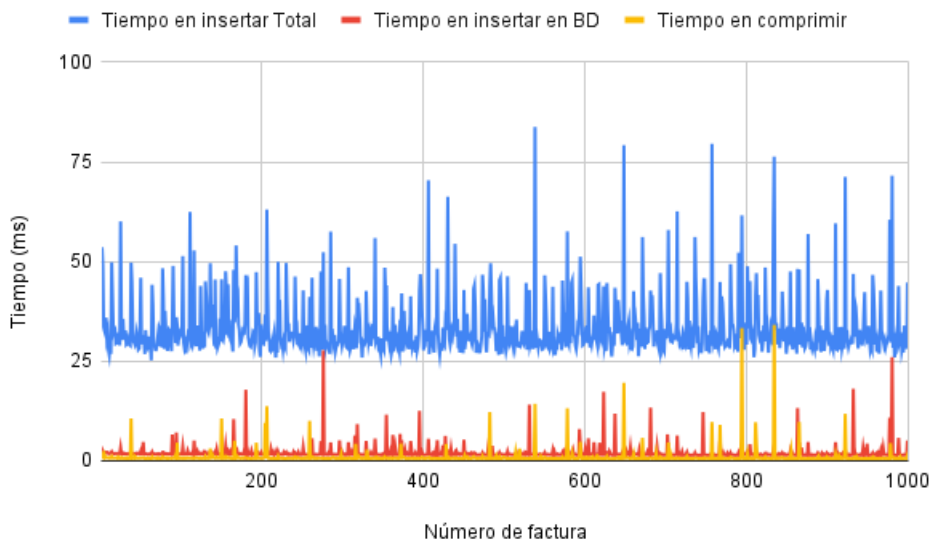

Figura 7.7: Tiempos de inserción y compresión de factura pequeña en Cassandra

Los resultados de este conjunto de facturas son bastante similares a los obtenidos en MongoDB, ver figura 7.31, no obstante, el tiempo de inserción total en Cassandra es ligeramente menor. A continuación se muestra un gráfico con la media de tiempos, para las facturas pequeñas, de realización de cada uno de los pasos de la inserción junto con el tiempo total.

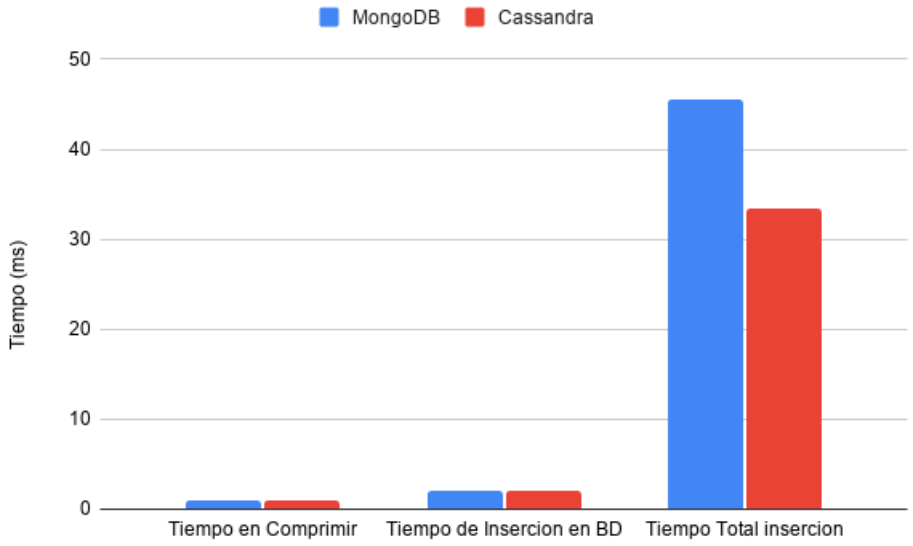

Figura 7.8: Comparación de fases de inserción entre Mongo y Cassandra con facturas pequeñas

En cuanto a las facturas de gran tamaño el tiempo total de inserción de estas es el siguiente.

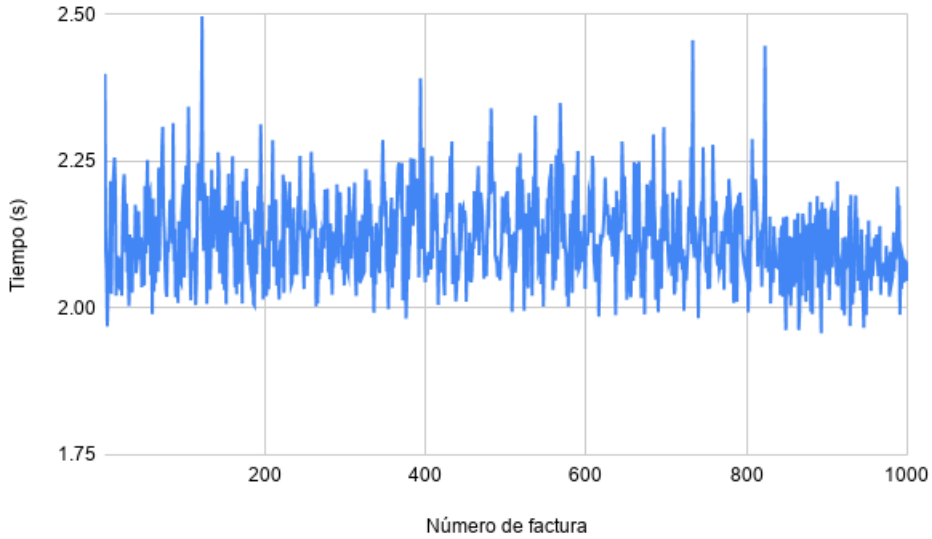

Figura 7.9: Tiempo de inserción total de facturas grandes en Cassandra

Si se comparan estos tiempos con los obtenidos en MongoDB, ver figura 7.32, la mejora es muy pequeña pero existente. El tiempo total de inserción de cada uno de los datos ha sido ligeramente más rápido, apenas le supera por 250 milisegundos.

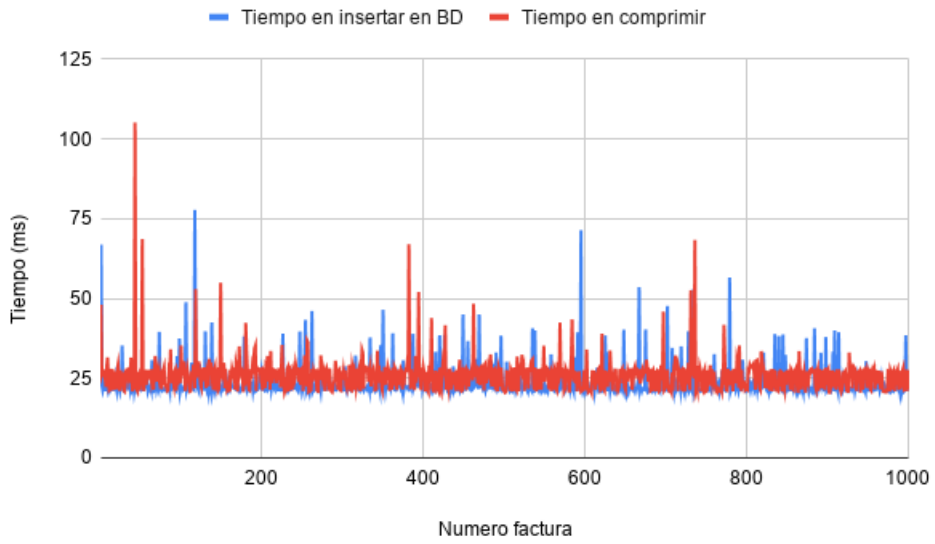

Figura 7.10: Tiempo de inserción en BD y compresión de facturas grandes en Cassandra

En cuanto al tiempo de inserción en la BD los tiempos en Cassandra son ligeramente peores que en MongoDB, unos 10 milisegundos de diferencia media. Aunque en principio estos resultados no encajan con la diferencia de tiempo total mostrada antes, en la que Cassandra tenía un tiempo de insertar total inferior a MongoDB, el resultado es de esperar, ya que en MongoDB se almacenan algunos datos más que en Cassandra, como puede ser la descripción de la factura.

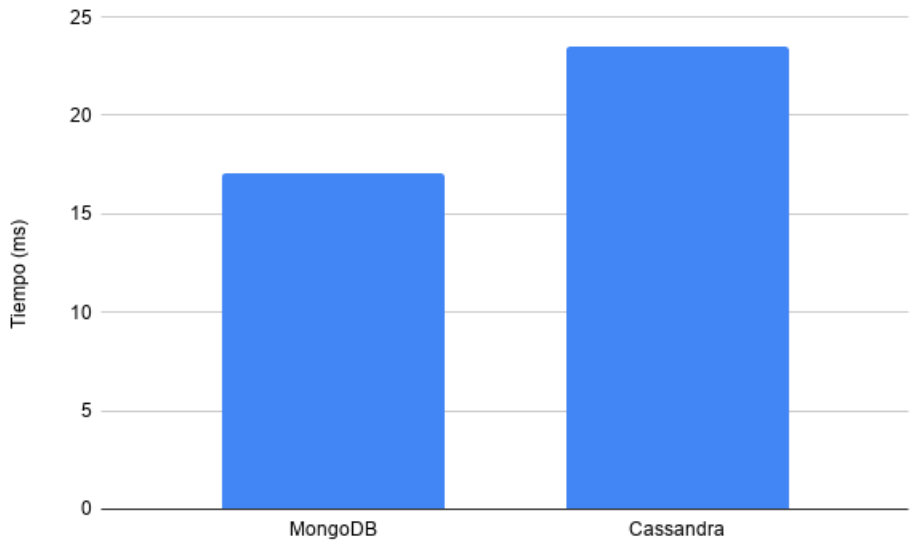

Figura 7.11: Tiempo de inserción de facturas grandes en ambos SGBDs

Tiempo en recuperar una factura

En este grupo de pruebas se ha identificado el caso relacionado con la obtención de una factura mediante el QR, es decir, se busca una factura mediante el identificador TBAI, y se devuelve la factura completa.

Pruebas sobre MongoDB

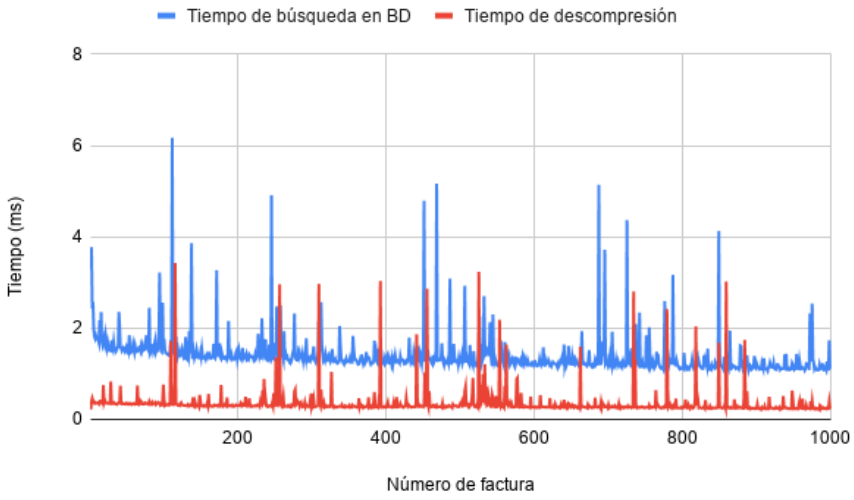

Figura 7.12: Tiempo de búsqueda de factura pequeña en MongoDB

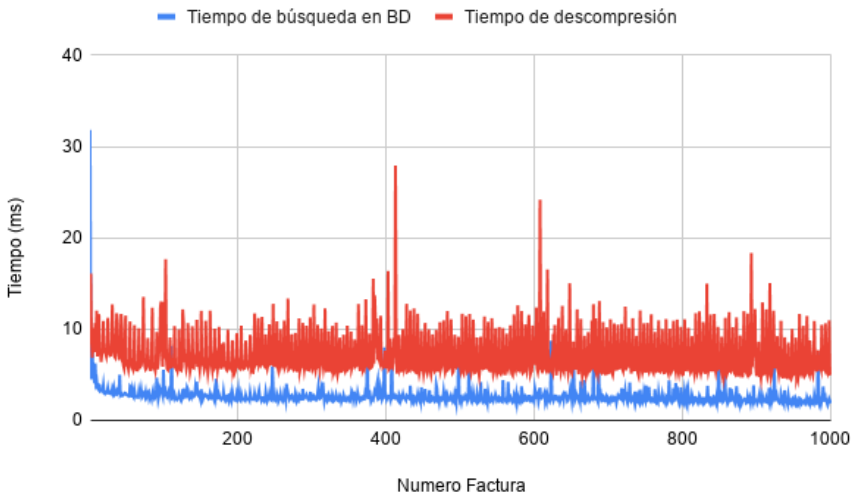

Figura 7.13: Tiempo de búsqueda de factura grande en MongoDB

Las dos gráficas anteriores, figuras 7.12 y 7.13, muestran el tiempo total de búsqueda de una factura pequeña y grande, respectivamente. El tiempo se divide entre tiempo de realización de la consulta y devolución del dato, y el tiempo de descompresión de la factura. Los tiempos en ambos casos son muy positivos, en las facturas pequeñas, el tiempo en realizar todos los pasos no supera los 5 milisegundos, mientras que para las facturas grandes tarda aproximadamente 20 milisegundos.

Pruebas sobre Cassandra

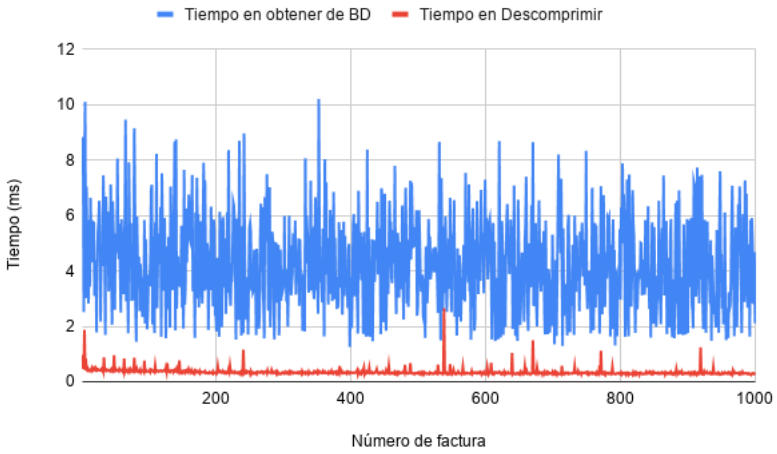

Figura 7.14: Tiempo de obtención y descompresión de facturas pequeñas en Cassandra

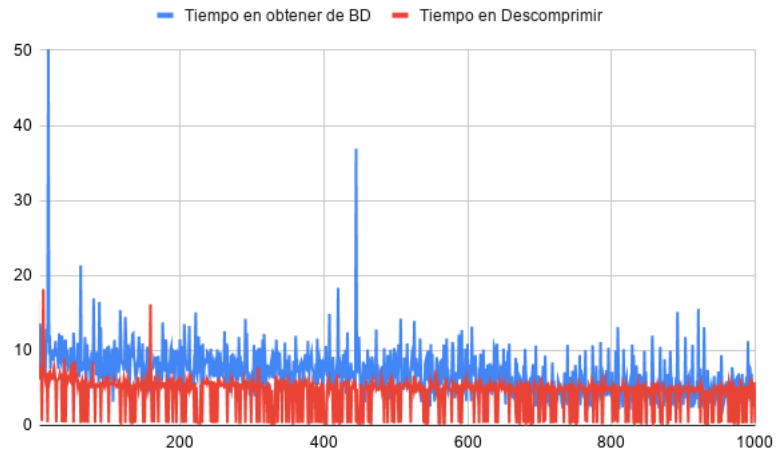

Figura 7.15: Tiempo de obtención y descompresión de facturas grandes en Cassandra

En las dos figuras anteriores se pueden ver los tiempos de obtención de factura de la Base

de Datos, es decir, el tiempo que tarda el SGBD en recuperar la factura solicitada, junto con el tiempo que tarda el sistema en descomprimirla.

El tiempo de búsqueda en la BD en Cassandra, para las facturas pequeñas, es algo más lento que en MongoDB, apenas tarda 2 milisegundos más. En cuanto al tiempo de descompresión ambos tiempos son muy similares, como es de esperar. En cuanto a las facturas grandes MongoDB es más rápido buscando que Cassandra. A pesar de que el tiempo medio de búsqueda de Cassandra no supera los 8 milisegundos MongoDb es capaz de realizar la búsqueda en menos de la mitad de tiempo. No obstante, no es una diferencia lo suficientemente significativa como para descartar el uso de uno de los sistemas.

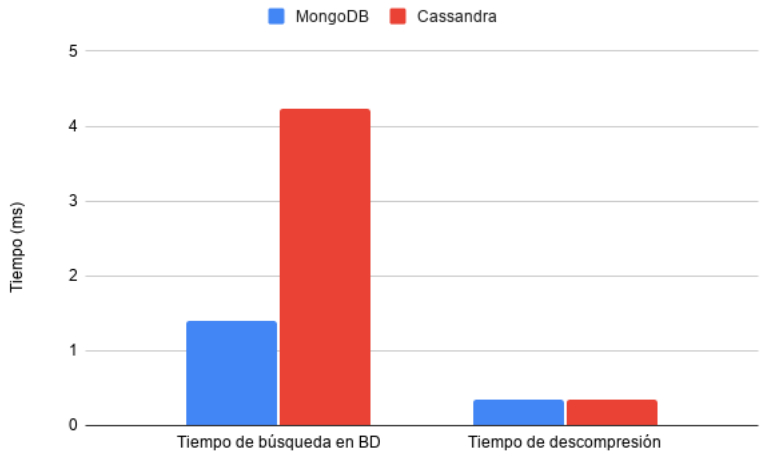

Figura 7.16: Tiempo medio de obtención de facturas pequeñas

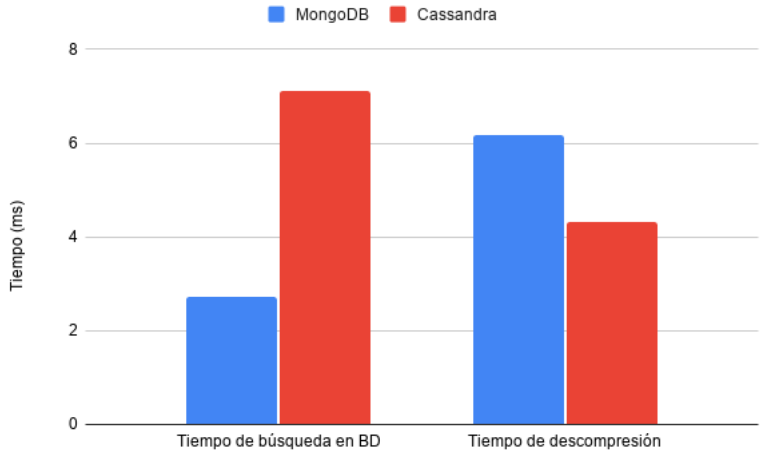

Figura 7.17: Tiempo medio de obtención de facturas grandes

Tiempo en recuperar un dato de una factura

El reducido tiempo de compresión y descompresión obtenido en las pruebas anteriores sugirió realizar la comparación entre la obtención de datos directamente de la base de datos, en RAW, frente a obtenerlos de la factura comprimida. Aunque el tiempo de descompresión sea muy reducido existe la posibilidad de perder una gran cantidad de tiempo buscando el dato pedido dentro de la propia factura.

Pruebas sobre MongoDB

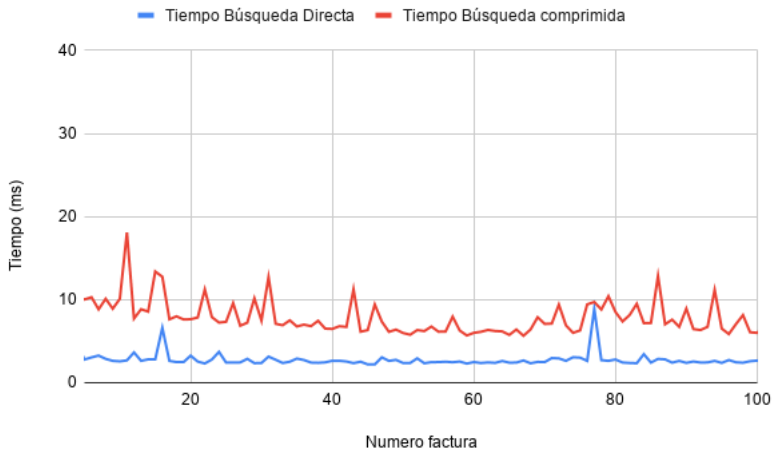

Figura 7.18: Tiempo de obtención de un dato concreto en facturas pequeñas

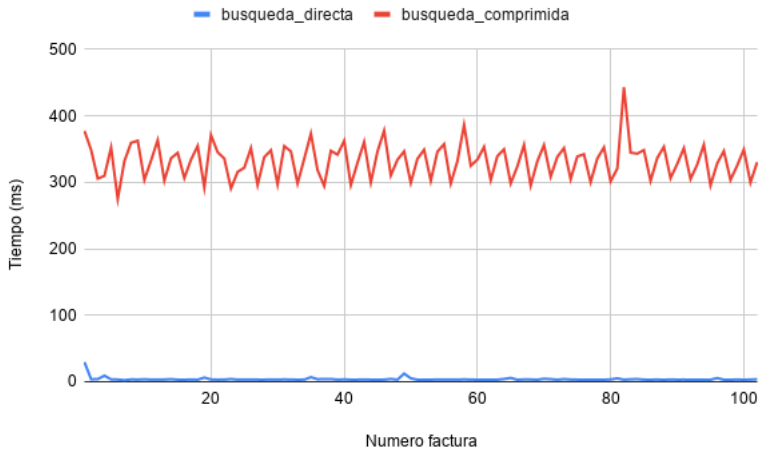

Figura 7.19: Tiempo de obtención de un dato concreto en facturas grandes

Los gráficos de las figuras 7.18 y 7.19 muestran la comparativa de tiempo de obtención de un dato, en este caso el campo *Importe Total Factura*, directamente de la base de datos, en *RAW*, frente a obtenerlo de la factura comprimida. En el primer caso simplemente se accede al documento correspondiente de la BD y se devuelve el dato. En el segundo caso, se accede al documento en cuestión y se devuelve la factura comprimida, la cual es descomprimida y accedida mediante XPath[1] para obtener el dato.

En la primera gráfica, figura 7.18, se muestra la comparativa para facturas pequeñas. Tal como se puede ver los tiempo son muy similares, apenas hay 10 milisegundos de diferencia entre ambos métodos de obtención del dato. No obstante, al analizar la gráfica de la figura 7.19 el resultado es diferente. El tiempo en obtener el dato directamente de la base de datos es similar al caso anterior, apenas supera los 10 milisegundos, sin embargo, al buscar el dato en la factura comprimida los tiempo se disparan a los 300 milisegundos.

Con estos datos, se puede concluir que es muy útil tener algunos datos en *RAW*, es decir, accesibles directamente en cada documento de la colección, para consultas que se ejecuten con gran frecuencia, este el caso de las consultas del QR. Los datos correspondientes a esta consulta estarían accesibles sin tener que realizar operaciones extra, y con un tiempo de respuesta muy bajo.

Pruebas sobre Cassandra

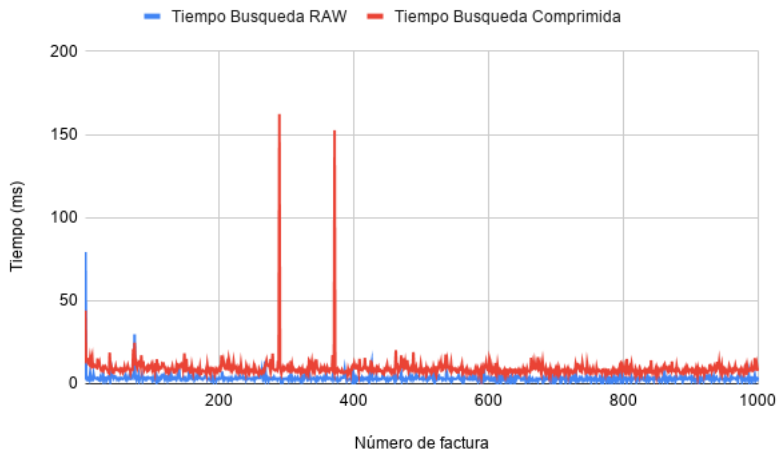

Figura 7.20: Tiempo de búsqueda en RAW frente a búsqueda en comprimido en Cassandra con facturas pequeñas

[1]Lenguaje que permite construir expresiones que recorren y procesan un documento XML

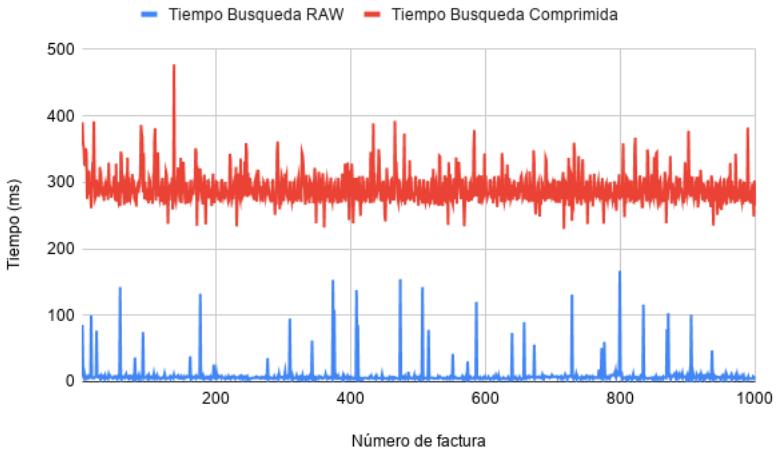

Figura 7.21: Tiempo de búsqueda en RAW frente a búsqueda en comprimido en Cassandra con facturas grandes

Las dos figuras anteriores muestran el resultado de comparar el tiempo de búsqueda de un dato en RAW frente a obtenerlo de la factura comprimida en Cassandra.

En términos generales, los resultados son similares a los obtenidos al realizar las pruebas en MongoDB. La diferencia entre obtener el dato en RAW y en comprimido para las facturas pequeñas es de unos 5 milisegundos de diferencia. En cuanto a las facturas grandes la diferencia es de unos 300 milisegundos, la misma que la ya vista en MongoDB.

En cuanto a la comparativa entre ambos SGBDs, en general, MongoDB ofrece unos mejores resultados, superando a Cassandra por unos pocos milisegundos de diferencia. No obstante, cuando se trata de devolver gran volumen de datos Cassandra, ver figura 7.24, obtiene un mejor rendimiento.

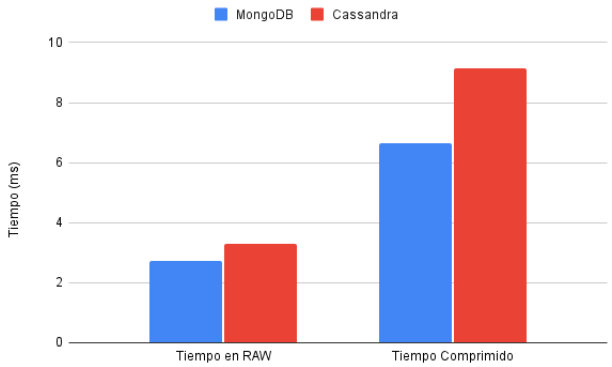

Figura 7.22: Comparativa obtención de dato concreto MongoDB y Cassandra con facturas pequeñas

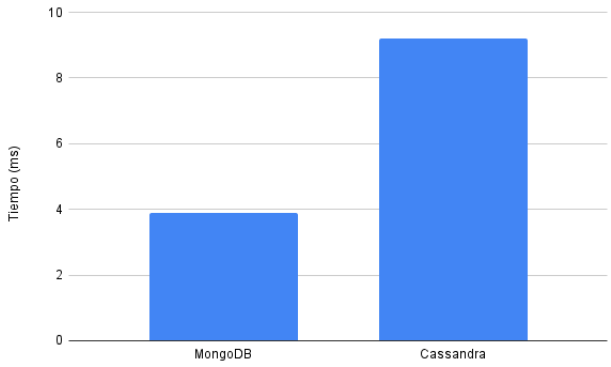

Figura 7.23: Comparativa obtención de dato concreto en RAW en MongoDB y Cassandra con facturas grandes

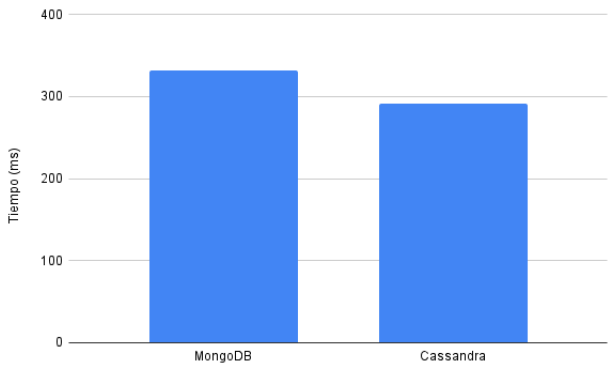

Figura 7.24: Comparativa obtención de dato concreto en comprimido en MongoDB y Cassandra con facturas grandes

7.1.3. Pruebas sobre facturas agrupadas

El objetivo de este conjunto de pruebas ha sido evaluar el rendimiento de las tablas o colecciones que contienen el conjunto de facturas agrupadas por NIF del emisor y rango de fechas de emisión. La meta de las pruebas ha sido también obtener el número óptimo de facturas agrupadas de cara a obtener un buen rendimiento y tiempos de respuesta.

Antes de comenzar, y tras realizar un nuevo análisis del diseño inicial de la colección de MongoDB enfocada a gestionar las facturas agrupadas, se ha decidido llevar a cabo la Corrección M.2.

Tiempo en recuperar una factura mediante el identificador TBAI

La búsqueda por identificador se realiza de la siguiente forma. Partiendo del identificador TBAI[2], se extraen el NIF y la fecha de expedición de la factura a la que corresponde el identificador. A continuación, se realiza la consulta, la cual se podría definir como "*Obtener las agrupaciones cuyo NIF se corresponde con el del identificador y cuya fecha de expedición de la factura se encuentra entre las fechas de la agrupación*" en cada uno de los SGBD.

Cabe destacar que la factura a recuperar en todas las pruebas siempre será la última de la agrupación, es decir, para una agrupación de 1000 facturas, la factura con la que se ha realizado la prueba de recuperación es la número 1000. De esta manera, los tiempos que se obtienen serán siempre el peor de los casos.

Pruebas sobre MongoDB

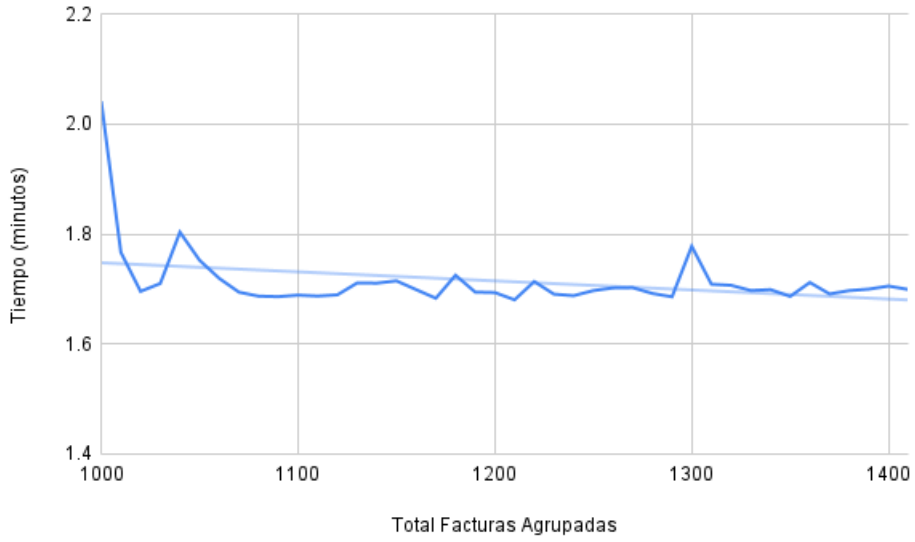

Figura 7.25: Tiempo de búsqueda en BD de la agrupación de facturas pequeñas

[2]TBAI-NNNNNNNNN-DDMMAA-FFFFFFFFFFFFF-CRC

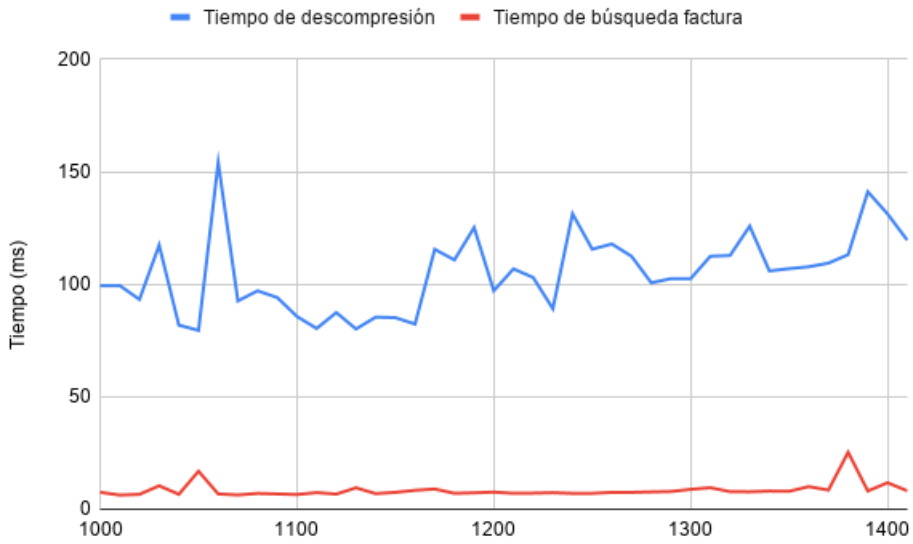

Figura 7.26: Tiempo de descompresión y búsqueda en la agrupación de facturas pequeñas

Tal como se aprecia en la figura 7.25, el tiempo de búsqueda en la base de datos no es muy óptimo. El tiempo medio de búsqueda de una agrupación es ligeramente inferior a los 2 minutos. Esto se debe a que en la búsqueda, ver listing B.2, se están comparando datos que no tienen índices aplicados sobre ellos.

Así mismo, si se comprueban los tiempos de búsqueda y recuperación de las agrupaciones de facturas grandes, el resultado que se obtiene es similar al de las facturas pequeñas.

Figura 7.27: Tiempo de búsqueda en BD de agrupaciones de facturas grandes

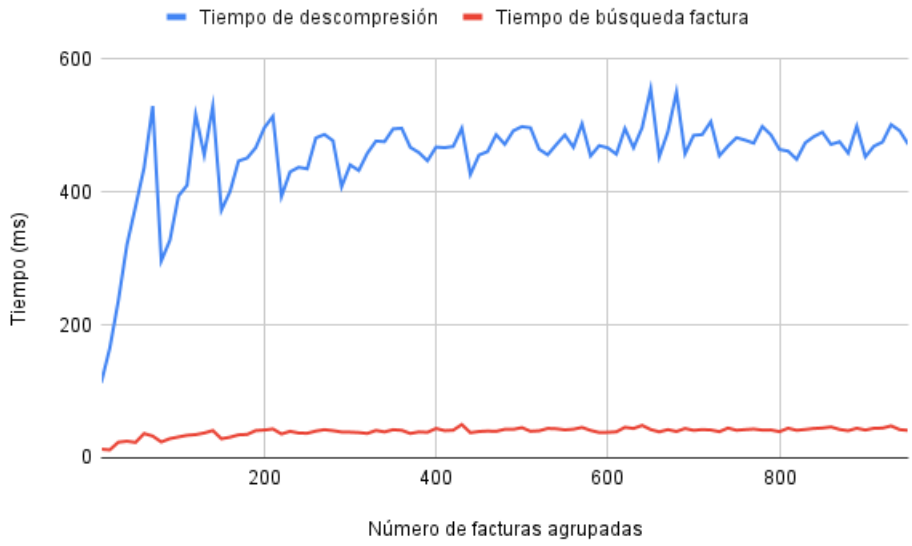

Figura 7.28: Tiempo de descompresión y búsqueda en la agrupación de facturas grandes

Tal como se adelantaba anteriormente, el motivo por el cual los tiempos de obtención de las agrupaciones es tan lento, es por la falta de índices. Por tanto, para solucionar el problema se ha llevado a cabo la Corrección M.3.

Pruebas sobre Cassandra

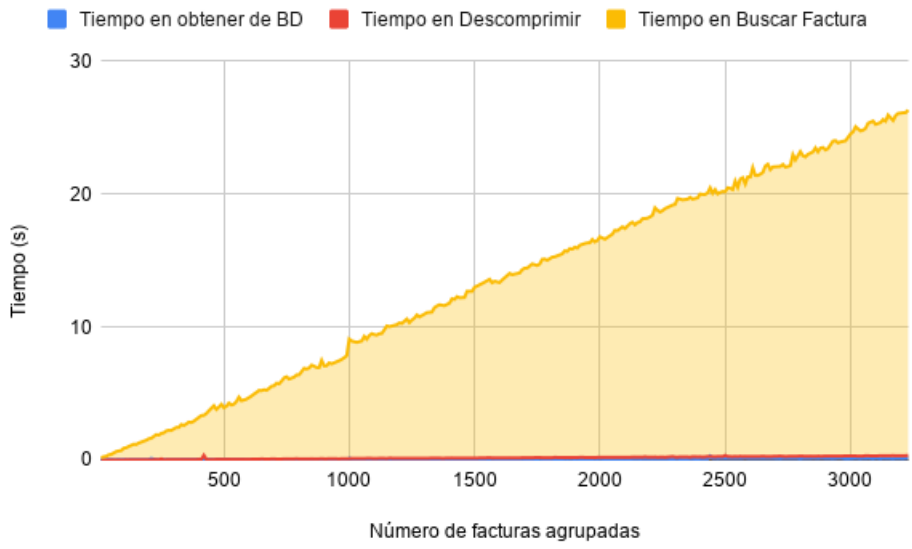

Figura 7.29: Tiempo de búsqueda de factura en agrupaciones pequeñas en Casssandra

Tal como se muestra en la figura anterior el tiempo de búsqueda es muy lento, para agrupaciones de 2000 facturas pequeñas el tiempo de búsqueda total es algo inferior a los 20 segundos.

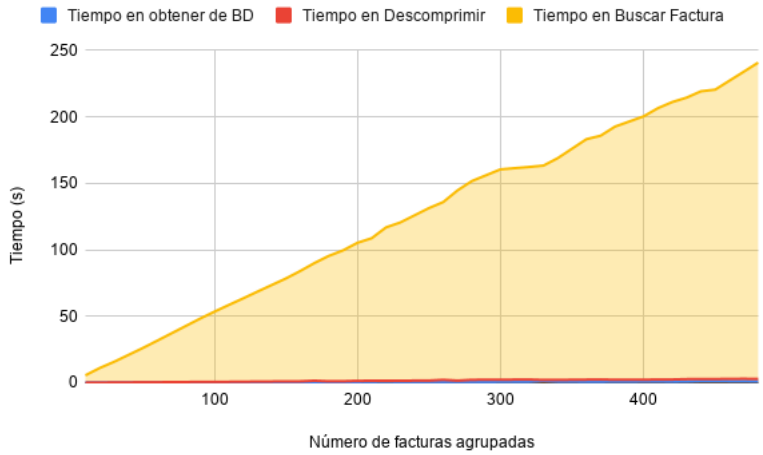

Figura 7.30: Tiempo de búsqueda de factura en agrupaciones grandes en Casssandra

En cuanto a las agrupaciones de facturas grandes los resultados son también muy lentos. Para agrupaciones de 200 facturas el tiempo de búsqueda dentro de la agrupación es de unos 100 segundos $\simeq$ 1,5 minutos.

El motivo de estos tiempos de respuesta es la forma de búsqueda de la factura concreta dentro de la agrupación. Una vez es descomprimida la agrupación correspondiente, se obtienen todas las facturas y se calcula el identificador TBAI de cada una de ellas hasta encontrar el que coincida con el solicitado. Dado que este método es muy ineficiente se ha decidido aplicar la Corrección C.1.

7.2. Correcciones en los diseños

7.2.1. Corrección M.1

Anteriormente se ha podido comprobar que guardar campos de la factura como los detalles en *RAW* en la base de datos puede no llegar a ser óptimo, en función de la cantidad de detalles que tenga la factura. No obstante, con el objetivo de aligerar los tiempos de inserción y recuperación de facturas, se ha eliminado el campo *DetallesFacturas* junto con su contenido, del esquema mostrado en el listing 6.3. Tras realizar el cambio, se ha

procedido a realizar unas nuevas pruebas para comprobar si efectivamente ha mejorado el rendimiento de la base de datos, siendo los resultados los que se muestran a continuación.

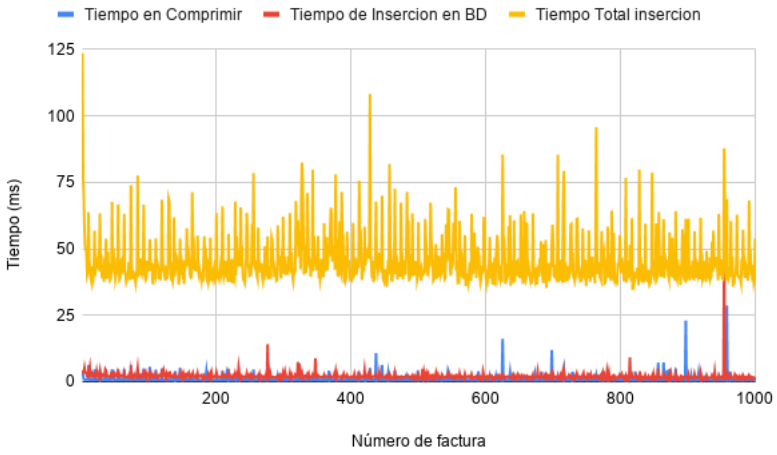

Figura 7.31: Tiempo total de inserción de facturas individuales pequeñas sin guardar los detalles en RAW

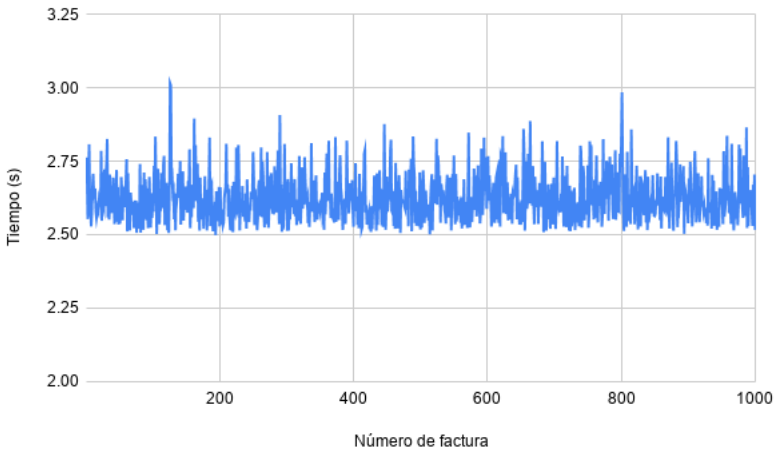

Figura 7.32: Tiempo total de creación e inserción de facturas individuales grandes sin guardar los detalles en RAW

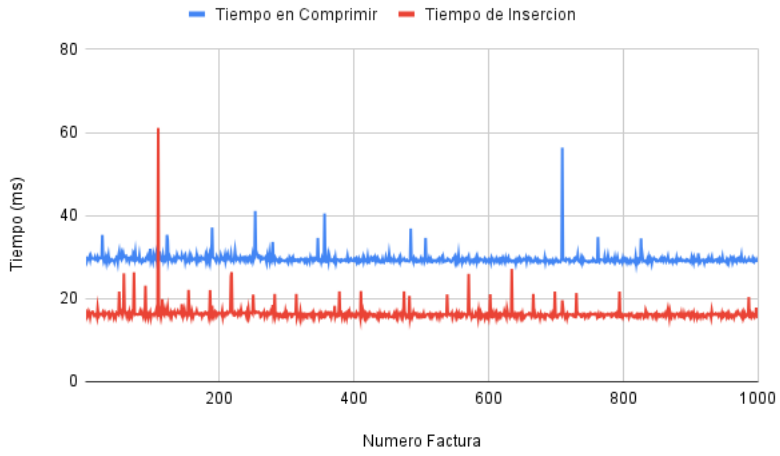

Figura 7.33: Tiempo de compresión en inserción en BD de las facturas grandes sin guardar los detalles en RAW

Si se comparan las dos figuras anteriores junto con las obtenidas con el diseño inicial, ver figuras 7.4 y 7.5, la mejora en el rendimiento ha sido notable. El tiempo de inserción de las facturas pequeñas se ha reducido a la mitad, unos 50 milisegundos. No obstante, en el caso de las facturas grandes, el tiempo total de inserción de facturas se ha reducido de 13 minutos a apenas 2 segundos, lo cual permite realizar una generación e inserción mucho más rápida de los datos.

Así mismo, en las figuras anteriores también se puede apreciar el tiempo de compresión e inserción de facturas en la base de datos. Salvo algunos picos en momentos concretos los tiempos resultantes son muy positivos, para las facturas pequeñas los tiempo de compresión e inserción en BD medios no superan los 10 milisegundos. En cuanto a las facturas grandes, los tiempos de compresión e inserción y compresión son algo más lentos, unos 20 milisegundos en insertar las facturas en la BD y 30 milisegundos en comprimirlas, unos resultados lógicos, ya que son facturas con mucho más contenido.

7.2.2. Corrección M.2

Inicialmente, el esquema de la colección que contiene el conjunto de facturas agrupadas contenía la forma mostrada en el listing 6.4, en el que el campo *_id* era de la forma "NIF/DD-MM-AAAA/DD-MM-AAAA", haciendo este campo único. No obstante, al realizar la inserción de datos de prueba se obtuvieron dos problemas. Por una parte, al estar

las fechas contenidas en un campo de texto, la búsqueda de las agrupaciones por fecha resultaba muy compleja de realizar.

El segundo problema estaba más relacionado con la propia configuración de MongoDB. Como ya se ha comentado anteriormente, MongoDB almacena los datos en documentos, que a su vez se almacenan en colecciones. No obstante, el tamaño de estos documentos tiene un límite, 16 MB [MongoDB, 2021b], por lo que es imposible insertar un documento que supere este tamaño.

Para evitar este problema y no tener complicaciones a la hora de insertar documentos, la solución empleada ha sido dividir las agrupaciones que superen este tamaño. Es decir, en caso de que una agrupación de facturas supere los 16MB, supongamos que ocupa 53 MB, se dividirá su contenido en 4 documentos diferentes ($\left\lceil \frac{53}{16} \right\rceil = 4$), pero todos representando el mismo NIF emisor y rango de fechas. Es por ello, que el campo *_id* dejaría de ser único, agregando de esta manera otro motivo más para modificar el esquema.

Por tanto, el nuevo esquema para las facturas agrupadas en MongoDB quedaría de la siguiente forma. Cabe destacar que el campo *_id* no se especifica, por lo que MongoDB creará uno al momento de la inserción, garantizando de esta manera la unicidad de este campo.

```
{
    nif: {
        type: Schema.Types.String,
        required: true
    },
    fechaInicio: {
        type: Schema.Types.Date,
        required: true
    },
    fechaFin: {
        type: Schema.Types.Date,
        required: true
    },
    idents: {
        type: Schema.Types.Array,
        required: true
    },
    agrupacion: {
        type: Schema.Types.String,
        required: true
    }
}
```

Listing 7.1: Modificación del esquema de las facturas en MongoDB

7.2.3. Corrección M.3

Tras analizar los lentos resultados obtenidos al buscar las agrupaciones de facturas se ha decidido crear un índice sobre los campos sobre los que se realiza la búsqueda de este conjunto de datos. Este índice se ha creado sobre los campos *nif* y *fechaInicio*, campos principales de búsqueda de las agrupaciones, ver listing B.1. Tras volver a repetir las consultas ejecutadas en 7.1.3, la mejora obtenida es muy buena.

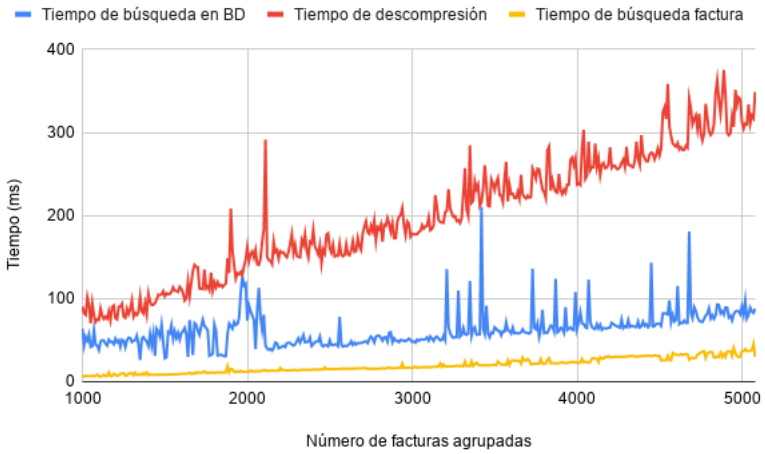

Figura 7.34: Desglose de tiempo de búsqueda de factura agrupada pequeña

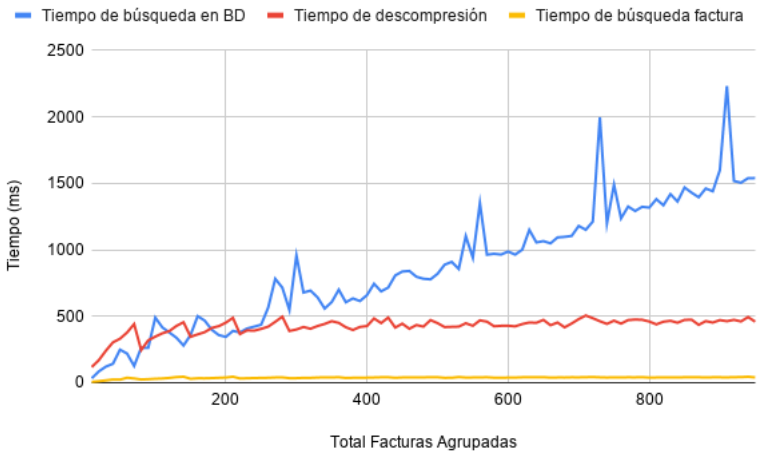

Figura 7.35: Desglose de tiempo de búsqueda de factura agrupada grande

Tal como se ve en las figuras anteriores, y como se comentaba antes, la mejora en el rendimiento gracias a los índices es notoria. Para agrupaciones de facturas pequeñas el

tiempo de búsqueda de la agrupación correspondiente se ha reducido de los 2 minutos de la figura 7.25 a apenas 100 milisegundos y creciendo muy poco a poco.

En el caso de las agrupaciones de facturas grandes los tiempos también se han reducido bastante respecto a los mostrados en la figura 7.27. No obstante, la curva del tiempo de búsqueda en la BD crece de forma mucho más pronunciada que con las facturas pequeñas. Esto se debe al problema mencionado en 7.2.2 relacionado con el tamaño de los documentos en MongoDB. A medida que aumenta el número de particiones también lo hace el tiempo de búsqueda, esto se debe a que a cada partición extra, el sistema tiene que devolver mayor volumen de datos. La siguiente figura muestra el aumento del tiempo de búsqueda medio en función del número de particiones en las que esta dividida la agrupación guardada.

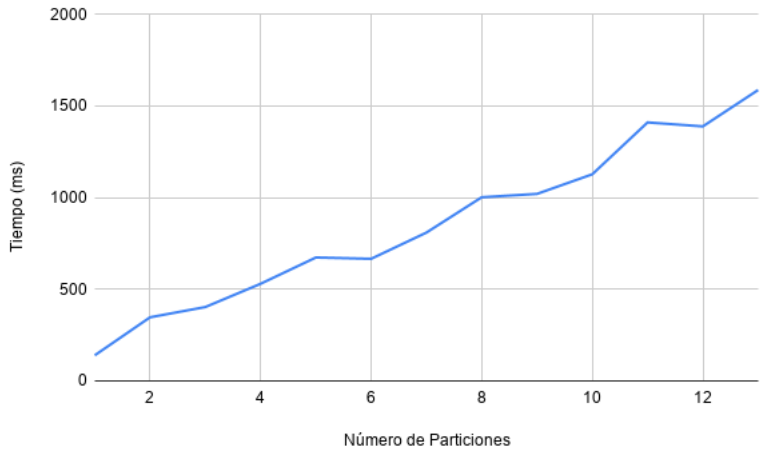

Figura 7.36: Tiempo medio de búsqueda por número de particiones de MongoDB

7.2.4. Corrección C.1

Anteriormente se ha podido comprobar que el método de búsqueda empleado en Cassandra no es nada óptimo, es por ello, que para realizar la búsqueda de las facturas se ha decidido emplear el mismo método que el utilizado en la BD de MongoDB. Aquí, los documentos de la colección que contiene las agrupaciones tienen un campo con los identificadores TBAI de las facturas que se están almacenando, de esta manera, no será necesario calcular el identificador TBAI de cada una de las facturas de la agrupación para despues compararlo, sino que se realizará la comparación dentro de la lista auxiliar ahorrando tiempo y mejorando considerablemente el rendimiento.

Partition Key		Clustering Key	Data		
Nif Emisor	Fecha Inicio	Fecha Inicio	Fecha Fin	Agrupación	Tbai_id_list

Tabla 7.1: Modificación de la tabla que contiene las agrupaciones de facturas

En cuanto a los nuevos resultados de las consultas los siguientes gráficos muestran la mejora obtenida tras realizar la corrección.

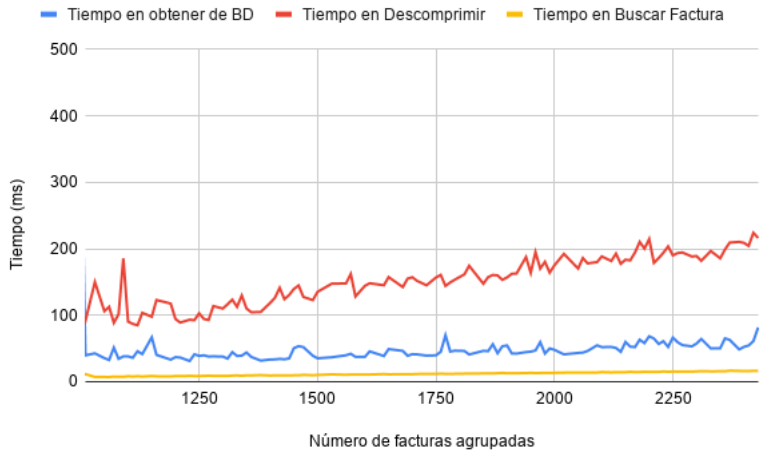

Figura 7.37: Tiempo de búsqueda de factura en agrupaciones pequeñas en Casssandra tras corrección

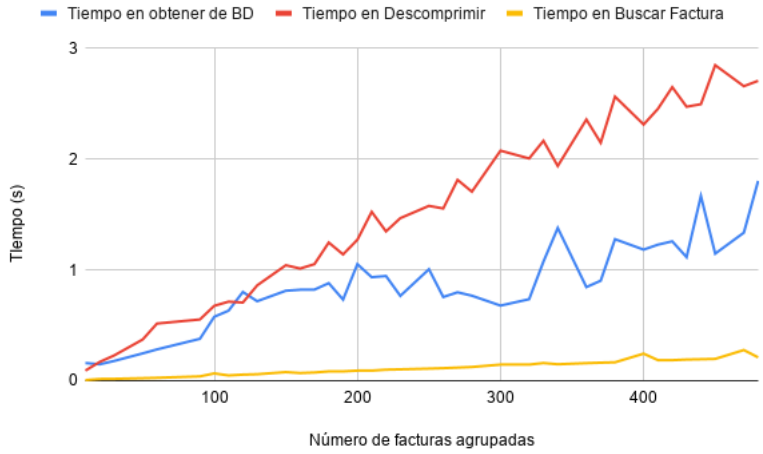

Figura 7.38: Tiempo de búsqueda de factura en agrupaciones grandes en Casssandra tras corrección

Tal como se ve en las figuras la mejora ha sido sustancial, mientras que los tiempos de obtención de la Base de Datos y de descompresión se han mantenido, el tiempo

de búsqueda dentro de la agrupación se ha reducido considerablemente. El tiempo de búsqueda en agrupaciones de facturas pequeñas y grandes es de 150 milisegundos de media, y aumentando muy lentamente. A destacar, los tiempos de las facturas grandes, en los que el tiempo de descompresión de los datos es superior al de búsqueda en la Base de Datos.

Si comparamos estos resultados en ambos sistemas, el resultado es el siguiente.

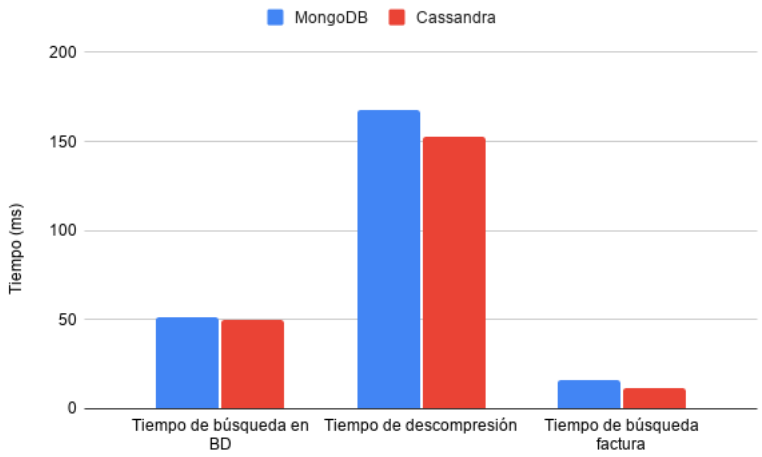

Figura 7.39: Comparación de búsqueda de facturas agrupadas pequeñas entre MongoDB y Cassandra

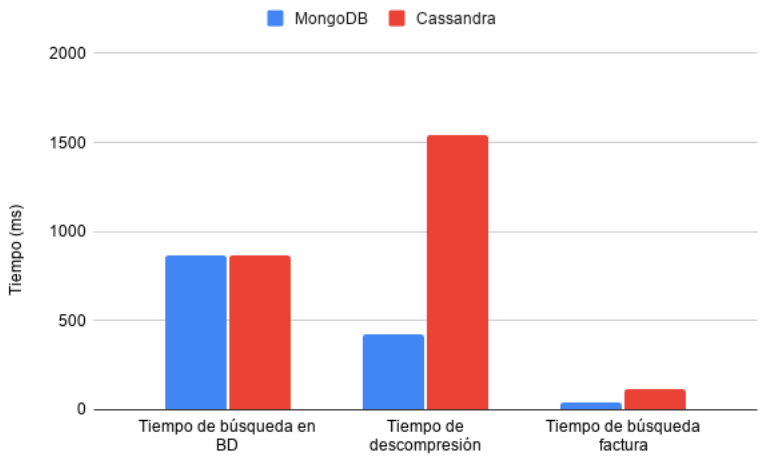

Figura 7.40: Comparación de búsqueda de facturas agrupadas grandes entre MongoDB y Cassandra

Tal como se ve en las figuras, para la búsqueda de facturas pequeñas en ambos sistemas los tiempos son similares, Cassandra es unos pocos milisegundos más rápido. No obstante,

al realizar la búsqueda en agrupaciones de facturas grandes los resultados son diferentes. A pesar de que el tiempo de búsqueda en la Base de Datos es similar en ambos sistemas, los tiempos de descompresión y búsqueda de facturas son mucho más altos en Cassandra. Esto se debe a que en MongoDB, cuando se superaba el límite de 16MB de tamaño de agrupación esta se dividía en diferentes particiones, es por ello, que al descomprimir y buscar, el sistema deberá hacerlos en particiones mucho más pequeñas, mientras que Cassandra deberá buscar en la agrupación completa que tiene mayor volumen de datos.

7.2.5. Corrección C.2

Tras realizar todas las pruebas relacionadas con la obtención de datos directamente de la factura comprimida se llegó a la conclusión de que separar las tablas 6.1 y 6.2 suponía una división innecesaria. Debido a la no tan frecuente consulta de datos relacionados con el QR y el identificador TBAI, los demás datos a consultar pueden ser obtenidos de la factura comprimida sin perder mucho rendimiento. Por ello, las tablas anteriores quedarían unidas de la siguiente manera.

Partition Key		**Clustering Key**	**Data**			
Nif	Fecha	Ident TBAI	Importe Total	Número Factura	Número Factura	xml

Tabla 7.2: Nueva tabla que sustituye a las tablas no relacionadas con las agrupaciones de las facturas

De esta manera, el esquema en MongoDB y Cassandra sería muy similar, cada una de las Bases de Datos distribuirá sus datos en dos tablas o colecciones, una para las consultas sobre facturas unitarias recientes y otra para facturas guardadas en una agrupación de varias facturas.

8. CAPÍTULO

Ondorioak/conclusiones

Anexos

A. ANEXO

Sentencias CQL de Cassandra

A.1. Creación del Keyspace

```
1 CREATE KEYSPACE ticketbai WITH REPLICATION = {'class': 'SimpleStrategy', 'replication_factor' :
      1};
```

Listing A.1: Creación del Keyspace de Cassandra

Dado que el sistema estará formado por una única máquina, para realizar todas las pruebas se ha utilizado un una estrategia simple (*SimpleStrategy*), la cual prepara el *Keyspace* para utilizar un único nodo y con un factor de replicación de 1. Como ya se comentaba esta estrategia esta pensada para realizar diseños "simples", para sacar el máximo rendimiento del SGBD se utiliza la estrategia *NetworkTopolyStrategy*, la cual está pensada para definir varios centros de datos, cada uno con un factor de replicación diferente, esta es la estrategia más utilizada y extendida.

A.2. Sentencias de creación de las tablas

A continuación se muestran las sentencias empleadas para generar las tablas finales, es decir, las tablas utilizadas en cassandra junto con las correciones aplicadas en 7.2.

```
CREATE TABLE IF NOT EXISTS facturas (nif text, fecha date, tbai_id text, importe double,
    num_factura int, serie text, xml text, PRIMARY KEY ((nif, fecha), tbai_id)) WITH CLUSTERING
    ORDER BY (tbai_id ASC);
```

Listing A.2: Creación de la tabla facturas

```
CREATE TABLE IF NOT EXISTS facturas_agrupadas (nif text, fecha_inicio date, agrupacion text,
    fecha_fin date, tbai_id_list list<text>, PRIMARY KEY (nif, fecha_inicio)) WITH CLUSTERING
    ORDER BY (fecha_inicio ASC);
```

Listing A.3: Creación de la tabla facturas_agrupadas

B. ANEXO

Sentencias y consultas de MongoDB

B.1. Creación de colecciones e indices

```
db.facturas_agrupadas.createIndex({"nif": 1, "fechaInicio": 1}, {name: "nifFechaIndex"});
```

Listing B.1: Creación de índice para las facturas agrupadas

B.2. Consultas sobre las colecciones

```
db.facturas_agrupadas.find(
{
    nif: nif,
    fechaInicio: {$lte: fecha},
    fechaFin: {$gte: fecha}
});
```

Listing B.2: Consulta de obtención de facturas agrupadas

Bibliografía

[Acc, 2016] (2016). Accept-encoding: Br on HTTPS connection - Chrome Platform Status. https://www.chromestatus.com/feature/5420797577396224. [En línea; consultado el 16-febrero-2021].

[Fir, 2016] (2016). Firefox 44.0, See All New Features, Updates and Fixes. https://www.mozilla.org/en-US/firefox/44.0/releasenotes/. [En línea; consultado el 16-febrero-2021].

[Administración Electrónica, 2021] Administración Electrónica (2021). Módulo de firmas xml. https://administracionelectronica.gob.es/ctt/resources/Soluciones/138/Descargas/CFv3-4-Manual-de-firmas-XML.pdf?idIniciativa=138&idElemento=4147. [En línea; consultado el 11-abril-2021].

[Alakuijala and Szabadka, 2016] Alakuijala, J. and Szabadka, Z. (2016). Brotli Compressed Data Format. (RFC 7932). [En línea; consultado el 16-febrero-2021].

[Apache, 2016] Apache, C. (2016). Cassandra Documentation. https://cassandra.apache.org/doc/latest/operating/compression.html. [En línea; consultado el 18-febrero-2021].

[ASALE and RAE, 2021] ASALE, R. and RAE (2021). Comprimir | Diccionario de la lengua española. https://dle.rae.es/comprimir. [En línea; consultado el 13-febrero-2021].

[Budhrani, 2019] Budhrani, D. (2019). How data compression works: Exploring LZ77. https://towardsdatascience.com/how-data-compression-works-exploring-lz77-3a2c2e06c097. [En línea; consultado el 14-febrero-2021].

[Codd, 2002] Codd, E. F. (2002). A relational model of data for large shared data banks. In Broy, M. and Denert, E., editors, *Software Pioneers: Contributions to Software Engineering*, pages 263–294. Springer Berlin Heidelberg, Berlin, Heidelberg. [En línea; consultado el 07-marzo-2021].

[Collet, 2016] Collet, Y. (2016). Facebook/zstd. https://github.com/facebook/zstd. [En línea; consultado el 18-febrero-2021].

[Collet, 2020] Collet, Y. (2020). Lz4/lz4. https://github.com/lz4/lz4. [En línea; consultado el 18-febrero-2021].

[DataFlair, 2018] DataFlair (2018). 5 Important Cassandra Features That You Must Know. https://data-flair.training/blogs/cassandra-features/. [En línea; consultado el 07-mayo-2021].

[DataStax, 2021] DataStax (2021). How is the consistency level configured? | Apache Cassandra 3.0. https://docs.datastax.com/en/cassandra-oss/3.0/cassandra/dml/dmlConfigConsistency.html. [En línea; consultado el 07-mayo-2021].

[Dave, 2012] Dave, M. (2012). SQL and NoSQL Databases. *International Journal of Advanced Research in Computer Science and Software Engineering*. [En línea; consultado el 08-marzo-2021].

[Deutsch and Katz, 1996] Deutsch, P. and Katz, P. (1996). DEFLATE Compressed Data Format Specification version 1.3. Technical Report RFC1951, RFC Editor. [En línea; consultado el 20-febrero-2021].

[Dipperstein, 2019] Dipperstein, M. (2019). LZSS (LZ77) Discussion and Implementation. https://michaeldipperstein.github.io/lzss.html. [En línea; consultado el 15-febrero-2021].

[Diputación Foral, 2021] Diputación Foral, G. (2021). Documentación y normativa - Ogasuna. https://www.gipuzkoa.eus/es/web/ogasuna/ticketbai/documentacion-y-normativa. [En línea; consultado el 22-marzo-2021].

[Gailly and Adler, 2002] Gailly, J.-L. and Adler, M. (2002). Zlib home site. http://zlib.net/apps.html. [En línea; consultado el 18-febrero-2021].

[Gally and Adler, 2002] Gally, J.-L. and Adler, M. (2002). An Explanation of the 'Deflate' Algorithm. https://zlib.net/feldspar.html. [En línea; consultado el 18-febrero-2021].

[GeeksforGeeks, 2012] GeeksforGeeks (2012). Huffman Coding | Greedy Algo-3. https://www.geeksforgeeks.org/huffman-coding-greedy-algo-3/. [En línea; consultado el 21-febrero-2021].

[Google, 2021] Google, G. (2021). Google/snappy. https://github.com/google/snappy. [En línea; consultado el 16-febrero-2021].

[Gupta et al., 2017] Gupta, A., Bansal, A., and Khanduja, V. (2017). Modern lossless compression techniques: Review, comparison and analysis. [En línea; consultado el 10-febrero-2021].

[Guru99, 2021] Guru99 (2021). Performance Testing Tutorial: What is, Types, Metrics & Example. https://www.guru99.com/performance-testing.html. [En línea; consultado el 12-mayo-2021].

[Hosseini, 2012] Hosseini, M. (2012). *A Survey of Data Compression Algorithms and Their Applications*. [En línea; consultado el 10-febrero-2021].

[loup Gailly and Adler, 2018] loup Gailly, J. and Adler, M. (2018). GNU Gzip. https://www.gnu.org/software/gzip/manual/gzip.html. [En línea; consultado el 13-febrero-2021].

[MongoDB, 2021a] MongoDB (2021a). Advantages of MongoDB. https://www.mongodb.com/advantages-of-mongodb. [En línea; consultado el 11-abril-2021].

[MongoDB, 2021b] MongoDB (2021b). Documents — MongoDB Manual. https://docs.mongodb.com/manual/core/document/#:~:text=Document%20Size%20Limit,MongoDB%20provides%20the%20GridFS%20API. [En línea; consultado el 10-mayo-2021].

[Palaniappan and Latifi, 2007] Palaniappan, V. and Latifi, S. (2007). Lossy text compression techniques. In Akhgar, B., editor, *ICCS 2007*, pages 205–210, London. Springer London. [En línea; consultado el 23-febrero-2021].

[Parlamento Europeo, 2014] Parlamento Europeo (2014). Reglamento (UE) n ° 910/2014 del Parlamento Europeo y del Consejo, de 23 de julio de 2014 , relativo a la identificación electrónica y los servicios de confianza para las transacciones electrónicas en el mercado interior y por la que se deroga la Directiva 1999/93/CE. [En línea; consultado el 11-abril-2021].

[Pavlov, 2011] Pavlov, I. (2011). 7-Zip / Discussion / Open Discussion: LZMA vs. LZMA2. https://sourceforge.net/p/sevenzip/discussion/45797/thread/2f6085ba/. [En línea; consultado el 24-febrero-2021].

[Schaefer, 2021] Schaefer, L. (2021). NoSQL vs SQL Databases. https://www.mongodb.com/nosql-explained/nosql-vs-sql. [En línea; consultado el 04-marzo-2021].

[W3C, 2013] W3C (2013). XML Signature Syntax and Processing Version 1.1. https://www.w3.org/TR/xmldsig-core1/. [En línea; consultado el 11-abril-2021].

[W3Schools, 2021] W3Schools (2021). SQL Introduction. https://www.w3schools.com/sql/sql_intro.asp. [En línea; consultado el 07-marzo-2021].

[Wallace, 1992] Wallace, G. K. (1992). The jpeg still picture compression standard. *IEEE Transactions on Consumer Electronics*, 38(1):xviii–xxxiv. [En línea; consultado el 18-febrero-2021].

[Wikipedia, 2021] Wikipedia (2021). Lempel–Ziv–Markov chain algorithm. *Wikipedia*. [En línea; consultado el 24-febrero-2021].