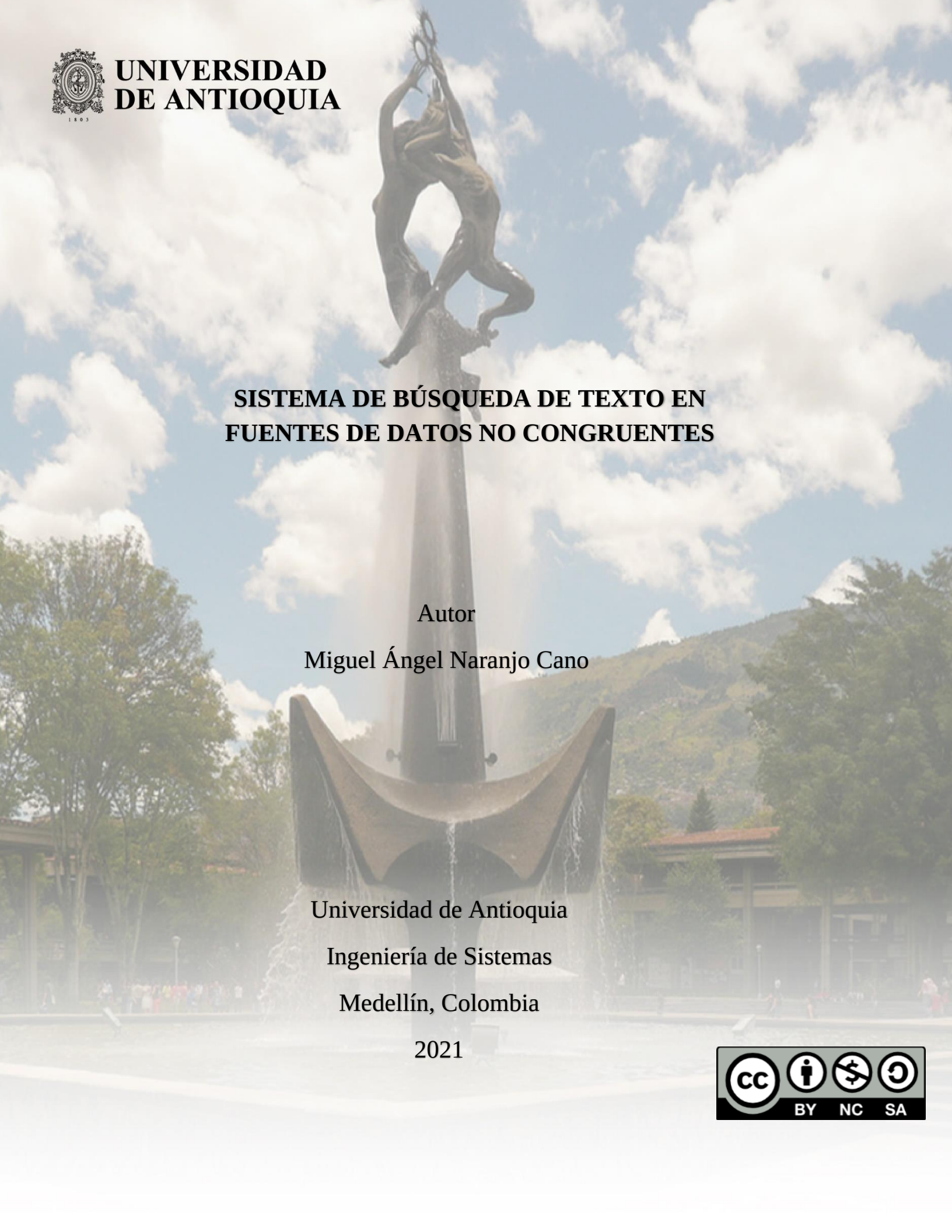

UNIVERSIDAD DE ANTIOQUIA

SISTEMA DE BÚSQUEDA DE TEXTO EN FUENTES DE DATOS NO CONGRUENTES

Autor

Miguel Ángel Naranjo Cano

Universidad de Antioquia

Ingeniería de Sistemas

Medellín, Colombia

2021

SISTEMA DE BÚSQUEDA DE TEXTO EN FUENTES DE DATOS NO CONGRUENTES

Miguel Ángel Naranjo Cano

Tesis o trabajo de investigación presentada(o) como requisito parcial para optar al título de:

Ingeniero de Sistemas

Asesores:

Deisy Loaiza Berrío - Coordinadora de Prácticas académicas

Camilo Úsuga Ortiz – Asesor Externo Talos Digital

Línea de Investigación:

Búsqueda y Recuperación de Información

Universidad de Antioquia

Ingeniería de Sistemas

Medellín, Colombia

2021.

Tabla de contenido

Resumen....3
Introducción....3
Objetivos....3
Objetivo general....3
Objetivos específicos....3
Marco Teórico....4
Ecuación N° 1....6
Metodología....7
1. Elicitación de requisitos:....7
a. Identificación problema....7
b. Definición de la estructura de datos....7
c. Determinar requisitos funcionales....7
d. Identificar requisitos no funcionales....7
e. Limitación de intervalos para los requisitos no funcionales....7
2. Caracterización tecnológica....7
a. Búsqueda de sistemas de indexación disponibles....7
b. Realizar compendio de características de los sistemas de búsqueda....8
c. Filtrar sistemas de búsqueda según requisitos y estado de la tecnología....8
3. Métricas de prueba....8
a. Determinar métricas de medición en sistemas de búsqueda....8
b. Definir la validez de las métricas para el sistema de búsqueda....8
c. Implementación de pruebas para el sistema de búsqueda....8
d. Probar sistema de búsqueda encontrado....8
e. Determinar sistema de búsqueda a implementar....8
4. Implementación Sistema....8
a. Identificar metodologías de distribución de carga para el sistema de búsqueda. 8
b. Levantamiento sistema de búsqueda....8
c. Integración con aplicativo....8
d. Pruebas a la implementación del sistema....8
e. Parametrización de efectos....8
f. Ajuste sistema búsqueda requisitos....9
g. Optimización rendimiento sistema de búsqueda....9
Resultados y análisis....9

1. Elicitación de requisitos: 9
Requisitos funcionales: 9
Requisitos no funcionales: 9
Intervalos considerados para los requisitos no funcionales del proceso de búsqueda: 10
2. Caracterización tecnológica: 10
3. Métricas de prueba: 11
Tabla N° 1 13
4. Implementación del Sistema: 13
Figura N° 1 15
Fragmento de código N° 1 16
Tabla N° 2 17
Fragmento de código N° 2 17
Tabla N° 3 17
Tabla N° 4 18
Tabla N° 5 18
Conclusiones 19
Referencias Bibliográficas 20

Sistema de búsqueda de texto en fuentes de datos no congruentes

Resumen

En el presente trabajo se buscó detallar el proceso de construcción y selección de un sistema de búsqueda para una fuente de datos con tipologías diferentes. Se comenzó recolectando los requisitos del sistema a lo que aproximo unos intervalos establecidos según la capacidad y alcance del sistema deseado en términos de escalabilidad, disponibilidad y rendimiento. A partir de estos requisitos se decantó una lista de posibles sistemas que cumplieran las expectativas deseadas. De esta lista se filtró al sistema de búsqueda de Elasticsearch con el cual se realizaron pruebas de rendimiento con una base de datos similar en naturaleza a la de producción. Se hizo una implementación de este sistema, definiendo su diseño distribuido y se optimizaron las peticiones para ser más adecuadas a los intervalos de los requisitos definidos de manera previa.

Introducción

En la actualidad existen sistemas de búsqueda ya preestablecidos que permiten un alto rendimiento en las consultas debido a las diferentes estructuras y algoritmos de los que se componen [1]. La delicadeza de estos sistemas radica en la configuración adecuada para una implementación que cumpla con las expectativas del cliente. Los requisitos que son definidos durante el problema pueden establecer cuál sistema de búsqueda usar y esto puede traer consigo problemas específicos de implementación, como la capacidad de escalar que exige una naturaleza distribuida, esta característica hace que el proceso de indexación de la información dispuesta sea necesario y requiera de un tiempo considerable [2]. El complejo del sistema a tratar consiste en fuentes de datos con tipologías diversas y requisitos enfocados a la priorización y subrayados de algunos de estos datos. El propósito es que, a partir de esta fuente, se pueda generar un resultado de búsqueda abarcado dentro de los requisitos interpuestos por el cliente.

Objetivos

Objetivo general

- Implementar sistema de búsqueda con alto rendimiento para fuentes de datos no congruentes.

Objetivos específicos

- Establecer valores deseados para los requisitos funcionales y no funcionales del sistema de búsqueda.
- Caracterizar los sistemas de búsqueda disponible basados en los requisitos.
- Probar sistemas de búsqueda filtrados de acuerdo con los requisitos.
- Integrar el sistema de búsqueda optimizado mediante parametrizaciones y pruebas.

Marco Teórico

El proceso de recuperar información lo más rápido posible de memoria ha sido un problema que los sistemas de información han enfrentado, varía según la necesidad y la naturaleza de los datos almacenados [3]. Algunos sistemas buscan únicamente en datos numéricos otros se encargan de búsqueda de información alfanumérica, si bien dependiendo del algoritmo y estructura usada la cota superior asintótica puede variar y ser incluso 1 en ciertas ocasiones, este no es siempre el caso para los sistemas de búsqueda [2]. Esto se debe a que las búsquedas son sobre datos que normalmente no se encuentran estructurados de una manera consistente o la naturaleza de la búsqueda no es común para definir un orden [3].

Los sistemas de búsqueda presentan varios problemas relacionados con los requisitos no funcionales. Como son:

- Efectividad [1] [4]
- Rendimiento [2]
- Disponibilidad [5]
- Escalabilidad [6]
- Resiliencia [1]

Cada uno de estos problemas es abordado de diversas formas según la naturaleza de la información que se almacena y las características de la búsqueda que sean deseadas. Si los datos almacenados tienden a cambiar con frecuencia (dinámicos) o si continúan siendo los mismos a través del tiempo de existencia de la información [3].

En el caso de los sistemas de búsqueda para texto es necesario recurrir a metodologías diferentes a la tradicionales, el texto se puede buscar por palabras, por oraciones o por frases, lo que haría que los métodos tradicionales de consulta no pudieran cumplir las expectativas requeridas, por ejemplo:

Si se tiene una columna en una base de datos relacional con la siguiente información:

titulo
Manzana roja una aventura
Arándanos morados
Rojas moras

Deseamos realizar una búsqueda en esta columna de los títulos que empiezan por rojo, para eso se usa la siguiente sentencia en SQL para realizar el proceso de búsqueda:

SELECT titulo FROM tabla WHERE titulo LIKE ‘rojo%’;

Esta sentencia representa los títulos que comienzan con la palabra rojo pero si miramos con detalle ningún título es extraído. Esto no es muy satisfactorio para el usuario que deseaba saber que títulos poseían el color rojo como primera palabra independiente de la cantidad a la que se refiriera. Para arreglar este problema se puede establecer una sentencia como la siguiente:

SELECT titulo FROM tabla WHERE titulo LIKE 'roj%';

Esta query extraería todos los títulos que empezaran por roj lo que abarcaría rojo, roja y rojas, pero cuando ejecutamos la sentencia nos encontramos con que el resultado sigue siendo vacío. Esto se debe a que la mayúscula de la palabra Rojas no está abarcado por la query. Si el usuario desea que la palabra rojo este en el título o que se pueda buscar "roja aventura" para el primer título esto requerirá de unas modificaciones en la query adicionales para generar estos filtrados. Un problema que surge es que al aumentar la cantidad de registros esta búsqueda se hace más demandante para la base de datos lo que impide que el sistema cumpla otras peticiones que se están exigiendo.

Es por este motivo que se usan sistemas de indexación para realizar procesos de recopilación de la información y facilitar el proceso de búsqueda [2]. Estos sistemas de indexación están basados en múltiples estructuras de datos con el objetivo de realizar una búsqueda más dinámica y permitir la adaptabilidad del sistema a la búsqueda de información.

Los sistemas de búsqueda son diversos pero tienen una base en común que consiste en una serie de pasos específicos para su funcionamiento, primero se realiza una recepción de los documentos donde se toman los documentos y se transforman en el formato deseado, posteriormente se toman los documentos y se les realiza un proceso de tokenización donde se divide en palabras las frases y se eliminan algunos elementos que pueden ser considerados no aportantes al proceso de búsqueda (Estos elementos pueden ser conjunciones, preposiciones y puntuación) Además algunos tokenizadores son capaces de identificar conjugaciones y reducir la palabra a su mínima expresión (raíz) [2]. Posteriormente se hace un proceso de indexación donde se asocia la palabra a el documento donde se encuentra referido, se denomina índice inverso a esta estructura por esta característica ya que la llave del índice son las palabras que conforman el documento o página contrario a un índice donde el documento es el regente director hacia el conjunto de palabras [7]. Es de esta forma que se estructura principal de los sistemas de búsqueda se conforma, Algunos de los sistemas de búsqueda que funcionan con esta base son Lucene, Rucene, Tantivy, Meilisearch y sistemas de búsqueda generados desde una base de datos relacional como los que expone Sumukh Bhandarkar [2].

Estos sistemas de búsqueda cumplen con los requerimientos básicos de búsqueda de palabras y son capaces de realizar procesos como autocompletado y corrección de errores gramaticales a través de estructuras como los autómatas de Levenshtein [8]. Pero incluso después de hacer el funcionamiento de la búsqueda más eficiente hay un problema que sufren estos sistemas. Un sistema de índices puede evidenciar un valor de repeticiones de cierto dato en una estructura, pero su relevancia puede no estar asociada directamente a este valor. Esto se denomina en campo de la recuperación de información como la efectividad del sistema [4]. Diversos sistemas de búsqueda usan un método de cálculo para encontrar la relevancia de un documento, este consiste en

enumerar las repeticiones de los términos en cada uno de los documentos y dividir este índice por el total de instancias de estos términos presentes en todos los documentos.

Frecuencia de términos (TF): Número de repeticiones de un término en un documento [9].

Frecuencia de documento inversa (IDF): Es uno divido el número de repeticiones presentes de un término en todos los documentos [9].

Al final la formula queda:

$$Puntaje = TF \times IDF$$

Ecuación N° 1

En los sistemas de recuperación de información como los sistemas de búsqueda, se incluyen dos propiedades esenciales que son la mejora de la calidad de los resultados (Que se aproxima a partir de la formula anterior) y la mejora de la escalabilidad y eficiencia de estos sistemas [4]. Para cumplir el requisito de escalabilidad, los sistemas de búsqueda deben ser distribuidos. Es de esta manera que servidores de búsqueda como Elasticsearch dividen los índices (En este caso administrados por Lucene) como agrupaciones de fragmentos que permiten la deposición en diferentes dispositivos [6]. Esto trae retos relevantes como encontrar un patrón de partición de los fragmentos que facilite la identificación del fragmento particular que posee la información y que distribuya la carga de manera eficiente evitando la replicación de los datos de manera efectiva.

En el caso de Elasticsearch el motor de búsqueda provee una selección basada en las siguientes hipótesis [6].

- Una colección de documentos puede ser particionada para que la mayoría de los documentos relevantes para una consulta estén guardados en algunos fragmentos sin previo conocimiento del conjunto de peticiones.
- Cuando los datos son particionados en diferentes fragmentos, estos pueden ser catalogados basados en su relevancia para la petición.
- El número mínimo de fragmentos superiores que debe ser buscado puede ser estimado.

Este motor funciona como una base de datos que está construido para manejar altos volúmenes de información con una gran disponibilidad. Su capacidad radica en el manejo distribuido de la información con una abstracción de una API [1].

Es bajo estas bases que se han hecho labores investigativas como las propuestas por Praveen M Dhulavvagol el cual realiza un proceso de mejora en este sistema de búsqueda a partir de técnicas de división de los fragmentos, donde se evita el contenido aleatorio de esta distribución y se ejecutan algoritmos de selección de estos cuando se realizan procesos de búsqueda como Redde, Rank-S y CORI [6]. Estos procedimientos

permitieron aumentar el desempeño del sistema en los procesos de búsqueda hasta un 28% con el algoritmo Rank-S [6].

La gran cantidad de información que manejan los sistemas de indexación obliga a tener nuevas estrategias para el manejo de la información es por lo que estos sistemas deben recurrir a modelos de compresión recurrente que permitan constantemente reducir los tamaños de los índices [10]. Es de esta manera como Elasticsearch maneja el sistema de indexación y es capaz de manejar altos volúmenes de datos sin sacrificar el espacio de disco de manera extensa.

Es de esta forma que en el trabajo actual se pretende abarcar las configuraciones necesarias para el sistema de búsqueda, con el objetivo de adecuarlo de forma óptima a los requisitos no funcionales impuestos.

Metodología

1. Elicitación de requisitos:

A través de una comunicación efectiva se establecen requisitos funcionales y no funcionales que definirán al sistema.

a. Identificación problema.

Consiste en el periodo previo a la ejecución en el que se define el contexto del problema y su aproximación general.

b. Definición de la estructura de datos.

Al tratarse de información no congruente se debe definir la fuente de datos relacional del sistema y la fuente de datos no relacional, su esquema y estructura.

c. Determinar requisitos funcionales.

En este paso se determinará los factores que afectan el posicionamiento de la información que se está desplegando. Así como limitaciones de acceso a los datos por restricciones de negocio.

d. Identificar requisitos no funcionales.

Se definen los requisitos no funcionales para los sistemas de búsqueda y se escala según la necesidad del cliente.

e. Limitación de intervalos para los requisitos no funcionales.

Definidos los requisitos no funcionales se establece un rango de aprobación de estos.

2. Caracterización tecnológica.

a. Búsqueda de sistemas de indexación disponibles.

Se realiza una búsqueda de los sistemas de búsqueda disponibles como sistemas de código abierto o cerrados para la implementación.

b. Realizar compendio de características de los sistemas de búsqueda.
Se establecen las características relevantes para el sistema y se hace un compendio de las comparaciones encontradas.

c. Filtrar sistemas de búsqueda según requisitos y estado de la tecnología.
Se someten los sistemas encontrados a un escrutinio dependiente de los requisitos predefinidos y las comparaciones halladas.

3. Métricas de prueba.

a. Determinar métricas de medición en sistemas de búsqueda.
Se determinan las métricas de evaluación del sistema que sean relevantes para el caso.

b. Definir la validez de las métricas para el sistema de búsqueda.
Se establece que tan apropiadas son a los requisitos establecidos.

c. Implementación de pruebas para el sistema de búsqueda.
Se complementan con pruebas para su rápido acceso y condicionamiento.

d. Probar sistema de búsqueda encontrado.
Se someten los sistemas encontrados a las pruebas realizadas.

e. Determinar sistema de búsqueda a implementar.
Se evalúan los resultados de las pruebas y se determina la viabilidad del sistema para su implementación en el respectivo caso.

4. Implementación Sistema.

a. Identificar metodologías de distribución de carga para el sistema de **búsqueda.**
Se establece la metodología de fragmentación de índices adecuada según los datos e información bibliográfica encontrada.

b. Levantamiento sistema de búsqueda.
Se implementa el sistema de búsqueda para su acceso e interacción.

c. Integración con aplicativo.
Se integra el aplicativo con el sistema de búsqueda con el aplicativo de uso.

d. Pruebas a la implementación del sistema.
Se realizan pruebas a la implementación correcta del sistema y su adecuación al caso.

e. Parametrización de efectos.
Se modifican las variables establecidas entre múltiples posibilidades con el objetivo de realizar pivotes de adecuación. Estas modificaciones están respaldadas por el conocimiento del sistema.

f. Ajuste sistema búsqueda requisitos.

Se establece el mejor ajuste de las modificaciones establecidas para los requisitos del sistema.

g. Optimización rendimiento sistema de búsqueda

Se realiza una optimización volviendo el proceso iterativo en la implementación. Adecuándose siempre a los requisitos establecidos.

Resultados y análisis

1. Elicitación de requisitos:

Se identifica que el problema radica en un sistema de búsqueda que potencie la capacidad de búsqueda de un aplicativo, esto incluye diferentes datos que van a ser almacenados. Cuatro en total que incluyen diferentes estructuras dentro del aplicativo. Uno de estos datos tendrá mayor afluencia que el resto por ser el centro del negocio y puede que este sistema se busque expandirse a un sistema de manejo de métricas.

Debido al avance del proyecto se definió que la fuente de datos recaía en una base de datos relacional manejada bajo el sistema de gestión de base de datos PostgreSQL donde todos los datos eran almacenados para el sistema.

Se realizó la recopilación de los requisitos funcionales y los no funcionales que se muestran a continuación.

Requisitos funcionales:

- Autocompletado de resultados de búsqueda.
- Subrayado de términos que se usen en el proceso de búsqueda.
- Búsqueda en múltiples columnas o campos de los datos.
- Generar mayor relevancia a los resultados más actuales.
- Dar mayor relevancia a ciertos campos en el proceso de búsqueda.
- Actualización constante de los registros para el proceso de búsqueda
- Usar un sistema previamente probado o en estado productivo.

Requisitos no funcionales:

- Escalabilidad del sistema de búsqueda, debe permitir al usuario aumentar la capacidad del sistema de búsqueda con facilidad sin sacrificar rendimiento del sistema para un aumento en la cantidad de usuarios y manejo de datos.
- Efectividad: los resultados que el sistema expone deben satisfacer las necesidades de búsqueda y debe incluir los nuevos registros que se agreguen con la evolución del aplicativo.
- Alta tasa de disponibilidad.
- Alto rendimiento en los procesos de búsqueda del dato principal.

Intervalos considerados para los requisitos no funcionales del proceso de búsqueda:

- En términos de escalabilidad se espera que el sistema pueda manejar una tasa de registros de 10 registros principales por hora en promedio lo que equivaldría a 87600 registros al año, a los que se les debe adicionar un promedio de 5 registros adicionales pertenecientes al resto de los datos. Lo que equivaldría a 433000 datos al año.
- La disponibilidad debe hacerse con una tasa de 99.9% según lo estipulado esto equivaldría a 43.2 minutos cada 30 días como tasa máxima de fallos [11]. Esto como patrón general debido a que no se saben los objetivos del negocio definidos debido a su naturaleza.
- La tasa de respuesta del proceso de búsqueda se espera que no suceda con un valor mayor a 500 ms para una búsqueda de un término. Esto basado en los términos de aceptable de la agencia de optimización CXL [12].

2. Caracterización tecnológica:

Actualmente en el mercado existen los siguientes sistemas de búsqueda de texto completo disponibles identificados:

- Elasticsearch
- Lucene
- Meilisearch
- Rucene
- Pisa
- Algolia
- Sphinx
- Manticore Search
- Typesense
- Postgres Full text search
- Bayard
- Sonic
- Toshi
- Implementación de algoritmo de búsqueda de texto completo [2]

Según los requisitos exigidos se buscaron las características de cada uno de los sistemas y se realizó un proceso de filtrado de estos. Lo que dio como resultado el sistema de búsqueda que iba a ser implementado bajo el siguiente análisis.

Sistemas como Lucene, Meilisearch, Rucene, Pisa, Sonic y Sphinx son sistemas diseñados para un único nodo de funcionamiento, si bien muchos de estos sistemas pueden ser escalados en varios nodos estos no presentan un sistema de escalamiento por defecto y para el alcance del proyecto debido al plazo de implementación se opta por una solución construida. Además, soluciones como Sonic únicamente manejan un sistema

de índices lo que implica que se debe realizar una consulta adicional a la base de datos relacional por la información requerida de cada uno de los resultados [13].

El mismo fenómeno de verse comprometido a una implementación propia y a escalamiento se puede evidenciar con el sistema de búsqueda a partir de la implementación de un algoritmo o la implementación de búsqueda de texto de PostgreSQL, donde debido a el tipo de motor de base de datos usado la escalabilidad estaría comprometida y la implementación llevaría más tiempo del abarcado para el proyecto.

Por otro lado, sistemas como Toshi y Bayard si bien poseen un sistema multinodo, aún no se encuentran en estado productivo debido a que tanto su dependencia principal Tantivy como estos proyectos en separados se encuentran en desarrollo. Se apremia estos sistemas de búsqueda cuando estén en estado productivo debido a su desempeño superior y al sistema de alta disponibilidad que poseen.

Sistemas de búsqueda como Manticore search poseen características de rendimiento buenas, pero su sistema de fragmentación es manual al igual que la replicación. Lo que implica que para una alta disponibilidad se requeriría un esfuerzo adicional en términos de replicabilidad y mantenimiento del sistema.

Typesense también es un buen prospecto como sistema de búsqueda, pero este es muy dependiente del tamaño de memoria RAM del dispositivo usado lo que implica que no puede ser escalado tan fácilmente sin un costo adicional. Algo similar sucede con Algolia donde el costo de uso del SAAS puede ser mayor al de otros sistemas autogestionados por lo que no se dispone como un sistema viable en primera opción para el proyecto [14].

Por último, Solr si bien es un sistema de búsqueda basado en Lucene al igual que Elasticsearch este no tiene algunas de las capacidades de este último como el rebalanceo automático de los fragmentos, además su mantenimiento y despliegue tienden a ser labores más intensas que Elasticsearch [15].

Fueron por estos motivos que se escoge Elasticsearch como motor de búsqueda para la implementación del sistema, ya que este cuenta con un sistema de gestión multinodo, cumple los requisitos funcionales y posee un sistema de réplicas automatizado bajo un modelo de replicación pasiva y activa. Además de su selección se realizaron pruebas definidas para la optimización de la petición según los requisitos interpuestos por el usuario.

3. Métricas de prueba:

En términos de métricas de prueba se han hecho esfuerzos comparativos de los diferentes sistemas de búsqueda en términos de su rendimiento para la ejecución de una búsqueda donde uno de los valores esenciales es el tiempo de respuesta de cada una de las peticiones para la búsqueda. Si bien este valor puede ser variable según el término buscado, se han encontrado resultados abarcados dentro de los intervalos de

los requisitos no funcionales. Los valores reportados para el motor de búsqueda de Elasticsearch (Lucene) son de 4,072 ms en promedio [16]. No son los resultados de menor valor ya que comparado con Tantivy está ubicado en un promedio del doble de tiempo de ejecución, pero se encuentra dentro del rango deseado. Estas pruebas se realizan haciendo diversas peticiones con diferentes términos a los sistemas de búsqueda buscando a través de diferentes estructuras de peticiones cuales son los tiempos de respuesta de los sistemas de búsqueda [17]. Esta métrica es esencial para el rendimiento de los motores de búsqueda, pero no expresa su capacidad de ser dinámico a nuevas entradas, ni la relevancia de los resultados encontrados.

Para la relevancia de los resultados se ha implementado en estudios la métrica de exhaustividad donde se divide el número de resultados esperados sobre el número de resultados totales del sistema. En el caso del estudio se considera un caso esperado que incluye el título referenciado en el proceso de búsqueda. De igual forma la capacidad de escalamiento y el rendimiento de la plataforma se han medido a través de pruebas de desempeño donde se han usado 50 usuarios virtuales durante un periodo de 20 min, en este caso se mide la cantidad de peticiones que han sido devueltas y el tiempo de respuesta promedio de estas, entre petición y entrega [5]. Este tipo de métricas atiende a los requisitos no funcionales y puede dar una dirección específica del rendimiento del sistema en términos de la implementación final.

En los términos del presente trabajo se realizó la implementación de pruebas de rendimiento sobre el sistema. Primero se tomaron datos de prueba de películas y series de televisión de Netflix debido a su correlación con el tipo de datos trabajados [18]. Se decidió manejar los mismos parámetros de las pruebas de rendimiento usadas por Ahilaa Rani Kamuthurai en su estudio teniendo una variación en el tiempo de realización de la prueba a 1 min [5]. Las pruebas fueron realizadas a través del programa Jmeter. Los resultados fueron de 20848 peticiones con un promedio de un tiempo de respuesta de 143 ms con máximos de 456 ms y mínimos de 33 ms en un computador con SO BigSur, 8 GB de RAM 2133 MHz y un procesador Intel dual core i5 de 2.3GHz, el sistema de Elasticsearch estaba montado sobre Docker en la versión 7.13. Se debe tener en cuenta que estos resultados no son comparables a los obtenidos por Ahilaa Rani debido a que se realizaron dentro de la misma red, la query que se ejecutó es diferente y los datos que se ejecutaron divergen. En este caso se ejecutó una petición de priorización basada en tres campos con una priorización a dos de estos campos. En la tabla 1 se muestran los resultados obtenidos:

Configuración	Número de peticiones efectuadas	Tiempo promedio de respuesta en (ms)	Tiempo mínimo de respuesta (ms)	Tiempo máximo de respuesta (ms)
Petición booleana a tres campos	20848	143	33	456

Tabla N° 1

Por otro lado, la escalabilidad del sistema fue comprobada a través de artículos donde los reportes de tiempos de respuesta reportados por Ahilaa Rani se encuentran entre los 276.66 y 323.42 ms para diferentes configuraciones de distribuciones de los nodos de Elasticsearch y diferentes peticiones.

Al final con los datos obtenidos y con los datos de la literatura se establece que el sistema es viable para la implementación de Elasticsearch como sistema de búsqueda debido a que cumple las características y los rangos de los requisitos funcionales y no funcionales.

4. Implementación del Sistema:

En términos de implementación lo primero a considerar es la metodología de distribución debido a la plataforma donde se maneja el aplicativo. Se opto por ver las opciones de montaje en AWS. En esta plataforma hay dos opciones para el montaje de Elasticsearch una es hosteada en los servidores virtuales (EC2) que posee AWS, gestionada y mantenida en configuración por el personal de la empresa. La otra opción es hacer uso del AWS Elasticsearch Service, un servicio autogestionado por AWS donde se establece una configuración mínima de la distribución y la mayor parte de la gestión se da por AWS.

En un estudio realizado por Ahilaa Rani se concluyeron las siguientes características de ambas plataformas [5]:

- El servicio de AWS ES no es posible detenerlo sin configuración adicional de respaldo en otra plataforma.
- La cantidad de configuraciones del servicio de AWS ES son limitadas. Esto incluye la segmentación de fragmentos.
- AWS ES son más fáciles de establecer y mantener.
- AWS ES provee un sistema de monitoreo por defecto del clúster a través del servicio de CloudWatch, aunque este está limitado al clúster y no se pueda un monitoreo más granular.
- El costo de AWS ES es dos veces mayor a un sistema autogestionado en términos de servicio (Esto no incluye costos de mantenibilidad).

- La integración con otros servicios de AWS es más rápida en términos de AWS ES, como pueden ser un sistema de acceso granular como AWS Cognito o AWS Kinesis Firehouse.

Sabiendo estas condiciones se decidió por usar el servicio de AWS ES, esto debido a que para el plazo de entrega del proyecto y el alcance de este en la fase inicial se prefiere la entrega del producto al cliente. Posteriormente se podría hacer el proceso de migración del proyecto a un sistema autogestionado (Con el objetivo de aprovechar la configuración granular que puede tener el sistema de búsqueda).

En términos de la configuración que se debe realizar en AWS ES está determinada por la cantidad de nodos maestros y nodos de datos que se van a poner a disponibilidad del sistema, al igual que el tamaño de estos en recursos. Se define a estos nodos como:

- Nodo maestro: nodo que es responsable de mantener la salud del cluster, monitorear la posición de los fragmentos, mantener el estado del esquema de los índices y mantener registro de los nodos que acompañan o abandonan el cluster [5].
- Nodo de datos: nodos que almacenan los datos y funcionaran como servicio para las peticiones (Nodos de trabajo) [5]

En un caso de un sistema de gran tamaño tener una configuración de tres nodos maestros para evitar el problema de cerebro partido en los nodos maestros, donde en caso de caída de uno de los nodos, otro de los nodos no es capaz de detectar si el otro nodo está fuera de servicio y no volverá a estar disponible o si es un problema de conexión. El hecho de tener más nodos maestros en el clúster reduce el rendimiento de manera considerable, pero en términos de resiliencia del sistema aumenta la disponibilidad que este puede tener. Debido a la naturaleza del aplicativo se opta por una configuración mínima de ninguna instancia dedicada maestra. Esto porque se trata del escalamiento inicial de la aplicación que es capaz de mantener los estándares de disponibilidad, debido a las capacidades y monitoreos de AWS que permitirían una acción efectiva en caso de picos de afluencia en la etapa inicial.

En el caso del sistema configurado por AWS ES la cantidad de réplicas mantenida por cada uno de los fragmentos es de 5 a 1 lo que significa que por 5 fragmentos hay una réplica de estos. Esta configuración representa un alto costo de almacenamiento dependiendo del tamaño de los fragmentos [19][20]. Es por eso por lo que se escoge un tamaño inicial de fragmento de 30 GB que abarca toda la información estimada en los primeros 6 meses.

En términos del tamaño de las instancias se decide usar una replicación de dos zonas con el objetivo de reducir la pérdida de datos, aunque se podría reducir a una zona de disponibilidad debido a que la información está contenida principalmente en el sistema de base de datos relacional, pero esto representaría un gran costo de puesta en marcha luego de una caída de una de las zonas de disponibilidad. Se decide usar nodos de tipo M5.large, Esto debido a que estos nodos son de uso intensivo en memoria y permiten un aprovechamiento más efectivo de los recursos de cacheo que presenta AWS ES [21].

Es de esta manera que se integra el sistema de búsqueda al aplicativo, para la replicación de la información contenida dentro de la base de datos al sistema de búsqueda se hace uso de Logstash que de manera periódica inserta nuevos registros a los índices. Para el ambiente de desarrollo se pone a disposición a través de contenedores de Docker montados a través de docker-compose el sistema de búsqueda que puede ser levantado a necesidad del desarrollador para su prueba y uso.

El flujo en la aplicación del proceso de búsqueda se planteó como se evidencia en la siguiente figura.

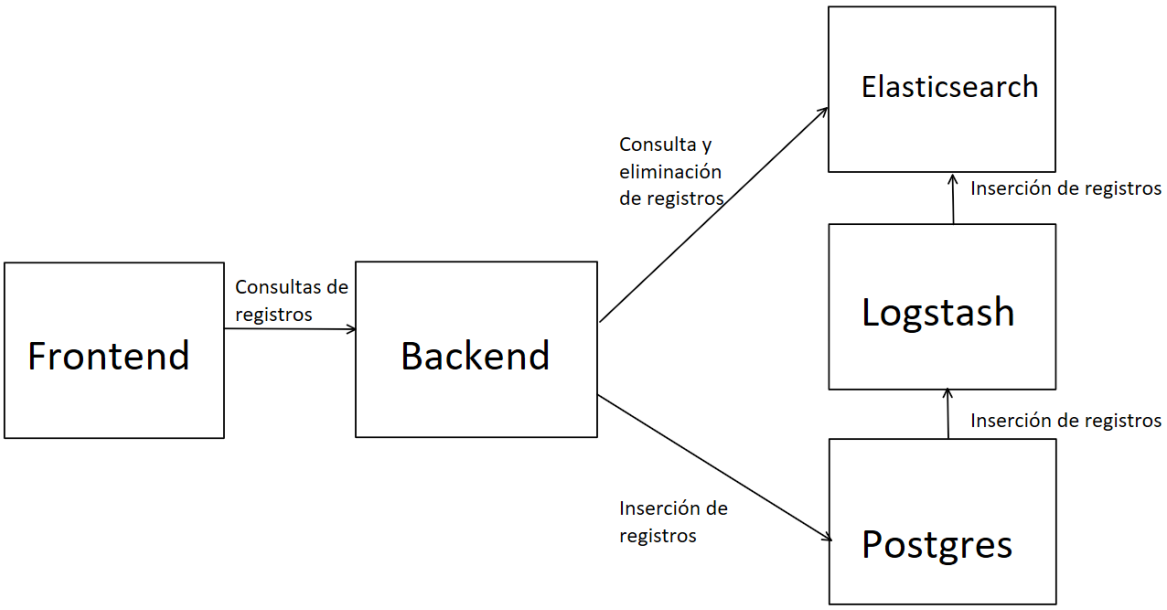

Figura N° 1

Como se puede evidenciar el proceso de inserción de registros se hace a través de PostgreSQL que gracias a la periodicidad de actualización de Logstash actualiza los índices de Elasticsearch, por otro lado la consulta y eliminación de registros se hace a través del backend del sistema esto con el objetivo de dar una capa de protección adicional al servidor de Elasticsearch de una intrusión directa de consulta del frontend donde se pueda comprometer la información de indexaciones de métricas no visibles para los usuarios.
La última labor realizada en el proceso fue el proceso de optimización de la petición de búsqueda de registros. En esta se tuvieron en cuenta que los resultados fueran los deseados como primera instancia con la métrica de exhaustividad y en segunda que la tasa de respuesta fuera la menor posible.

Primero se generó una petición de prueba base formada por tres peticiones booleanas de coincidencia de un término (Match) como se muestra a continuación. Estas peticiones permiten encontrar una coincidencia en la petición si alguno de los campos establecidos posee el término referido.

```
"query": {
    "bool": {
        "should": [
            {
                "match": {
                    "field1": {
                        "query": "test",
                        "_name": "field1"
                    }
                }
            }, {
                "match": {
                    "field2": {
                        "query": "test",
                        "_name": "field2"
                    }
                }
            }, {
                "match": {
                    "field3": {
                        "query": "test",
                        "_name": "field3"
                    }
                }
            }
        ]
    }
}
```

Fragmento de código N° 1

En términos de rendimiento se obtiene en promedio un tiempo de respuesta de 157 ms en promedio donde se resume las métricas obtenidas en la siguiente tabla.

Configuración	**Número de peticiones efectuadas**	**Tiempo promedio de respuesta en (ms)**	**Tiempo mínimo de respuesta (ms)**	**Tiempo máximo de respuesta (ms)**
Petición booleana termino a tres campos	18930	157	57	431

Tabla N° 2

Por otro lado, se hace uso de la petición a través de la interfaz de multimatch, esta permite realizar el mismo procedimiento que la anterior petición con la diferencia que la sintaxis es más resumida.

```
"query": {
    "multi_match": {
        "query": "test,
        "fields": ["field1", "field2", "field3"]
    }
}
```

Fragmento de código N° 2

A través de esta petición se obtiene un tiempo de respuesta promedio de 153ms, además se obtienen las siguientes métricas.

Configuración	Número de peticiones efectuadas	Tiempo promedio de respuesta en (ms)	Tiempo mínimo de respuesta (ms)	Tiempo máximo de respuesta (ms)
Petición multi_match termino a tres campos	19525	153	40	305

Tabla N° 3

Donde se puede intuir que debido a la naturaleza de esta petición los tiempos de respuesta son menores que en el caso de la petición booleana de match. Esta petición se edita debido a que muchos de los resultados obtenidos no eran los esperados ya que el autocompletado requería de un manejo de la petición de búsqueda como prefijo de los términos que se encuentran en el índice. Esta modificación mejora los resultados de exhaustividad del sistema y alcanzó tasas de respuesta de 133 ms, este valor fue menor a los anteriores en promedio, pero presentó máximos mayores que la petición multi_match sin este tipo de variación de búsqueda. Como se muestra en la tabla a continuación.

Configuración	Número de peticiones efectuadas	Tiempo promedio de	Tiempo mínimo de	Tiempo máximo de

		respuesta en (ms)	respuesta (ms)	respuesta (ms)
Petición multi_match termino a tres campos con selección de prefijos	22416	133	31	376

Tabla N° 4

La reducción en el tiempo de ejecución puede deberse a la comparación únicamente de secciones iniciales de los términos, lo que podría ahorrar en búsqueda. Por último, a esta petición se le adicionó el sistema de subrayado de los términos buscados y el sistema de relevancia según el tiempo más reciente de actualización del término.

Con esas variaciones aumenta el tiempo promedio de respuesta a 218 ms lo que implica que debido a la priorización de los campos y la generación de los subrayados en la búsqueda representan un tiempo de computación adicional. En la siguiente tabla se muestra el resumen de los resultados obtenidos para esta petición.

Configuración	Número de peticiones efectuadas	Tiempo promedio de respuesta en (ms)	Tiempo mínimo de respuesta (ms)	Tiempo máximo de respuesta (ms)
Petición multi_match termino a tres campos con selección de prefijos, subrayado y priorización de recientes	13734	218	139	553

Tabla N° 5

Podemos evidenciar que el tiempo máximo aumenta al límite del factor de rendimiento interpuesto en los requisitos funcionales, para esto se puede buscar realizar el proceso de subrayado en el cliente con el objetivo de reducir la carga del servidor, pero este no estaría optimizado para el manejo de los términos y el tiempo de respuesta del cliente podría aumentar más de lo esperado.

Conclusiones

Este trabajo presentó el proceso de selección, configuración y puesta en marcha de un sistema de búsqueda para datos no congruentes. Donde los requisitos que tuvieron más relevancia estaban enfocados al rendimiento, disponibilidad y efectividad del sistema. Cada uno de estos requisitos definido dentro de un margen establecido por el cliente.

La definición de los requisitos exigió la decantación del sistema de búsqueda a Elasticsearch, debido a que este sistema poseía capacidades de escalamiento bases, sin la necesidad de la construcción de un programa para dicho fin. Además, este sistema de búsqueda cumplía con estar a nivel productivo y tener capacidades de subrayado y priorización de campos.

Se encontró que los sistemas de búsqueda podían ser probados a través del uso de pruebas de desempeño para medir los tiempos de respuesta y que los rangos aceptables podían ser menores a 1 segundo. Además, se identificó que la exhaustividad del sistema era una medida relevante, pero dependía de los resultados esperados por el usuario para su definición.

De esta manera se concluyó que, en la implementación del sistema de búsqueda, se evidenció que el tiempo de respuesta era dependiente de la distribución realizada debido a estudios previos en el campo. La petición juega también un papel relevante ya que peticiones como multi match son más rápidas en este caso que una petición de base booleana y agregando nuevas funcionalidades a la petición como procesos de filtrado y subrayado aumentaban al doble los tiempos de respuesta. Para futuro trabajo se debe buscar tokenizadores o estructuras de datos más eficientes para el proceso de subrayado con el objetivo de bajar los tiempos máximos de búsqueda a los rangos esperados. Una de estas estructuras que puede ser probada son los vectores de términos que contienen información producida durante el proceso de análisis de los términos, evitando cálculos en tiempo de petición y convirtiéndolos en consultas.

Con el proceso de la practica se afianzó los conocimientos adquiridos durante el proceso de formación académica, siendo este un aporte considerable.

Durante el lapso se valoró el concepto de la investigación, uso de herramientas tecnológicas, análisis y creatividad para la resolución de problemas que fue de vital importancia en el desempeño del presente trabajo y estuvo relacionado con el proceso de formación académica dado por la universidad a través del uso continuo de casos de estudio referentes al sector, a las necesidades y procesos de desarrollo de un proyecto en el área de los sistemas de información, donde se dio también cabida a la importancia de las habilidades comunicativas o blandas en el proceso, siendo los conocimientos humanísticos y éticos de vital importancia para las interrelación laborales y la realización de un proceso integral científico y humano.

Referencias Bibliográficas

[1] M. S. Divya and S. K. Goyal, "ElasticSearch An advanced and quick search technique to handle voluminous data," *Compusoft*, vol. 2, no. 6, pp. 171–175, 2013.

[2] S. Bhandarkar and N. B. N., "A Full-Text-Based Search Algorithm vs Elasticsearch," *Stud. Indian Place Names, UGC Care J.*, vol. 40, no. 74, pp. 2168–2171, 2020.

[3] D. E. Knuth, *The Art of computer Programming. Sorting and Searching*, Second edi. Addison Wesley, 2011.

[4] A. Mallia, M. Siedlaczek, J. MacKenzie, and T. Suel, "PISA: Performant indexes and search for academia," *CEUR Workshop Proc.*, vol. 2409, pp. 50–56, 2019.

[5] A. R. Kamuthurai, "A Comparative Study of Self Hosted Elasticsearch vs AWS Elasticsearch," no. May, 2020.

[6] P. M. Dhulavvagol, V. H. Bhajantri, and S. G. Totad, "Performance Analysis of Distributed Processing System using Shard Selection Techniques on Elasticsearch," *Procedia Comput. Sci.*, vol. 167, no. 2019, pp. 1626–1635, 2020, doi: 10.1016/j.procs.2020.03.373.

[7] NIST, "inverted index," 2021. https://xlinux.nist.gov/dads/HTML/invertedIndex.html.

[8] H. Touzet and H. Ene Touzet, "On the Levenshtein Automaton and the Size of the Neighborhood of a Word," pp. 207–218, 2016, doi: 10.1007/978-3-319-30000-9_16ï.

[9] A. Brasetvik, "Elasticsearch from the bottom up," 2014. https://www.youtube.com/watch?v=PpX7J-G2PEo&t=475s.

[10] I. Syn Hershko, "Elasticsearch Do's, Don'ts and Pro-Tips," 2017. https://www.youtube.com/watch?v=c9O5_a50aOQ&t=2616s.

[11] A. Hilton, A. Ross, M. Brown, and D. Rensin, "SRE at Google: What is availability and what does it mean | Google Cloud Blog," 2017. https://cloud.google.com/blog/products/gcp/available-or-not-that-is-the-question-cre-life-lessons.

[12] M. Diggity, "Reduce Your Server Response Time for Happy Users, Higher Rankings | CXL," 2020. https://cxl.com/blog/server-response-time/.

[13] V. Saliou, "Announcing Sonic: A Super-Light Alternative to Elasticsearch," 2019. https://journal.valeriansaliou.name/announcing-sonic-a-super-light-alternative-to-elasticsearch/.

[14] Typesense, "Typesense vs Algolia vs Elasticsearch vs Meilisearch Comparison," 2021. https://typesense.org/typesense-vs-algolia-vs-elasticsearch-vs-meilisearch/.

[15] J. Blasenak, "Solr vs. Elasticsearch | Open Source Search | Accenture," 2019. https://www.accenture.com/us-en/blogs/search-and-content-analytics-blog/solr-elasticsearch-open-source-search-engines.

[16] Tantivy, "Search Benchmark," 2021. https://tantivy-search.github.io/bench/.

[17] K. Nilsson, A. Larsson, and J. Hagelbäck, "Search engines and how we evaluate them-A comparison between the search engines Elasticsearch and EPiServer Find," 2021.

[18] S. N. Bansal, "Netflix Movies and TV Shows | Kaggle," 2021. https://www.kaggle.com/shivamb/netflix-shows.

[19] J. Handler, "Get Started with Amazon Elasticsearch Service: How Many Shards Do I Need? | AWS Database Blog," 2017. https://aws.amazon.com/es/blogs/database/get-started-with-amazon-elasticsearch-service-how-many-shards-do-i-need/.

[20] V. Sharma and J. Handler, "Demystifying Elasticsearch shard allocation | AWS Open Source Blog," Aug. 13, 2019.

https://aws.amazon.com/es/blogs/opensource/open-distro-elasticsearch-shard-allocation/.
[21] J. Handler, "Reducing cost for small Amazon Elasticsearch Service domains | AWS Database Blog," May 05, 2020. https://aws.amazon.com/es/blogs/database/reducing-cost-for-small-amazon-elasticsearch-service-domains/.