DataFusionDB: Recuperación de texto e imágenes

José Joaquín Osnayo Matos, Luis Robledo Chuquicusma

December 5, 2024

1 Introducción

El proyecto **DataFusionDB** tiene como objetivo desarrollar una plataforma avanzada de bases de datos que integre diversos modelos de datos y técnicas de recuperación de información para proporcionar una solución eficiente y versátil para la búsqueda y el acceso a datos estructurados y no estructurados. En un mundo donde la cantidad de datos crece exponencialmente y la diversidad de los formatos se amplía, la capacidad para combinar datos textuales, imágenes y archivos de audio en una misma base de datos es crucial para proporcionar sistemas de recomendación y búsqueda más precisos y efectivos.

La plataforma **DataFusionDB** tiene como enfoque la implementación de dos partes fundamentales en la gestión de datos:

- **Búsqueda y Recuperación de Información en Documentos Textuales:** Para esta parte, se trabajará en la construcción de un índice invertido que permita realizar búsquedas eficientes en grandes volúmenes de texto. El índice invertido es un tipo de estructura de datos fundamental para sistemas de búsqueda, que permite realizar consultas rápidas basadas en el contenido textual. Este índice se complementará con un sistema de puntuación basado en la similitud de coseno, un algoritmo utilizado ampliamente para medir la relevancia de los documentos frente a una consulta.
- **Búsqueda y Recuperación de Información en Objetos Multimedia (Imágenes y Audio):** Dado el creciente uso de datos multimedia, la segunda parte del proyecto se centra en la construcción de un índice multidimensional para la recuperación eficiente de imágenes y audios. En esta sección, se implementarán técnicas avanzadas como la búsqueda KNN (k-nearest neighbors) y el uso de índices espaciales como R-trees para mejorar la velocidad de búsqueda en colecciones de grandes volúmenes de datos multimedia.

La capacidad de combinar la búsqueda de texto tradicional con la búsqueda en objetos multimedia amplía las posibilidades de recuperación de información, lo que hace que el sistema sea adecuado para una amplia gama de aplicaciones, como sistemas de recomendación, análisis de grandes volúmenes de datos y plataformas de búsqueda inteligentes.

Este proyecto también incluirá una evaluación del rendimiento de las técnicas implementadas, comparando su efectividad con soluciones estándar como PostgreSQL y MongoDB. Además, se implementará una interfaz de usuario que permita la interacción con el sistema de manera intuitiva, tanto para la búsqueda textual como multimedia, asegurando que los usuarios puedan obtener resultados rápidos y precisos.

1.1 Objetivo

El objetivo principal de este proyecto es desarrollar una plataforma integral de base de datos que permita la búsqueda y recuperación eficiente de información tanto en datos estructurados como no estructurados. A través de la construcción de un índice invertido para la búsqueda de documentos textuales y un índice multidimensional para la recuperación de objetos multimedia, se busca optimizar la recuperación de información en un sistema de recomendación de canciones.

Para lograr esto, el proyecto se divide en dos partes clave:

- **Parte 1:** Implementación de un índice invertido para realizar búsquedas eficientes en documentos textuales, como las letras de las canciones. El sistema debe permitir realizar consultas de texto libre y optimizar el proceso de recuperación mediante la técnica de similitud de coseno.
- **Parte 2:** Implementación de un índice multidimensional que permita la recuperación eficiente de objetos multimedia, como imágenes o audios, asociados a las canciones. Este índice se basa en técnicas como la búsqueda KNN (k-nearest neighbors) y el uso de estructuras espaciales como los R-trees.

Al integrar tanto la búsqueda textual como la búsqueda multimedia en una plataforma unificada, el proyecto tiene como propósito mejorar la experiencia de los usuarios de sistemas de recomendación, proporcionando resultados más precisos y relevantes basados en el contenido de las canciones y sus atributos asociados.

1.2 Dominio de datos

El dominio de datos de este proyecto está centrado en una base de datos que contiene canciones de Spotify. Este conjunto de datos incluye tanto información estructurada como no estructurada relacionada con las canciones, lo que lo convierte en un caso ideal para aplicar técnicas avanzadas de recuperación de información.

- **Datos Estructurados:** La base de datos contiene metadatos de las canciones, como el título, el artista, el género musical, la fecha de lanzamiento, la duración y otras características numéricas. Estos datos se organizan en un formato tabular, lo que facilita la indexación y la recuperación a través de técnicas tradicionales de bases de datos relacionales.
- **Datos No Estructurados:** Además de los metadatos, las canciones también incluyen datos no estructurados, como las letras de las canciones, que se almacenan en formato textual. La recuperación de la información en este caso se realiza utilizando técnicas de procesamiento de texto, como la tokenización y la eliminación de palabras irrelevantes (stopwords), seguidas de la construcción de un índice invertido para permitir la búsqueda eficiente.

La combinación de estos diferentes tipos de datos (estructurados y no estructurados) presenta desafíos interesantes en términos de diseño de bases de datos y recuperación de información, y es la razón por la cual este proyecto busca integrar diversas técnicas de indexación para facilitar una búsqueda eficiente tanto en textos como en objetos multimedia. Esta integración es crucial para mejorar la experiencia del usuario en plataformas de música como Spotify, donde la búsqueda de canciones, álbumes y artistas debe ser rápida y precisa.

2 Backend: Índice invertido

2.1 Construcción:

La construcción del índice invertido se realiza en varias etapas, comenzando con la obtención de los términos a partir de un conjunto de datos y finalizando con la creación de un índice invertido optimizado utilizando la técnica de *blocked sort-based index* y el algoritmo *SPIMI* para la fusión de bloques. A continuación se describen los pasos detallados:

2.1.1 Extracción de Datos

Primero, se selecciona una cantidad limitada de filas del archivo CSV que contiene los datos de las canciones. En este caso, se extraen las siguientes columnas:

- **track_name**: Nombre de la canción.
- **track_artist**: Artista de la canción.
- **lyrics**: Letras de la canción.
- **track_album_name**: Nombre del álbum de la canción.

- **playlist_genre**: Género de la canción en la playlist.

A continuación, se toma una muestra de estas columnas para generar un conjunto de términos que serán indexados. Este proceso se realiza de acuerdo con un número límite de filas (`dlimit`), para no sobrecargar la memoria del sistema.

2.1.2 Preprocesamiento de los Datos

Para poder construir un índice invertido eficiente, los datos deben pasar por un proceso de limpieza y preprocesamiento, que incluye las siguientes etapas:

1. **Tokenización**: Se separan las palabras de cada columna utilizando delimitadores comunes (espacios, comas, puntos, etc.). Cada término se extrae como una unidad básica que será indexada.
2. **Eliminación de Stopwords**: Se eliminan las palabras irrelevantes o comunes (*stopwords*), como artículos, preposiciones y pronombres, que no aportan valor significativo en la búsqueda. Para este propósito, se utiliza un archivo `stoplist.txt` que contiene listas de stopwords en inglés y español.
3. **Stemming**: Se aplica un algoritmo de stemming para reducir las palabras a su raíz o forma base. Esto ayuda a agrupar diferentes variaciones de una misma palabra (por ejemplo, "correr", "corriendo" y "corrí" se reducen a "corr").

Una vez finalizado el preprocesamiento, se obtiene una lista de términos preprocesados que son utilizados en la siguiente etapa.

2.1.3 Creación del Índice Invertido Local

En esta fase, el objetivo es construir un índice invertido que relacione cada término con los documentos en los que aparece. Para ello, se realizan las siguientes operaciones:

- Para cada fila del conjunto de datos, se examinan las columnas seleccionadas (`track_name`, `track_artist`, `lyrics`, `track_album_name`, `playlist_genre`).
- Por cada término de la lista de palabras preprocesadas, se realiza una búsqueda para determinar en qué documentos aparece. Cada aparición se registra como un par `t -> doc_id`.
- Los resultados se almacenan en bloques, donde cada bloque tiene un tamaño máximo de 4096 bytes (el tamaño de página del sistema operativo en este caso). A medida que se llena un bloque, se escribe en memoria secundaria (disco) y se continúa el proceso con un nuevo bloque.
- Cada bloque almacenado contiene las referencias a los documentos en los que aparece cada término.

El índice se almacena de forma incremental: cuando un bloque alcanza el límite de tamaño de página, se guarda en memoria secundaria, y se comienza a escribir un nuevo bloque. Esto permite manejar grandes volúmenes de datos sin cargar todo el índice en memoria RAM.

2.1.4 Construcción del Índice Global con SPIMI

Una vez que se han creado todos los bloques, el siguiente paso es combinar los índices invertidos locales de cada bloque en un único índice invertido global utilizando el algoritmo *SPIMI* (Single-Pass In-Memory Indexing). El algoritmo SPIMI permite fusionar de forma eficiente los índices invertidos de múltiples bloques sin necesidad de cargar todos los datos en memoria simultáneamente.

Los pasos para la fusión de los bloques con SPIMI son los siguientes:

1. Se ordenan los términos de cada bloque local.
2. Se realizan pasadas sobre los bloques, y para cada término se agrupan las entradas de los diferentes bloques que comparten el mismo término.

3. Durante la fusión, se mantiene el índice invertido en memoria secundaria, evitando cargar todo el índice en la RAM.

4. Finalmente, los índices locales de cada bloque se combinan en un solo índice global que contiene la lista de documentos para cada término.

Este proceso permite reducir considerablemente el tamaño del índice invertido global y optimizar la búsqueda y recuperación de información.

2.1.5 Resultado Final

Al concluir la fase de fusión con SPIMI, el índice invertido global estará disponible en memoria secundaria y será mucho más eficiente que los índices iniciales creados para cada bloque. El índice invertido global contendrá las referencias a todos los términos de los documentos de las canciones y permitirá realizar búsquedas rápidas y eficientes sobre el conjunto completo de datos.

Este índice final se puede utilizar para realizar consultas de texto completo sobre las letras de las canciones, los géneros y los nombres de los álbumes, permitiendo a los usuarios recuperar canciones relevantes basadas en términos de búsqueda específicos.

2.1.6 Resumen del Proceso

El proceso de construcción del índice invertido se puede resumir en los siguientes pasos:

- **Extracción de términos**: Se obtienen términos de las columnas `track_name`, `track_artist`, `lyrics`, `track_album_name` y `playlist_genre`.
- **Preprocesamiento**: Tokenización, eliminación de stopwords y stemming.
- **Creación de bloques**: Se crean bloques de términos y se almacenan en memoria secundaria cuando el tamaño del bloque alcanza 4096 bytes.
- **Fusión de bloques**: Se utilizan los índices locales creados por cada bloque para fusionarlos con el algoritmo SPIMI.
- **Índice invertido global**: El resultado es un índice invertido optimizado en memoria secundaria.

2.2 Consultas

Una vez construido el índice invertido en memoria secundaria, el siguiente paso es permitir que los usuarios realicen consultas sobre los datos indexados. Las consultas se procesan mediante una frase en lenguaje natural, y los resultados se ordenan en función de la relevancia de los documentos, medida mediante la similitud de coseno.

2.2.1 Proceso de Consulta

El proceso de consulta se realiza en los siguientes pasos:

1. **Entrada de la consulta**: El usuario ingresa una consulta en lenguaje natural. La consulta puede ser una frase que incluye términos relacionados con los atributos de las canciones, como el nombre del artista, el título de la canción, el género musical o incluso partes de las letras.

2. **Preprocesamiento de la consulta**: Al igual que las canciones durante la construcción del índice, la consulta debe pasar por un proceso de limpieza y preprocesamiento. Esto incluye:

    - **Tokenización**: Separar la consulta en términos individuales.
    - **Eliminación de Stopwords**: Se eliminan las palabras irrelevantes de la consulta utilizando un archivo `stoplist.txt`.
    - **Stemming**: Se aplica stemming a los términos de la consulta para obtener sus raíces, alineando así la consulta con el formato del índice.

3. **Búsqueda en el índice invertido**: Una vez preprocesada la consulta, se busca cada término de la consulta en el índice invertido. Dado que el índice está almacenado en memoria secundaria (disco), se realiza una búsqueda eficiente en los bloques del índice, evitando cargar todo el índice en la memoria RAM. La consulta solo accede a los bloques relevantes que contienen los términos de la consulta.

4. **Cálculo de la similitud de coseno**: Para cada documento relevante que contiene los términos de la consulta, se calcula la similitud de coseno entre el vector de características del documento y el vector de la consulta. La similitud de coseno es una métrica que mide la orientación de dos vectores, y es ideal para comparar la relevancia entre documentos y consultas. La fórmula para la similitud de coseno es:

$$\text{Similitud}(\mathbf{q}, \mathbf{d}) = \frac{\mathbf{q} \cdot \mathbf{d}}{\|\mathbf{q}\|\|\mathbf{d}\|}$$

   Donde:

   - $\mathbf{q}$ es el vector de la consulta, que contiene los pesos de los términos presentes en la consulta.
   - $\mathbf{d}$ es el vector del documento, que contiene los pesos de los términos presentes en el documento.
   - $\|\mathbf{q}\|$ y $\|\mathbf{d}\|$ son las normas (longitudes) de los vectores de la consulta y el documento, respectivamente.

   La similitud de coseno devuelve un valor entre -1 y 1, donde un valor más cercano a 1 indica que el documento es más relevante para la consulta.

5. **Top-k documentos**: Una vez calculada la similitud de coseno para todos los documentos relevantes, se ordenan los documentos en función de su puntuación (scoring). Los primeros k documentos con la puntuación más alta son seleccionados y devueltos como los resultados de la consulta. Este valor k es un parámetro configurable, que determina cuántos documentos serán recuperados.

2.2.2 Evitar Cargar Todo el Índice en la RAM

Como el índice invertido está almacenado en memoria secundaria, es importante evitar cargar todo el índice en la memoria RAM, especialmente cuando el tamaño del índice es muy grande. Para optimizar el uso de memoria, se sigue la siguiente estrategia:

- La consulta solo accede a los bloques del índice que contienen los términos relevantes de la consulta. No se cargan en memoria los bloques que no contienen términos de la consulta.
- El acceso a los bloques se realiza de forma secuencial, y los datos se leen desde la memoria secundaria (disco) según sea necesario.
- Solo se cargan los documentos de los términos que están presentes en la consulta, evitando así el uso innecesario de memoria.

Esta estrategia permite que las consultas se realicen eficientemente sin necesidad de cargar el índice completo en la memoria RAM, lo que es crucial cuando el índice es grande.

2.2.3 Recuperación de Resultados

El resultado final de la consulta es el conjunto de los k documentos más relevantes, que son devueltos en función de su similitud con la consulta. Los documentos se ordenan de mayor a menor relevancia, y la función de recuperación retorna los siguientes elementos:

- El **Nombre de la cancion** o `track_name`, que corresponde al documento recuperado (en este caso, la canción).
- El **Artista** (`track_artist`) del documento.

- El **Título de la canción** (`track_album_name` o `lyrics`), según la columna que más se ajuste a la consulta.
- La **Similitud de coseno** calculada, que indica cuán relevante es el documento respecto a la consulta.

Finalmente, los resultados son presentados al usuario en un formato amigable y ordenado, mostrando las canciones más relevantes primero.

2.2.4 Resumen del Proceso de Consulta

El proceso de consulta se puede resumir en los siguientes pasos:

- El usuario ingresa una consulta en lenguaje natural.
- La consulta se preprocesa (tokenización, eliminación de stopwords y stemming).
- Se busca cada término de la consulta en el índice invertido almacenado en memoria secundaria.
- Se calcula la similitud de coseno entre los documentos relevantes y la consulta.
- Se ordenan los documentos según su puntuación y se retornan los primeros k documentos más relevantes.
- Los resultados se presentan de manera amigable al usuario.

2.3 Construcción y consultas en Postgresql:

Para la construcción del índice en postgresql se usó psycopg2 para conectar python con postgresql, esto debido a que varios documentos de la base de datos tenían caracteres no convertibles a utf-8, por lo que la copia directa a una tabla de postgresql no funcionaba. Para el llenado de datos se creó la tabla canciones con la misma estructura de la base de datos y se pasaron los datos fila por fila usando excepciones para las filas con caracteres inválidos.

Para la creación del índice primero se redujo la base de datos a las columnas relevantes, luego se creó una columna con los términos de la letra de las canciones de tipo tsvector y se le indexó con un índice GIN.

```
'''
alter table canciones rename to canciones_completo;
create table canciones as
(select track_name,track_artist,lyrics from canciones_completo);
alter table canciones add column terms_lyrics tsvector;
update canciones set terms_lyrics = to_tsvector('english', lyrics);
drop table canciones_completo;
create index lyricsidx on canciones using GIN(terms_lyrics);
'''
```

Para la recuperación primero se crean los términos de la query a partir de lenguaje natural con 'plaintotsquery', luego mediante 'tsrankcd' con la columna de términos se obtienen los más similares al ordenar el ranking. No se usa '@@' porque restringe los resultados a los documentos que contengan todos los términos de la query, y queremos cualquier k que se nos pida.

```
'''
select track_name, ts_rank_cd(terms_lyrics, plainto_tsquery('english',%s))
as rank, track_artist, lyrics
from canciones
order by rank desc .
limit %s;
'''
```

3 Backend: Índice multidimensional

4 Frontend

La interfaz de usuario se compone de 3 vistas, el menú inicial para seleccionar entre la interfaz de recuperación de texto o imágenes, y las interfaces como tal. El menú principal simplemente se compone de una casilla grande con el título de la página y 2 botones grandes con cada opción.

4.1 Recuperación de texto:

La interfaz de recuperación de texto contiene el título de la interfaz, una casilla grande para ingresar texto, una casilla pequeña numérica para ingresar el k, una casilla de 2 opciones para seleccionar entre el índice propio o postgresql, y finalmente un botón para hacer la búsqueda. Al presionar el botón de buscar retornará una tabla con el título de la canción, el puntaje, el artista y la letra de la canción, donde la tabla contendría k filas.

4.2 Recuperación de imágenes:

5 Experimentación

5.1 Recuperación de texto: Implementación propia vs Postgresql

Para medir la eficiencia de la implementación del índice invertido propio se medirá el tiempo de ejecución de las consultas. Para esto se toma en cuenta el número de documentos de los que está formado el índice considerando un tamaño similar de términos para cada documento. También se toma en cuenta el número de términos que se forman del input. Para simplificar se trabajará con un mismo k = 10 y una misma consulta de prueba. Entonces lo que variará será el tamaño del índice, determinado por los n documentos que lo formaron.

En postgresql:

Para n menor a base de datos:

```
'''
create table canciones_t as
(select track_name, track_artist, lyrics from canciones limit x);
alter table canciones_t add column terms_lyrics tsvector;
update canciones_t set terms_lyrics = to_tsvector('english', lyrics);
create index lyricsidx_t on canciones_t using GIN(terms_lyrics);
'''
```

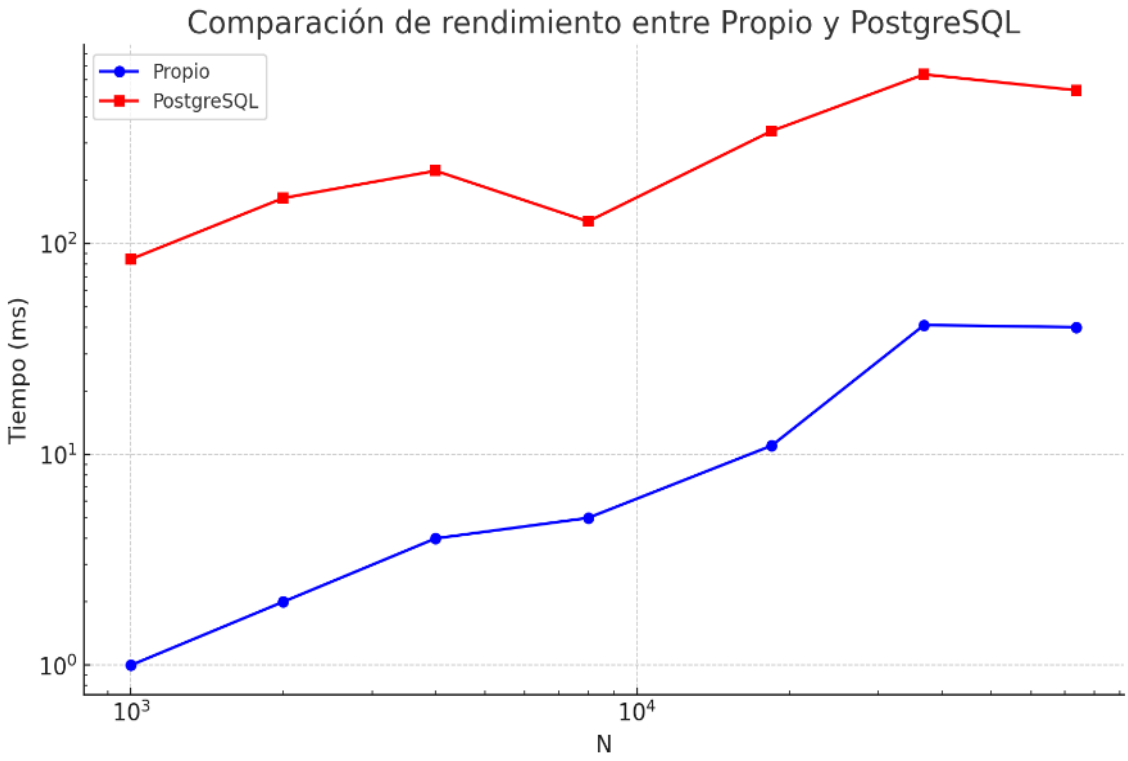
Para n mayor a base de datos:

```
'''
create table canciones_t as
(select track_name, track_artist, lyrics from canciones, generate_series(1, x) as s);
alter table canciones_t add column terms_lyrics tsvector;
update canciones_t set terms_lyrics = to_tsvector('english', lyrics);
create index lyricsidx_t on canciones_t using GIN(terms_lyrics);
'''
```

N = número de documentos del índice
resultados: tiempo de ejecución (milisegundos)
Consulta: one more time we're gonna celebrate
k = 10

N	Mi índice	Postgresql
1000	1.0006	84.5361
2000	2.0008	164.849
4000	3.999	221.9701
8000	5.0001	127.6231
18454	11.023	343.1873
36908	41.1575	636.5948
73816	40.1812	535.145

Table 1: Comparación de tiempos de ejecución para N documentos

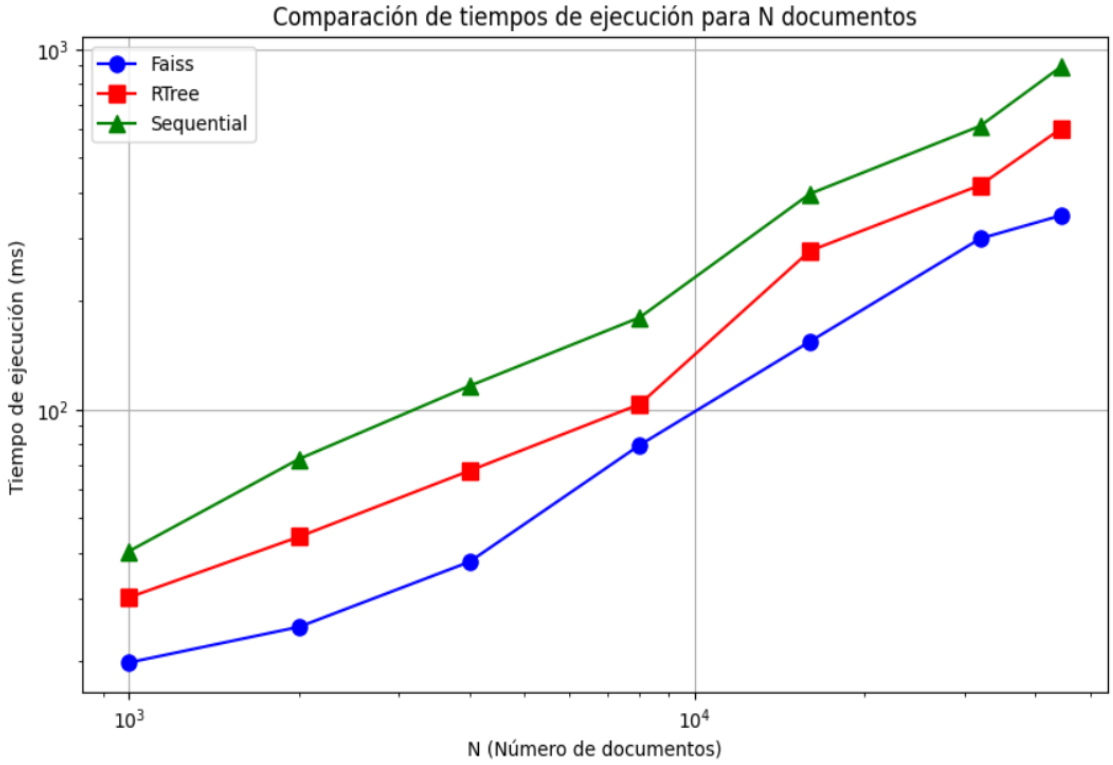


5.2 Recuperación de imágen: Faiss vs RTree vs Sequential

A continuación se procederá a realizar una comparación entre los 3 métodos usados para la recuperación de imágenes. Similar al caso anterior, el valor de K se mantendrá, y lo que cambiará sera el número de documentos usados para el testeo.

N	Faiss	RTree	Sequential
1000	19.87	30.12	40.45
2000	24.96	44.39	72.98
4000	37.81	67.65	116.32
8000	79.74	103.54	180.23
16000	154.57	275.88	397.21
32000	298.35	419.67	612.71
44446	345.78	602.31	894.321

Table 2: Comparación de tiempos de ejecución para N documentos



5.2.1 Análisis de resultados:

El índice propio es más eficiente que el de postgresql porque está hecho con un índice invertido y método de cosenos, lo que evita que se calcule todo el vector, está lo más optimizado posible. En cambio, si bien el índice de postgresql hace lo mismo, la forma en la que se consigue el resultado es distinta, el puntaje requiere que todos los términos de la consulta estén en los documentos, y no está limitada a 1, en el top pueden haber muchos documentos con puntaje 0, no usa exactamente el algoritmo eficiente de combinar índice invertido y método de cosenos.

En cuanto a la recuperación de imágenes, se puede observar que Faiss es la técnica más eficiente. Esto se debe a su diseño optimizado para grandes volúmenes de datos y su uso de téctnicas avanzadas de búsqueda aproximadaen espacios de alta dimensión. En comparación. RTree es menos eficiente, y esto se puede deber a que no está diseñado específicamente para bpusqueda de similitudes en imágenes, sobre todo cuato tienen altas dimensiones. Por último, la recuperación secuencial es la menos eficiente, especialmente cuando se manejan grandes volúmenes de datos, debido a su naturaleza de búsqueda sin optimización.