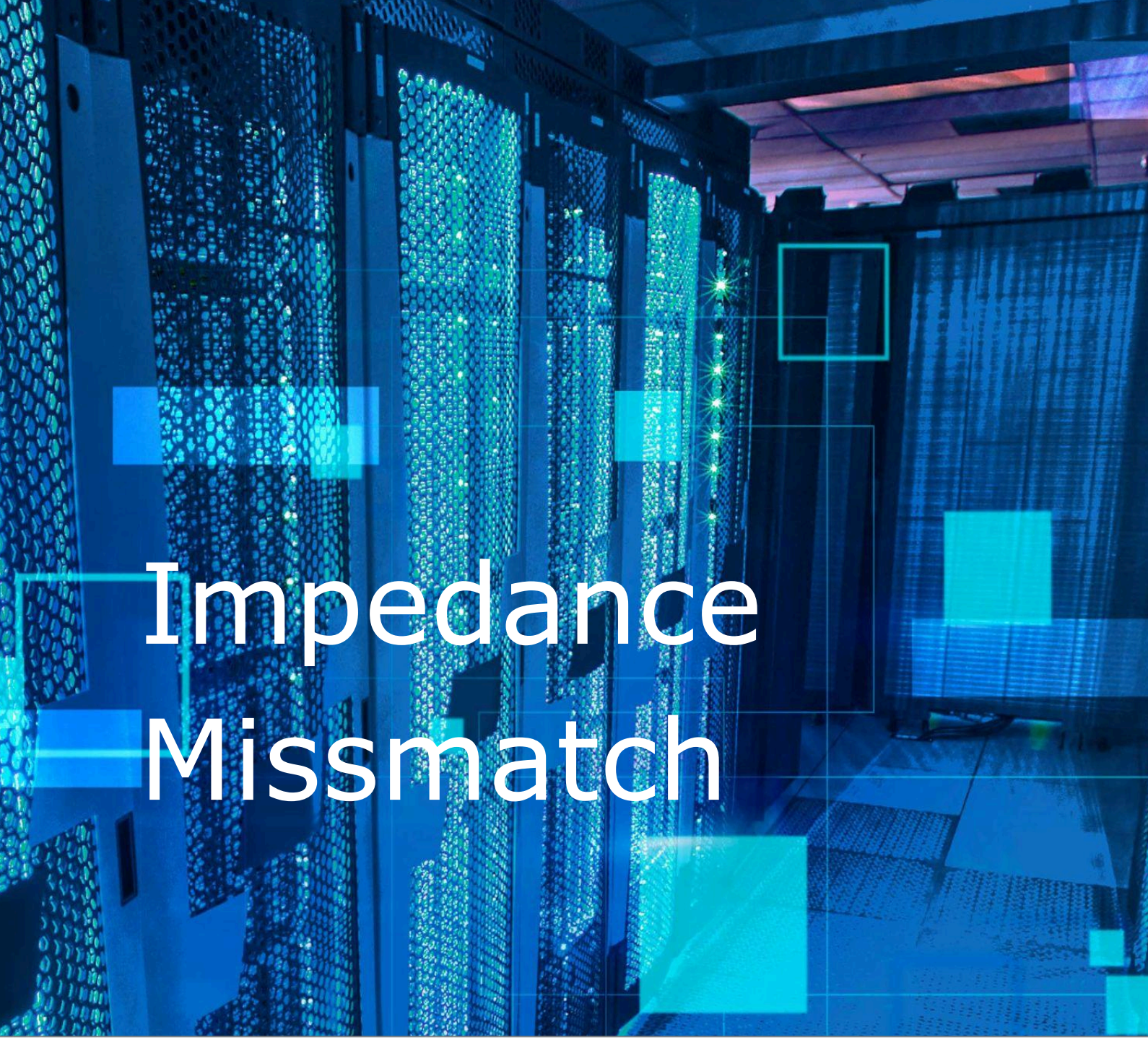


Impedance Missmatch

Hossam Zerouali

Richard Pie

UNIVERSITAT POLITÈCNICA
DE CATALUNYA
BARCELONATECH

Índice

https://github.com/ZeroualiHossam/cbde-lab1

Introducción

En este primer laboratorio, nuestro objetivo es ver y estudiar el impedance mismatch que se produce al trabajar con datos vectoriales. Para ello, vamos a ver las diferencias que podemos encontrar entre una base de datos relacional como podría ser PostgreSQL y una base de datos vectorial como es Chroma procesando y almacenando embeddings.

Lo primero que hemos hecho ha sido entender el concepto de impedance mismatch, el cual se refiere a esa pérdida de rendimiento que tenemos al almacenar datos vectoriales en una base de datos que no está diseñada para ello, ya que se requiere de una traducción de esos datos porque el método de almacenamiento de la base de datos y el formato que utiliza Python para estos datos no es el mismo.

Una vez entendimos el concepto de impedance mismatch empezamos a buscar un bookCorpus para descargar y del que obtener las frases. Una vez descargado, creamos un código simple para guardar 10.000 frases, que era la cantidad recomendada en las instrucciones para realizar la práctica, en un archivo txt (frases_extraidas.txt).

PostgreSQL

Para realizar la primera parte, la cual corresponde a la base de datos relacional PostgreSQL, instalamos todo lo necesario que se indicaba en las instrucciones y creamos nuestro entorno virtual para empezar a trabajar.

Tras realizar el setup de PostgreSQL y la instalación de todas las configuraciones necesarias tenemos tanto el archivo database.ini para la configuración de la base de datos (host, database, user, password), además del archivo config.py para realizar la configuración previa de la base de datos que consiste en leer el documento database.ini y parsear la información para que pueda ser utilizada por el archivo

connect.py el qual se encarga de realizar la conexión con la base de datos PostgreSQL mediante la configuración obtenida por parte de config.py

Para realizar el primer apartado, hemos modificado el código del archivo connect.py (connect_p0.py) para crear la tabla para las frases e introducir las 10.000 oraciones que tenemos en la base de datos. En un primer momento habíamos implementado un código que realizaba los inserts de cada frase por separado, pero vimos que existe la función executemany, con la que podíamos crear lotes de varios inserts y hacerlos a la vez. Por ello, hemos implementado que se realicen los inserts de 1000 en 1000. Esto nos ha mejorado el rendimiento en casi un 30%, ya que pasamos de tardar 1.16s a 0.83s.

En cuanto al segundo apartado, hemos modificado nuevamente el codigo del archivo connect.py (connect_p1.py) para crear la tabla para almacenar los embeddings, los cuales creamos mediante el uso de SentenceTransformer, una herramienta de Hugging Face Transformers, que facilita la conversión de las oraciones a embeddings, y que se nos ha recomendado en la introducción de la práctica. Una vez generados estos embeddings son almacenados en la tabla de PostgreSQL para poder calcular con ellos las siguientes métricas que se nos piden.

Por último, para realizar este tercer apartado, hemos modificado nuevamente el codigo del archivo connect.py (connect_p2.py) para coger 10 frases de las que tenemos almacenadas en nuestra base de datos PostgreSQL, y calcular la similitud mediante el cálculo de la distancia coseno y euclidiana utilizando sus embeddings. Para poder obtener las 2 frases con mayor similitud a las frases objetivo.

Are the insertion times for text and embeddings stable? What are your conclusions on this matter?

No, el tiempo que se tarda en realizar los inserts de las oraciones es mucho menor que el tiempo que se necesita para insertar los embeddings. Desde nuestro punto de vista, esto debe ser principalmente por el tiempo que se tarda en calcular estos embeddings. Se deben procesar las oraciones y calcular sus correspondientes

embeddings antes de insertarlas. En cuanto al tiempo de cada acción, en ese caso el de los inserts es estable, pero el del cálculo de embeddings no, tenemos un tiempo mínimo de 0.013s y un máximo de 2.099s, lo que nos muestra que los tiempos son inestables, pensamos que debe ser por la composición de las frases, ya que algunas de ellas son más largas y contiene más símbolos como podrían ser comas o apóstrofos.

Are the querying times stable when computing the similarities? Did you find differences between the two distance metrics you used? What are the conclusions on this matter?

Sí, en base a los tiempos que hemos obtenido como mínimo y máximo, podemos deducir que los tiempos son estables, es decir no hay grandes variaciones. Ya que tenemos un mínimo de 0.267s y un máximo de 0.3155s lo cual da una average de aproximadamente 0.277s. Esto nos hace ver que gracias al uso de embeddings, conseguimos que el tiempo de cálculo de estas similitudes sea estable y no haya grandes desviaciones, ya que pensamos que en caso de hacerlo sin el uso de los embeddings provocaría que hubiera mayores desviaciones porque pensamos que se haría en base a la composición de la frase.

Can you think of any available insertion method, data structure or indexing technique available in PostgreSQL (Pgvector is still not allowed to be used yet) that would improve the performance of these operators?

En nuestro caso, hemos decidido utilizar la estrategia de un mejor método de inserción. En vez de realizar los inserts cada vez que queremos añadir una frase, hemos decidido modificar el código para que se hagan varios inserts cada vez. De esta manera reducimos las llamadas a la base de datos porque cada insert por separado cuesta mucho tiempo.

Conclusiones

Tras ver los resultados obtenidos en estos ejercicios realizados con PostgreSQL, vemos principalmente un problema, el tiempo de cálculo de los embeddings es alto

e inestable, ya que creemos que depende del tipo de oración que se procesa y su composición, porque en el cálculo de embeddings tenemos una desviación estándar de 0.59s, en este caso al ser cantidades pequeñas no tiene un gran impacto, pero en caso de trabajar con datasets mucho más grandes, esta inestabilidad provocaría una gran pérdida de tiempo.

Chroma

En el primer apartado, hemos modificado el código del archivo chroma_c0.py para introducir las mismas oraciones que en caso de Postgres (10.000 oraciones).
para crear la tabla para las frases e introducir las 10.000 oraciones que tenemos en la base de datos.
De lo aprendido en Postgres hemos hecho inserts de lotes de frases, en este caso de 5.003 en 5.003 (la razón de esto es para hacer 2 operaciones ya que la cantidad de oraciones total es realmente 10.006 y esto afecta al cálculo de las estadísticas).
Para nuestra sorpresa, el añadir las frases de una en una es más rápido que hacerlo en lotes.
Por ejemplo:

- Introducidas de 1 en 1:

```
Tiempo máximo: 0.1459 segundos
Tiempo mínimo: 0.0145 segundos
Tiempo total: 211.9218 segundos
Tiempo promedio: 0.0212 segundos
```

- Introducidas de 5003 en 5003:

```
Tiempo máximo: 113.5333 segundos
Tiempo mínimo: 109.1776 segundos
Tiempo total: 222.7109 segundos
Tiempo promedio: 111.3554 segundos
```

En cuanto al segundo apartado no tiene mucho sentido en relación con Chroma ya que las frases que se guardan son embeddings.

Para el tercer apartado, hemos modificado nuevamente el código del archivo (chroma_c2.py) para coger 10 frases de las que tenemos almacenadas en nuestra

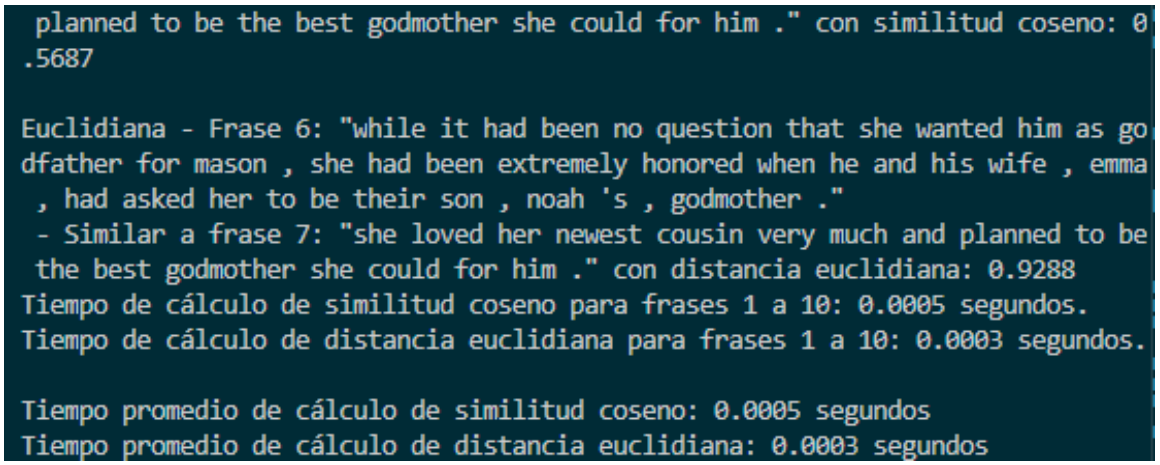
base de datos PostgreSQL, y calcular la similitud mediante el cálculo de la distancia coseno y euclidiana en sus embeddings.

Are the insertion times for text and embeddings stable? What are your conclusions on this matter?

Los tiempos son bastante estables. Hemos visto que varía dependiendo del tamaño del batch, pero tiene una desviación estándar de 7,63 segundos, que teniendo en cuenta el tiempo total de la operación es aceptable. Así queda un promedio de 217.31 ± 7,63 segundos.

Are the querying times stable when computing the similarities? Did you find differences between the two distance metrics you used? What are the conclusions on this matter?

Los tiempos son estables y tienden a 0. Entre ambas métricas no hay diferencias destacables, las dos arrojan resultados similares.
A diferencia de Postgres, Chroma trabaja mejor con embeddings por tanto no es extraño que los resultados obtenidos superan a los de Postgres.



Can you think of any available insertion method, data structure or indexing technique available in Chroma that would improve the performance of these operators?

Una buena forma de hacerlo sería encontrar un batch adecuado que no fuera tan grande como 5000 pero no tan pequeño como 1. Se podría hacer una búsqueda dicotómica del valor del bach que mejor se ajuste a un buen rendimiento, probando

con 2500, a continuación con 1250, etc.. para que en pocas iteraciones diéramos con el número de batch adecuado.

Conclusiones

En conclusión, el análisis realizado con Chroma ha revelado observaciones importantes sobre la inserción y consulta de datos. Aunque inicialmente se esperaba que la inserción de frases en lotes grandes fuera más eficiente, se encontró que insertar las frases individualmente resultó ser más rápido.

Esto sugiere que el manejo de lotes grandes puede no ser óptimo en este contexto específico, posiblemente debido a cómo Chroma gestiona las operaciones en comparación con Postgres. En cuanto a la estabilidad de los tiempos de inserción y consulta, se observó que los tiempos de inserción son bastante consistentes, con una desviación estándar aceptable de 7.63 segundos. Esta estabilidad es crucial para predecir el rendimiento y planificar operaciones a gran escala.

Por otro lado, los tiempos de consulta al calcular similitudes mediante las métricas de distancia coseno y euclidiana también fueron estables y rápidos, sin diferencias significativas entre ambas métricas. Esto indica que Chroma maneja eficientemente las operaciones con embeddings, superando en este aspecto a Postgres.

Para mejorar el rendimiento de las operaciones de inserción en Chroma, se sugiere explorar un tamaño de lote óptimo mediante una búsqueda dicotómica. Esto implicaría probar con tamaños de lote intermedios, como 2500 o 1250, para encontrar un equilibrio entre la eficiencia y el rendimiento.

Conclusiones finales:

From an impedance mismatch perspective, what is the difference between using PostgreSQL and Chroma? What are the pros and cons for each system? If you used the optional part, include Pgvector in the discussion.

PostgreSQL

Ventajas:

- **Flexibilidad y versatilidad:** PostgreSQL es una base de datos relacional muy robusta que ofrece una amplia gama de funcionalidades para gestionar datos estructurados y no estructurados.

Desventajas:

- **Rendimiento inestable en cálculos de embeddings:** Como se observó, el tiempo de cálculo de los embeddings en PostgreSQL es alto e inestable, con una desviación estándar de 0.59 segundos. Esto puede ser problemático al trabajar con grandes volúmenes de datos, ya que la inestabilidad podría provocar pérdidas significativas de tiempo.
- **Complejidad en el manejo de grandes volúmenes de datos:** Aunque PostgreSQL puede manejar grandes cantidades de datos, su rendimiento puede no ser óptimo para operaciones intensivas en cálculos vectoriales sin ajustes adicionales.

Chroma

Ventajas:

- **Estabilidad en inserciones y consultas:** Chroma mostró tiempos de inserción consistentes con una desviación estándar aceptable de 7.63 segundos, lo cual es crucial para planificar operaciones a gran escala.
- **Eficiencia en cálculos de similitud:** Los tiempos para calcular similitudes mediante métricas como la distancia coseno y euclidiana fueron estables y rápidos, indicando un manejo eficiente de las operaciones con embeddings.

- **Rendimiento superior en operaciones con embeddings:** En comparación con PostgreSQL, Chroma maneja más eficientemente las operaciones relacionadas con embeddings.

Desventajas:

- **Ineficiencia en inserciones por lotes grandes:** Se encontró que insertar frases individualmente era más rápido que hacerlo en lotes grandes, lo que sugiere que el manejo de lotes grandes no es óptimo en Chroma. Esto podría deberse a cómo Chroma gestiona internamente las operaciones.