Comparación de desempeño de algoritmos de camino más corto en Bases de Datos Relacionales y de Grafos para Redes de Transporte

Guillermo Ripa
Instituto de Computación
Facultad de Ingeniería, Universidad de la República
Montevideo, Uruguay
guillermo.ripa@fing.edu.uy

Ezequiel Velázquez
Instituto de Computación
Facultad de Ingeniería, Universidad de la República
Montevideo, Uruguay
ezequiel.velazquez@fing.edu.uy

I. Introducción

El presente informe tiene como objetivo realizar un abordaje global sobre los experimentos del articulo académico *The shortest path algorithm performance comparison in graph and relational database on a transportation network* [1] de 2013.

En las siguientes secciones se presentan los resultados de las reproducciones de los experimentos de dicha publicación así como la actualización de los mismos a las soluciones disponibles en la actualidad.

I-A. Descripción del problema

El objeto de análisis de la publicación es el problema del camino más corto entre dos puntos sobre una red de transporte, esto es, cual es la forma más efectiva para llegar desde un punto a otro. Para resolver dicho problema el artículo plantea una pregunta: *¿Qué Modelo de base de datos resuelve mejor la problemática del camino más corto?*, para responder esta pregunta se abordan un abanico de experimentos utilizando los motores de base de datos Neo4j [1] (Grafos) y PostgresSQL[2] (Relacional) con el fin de comparar los resultados entre estos y obtener respuesta a la pregunta planteada.

II. Marco teórico

A continuación se muestra un resumen del marco teórico empleado en la construcción del informe.

II-A. Algoritmo de Dijkstra:

El algoritmo de Dijkstra [2] tiene como objetivo el cálculo de el camino más corto entre dos nodos sobre un grafo de costes. La idea básica de dicho algoritmo se basa en partir de un subconjunto de nodos Q (que originalmente solo contiene la fuente) e ir agregando de forma iterativa los nodos subyacentes de menor costo hasta que no queden más nodos del grafo que no formen parte del subconjunto origen.

[1]https://neo4j.com/docs/
[2]https://www.postgresql.org/docs/13/index.html

II-B. Algoritmo A:*

El algoritmo A* [3] es una variación del algoritmo de Dijkstra el cual considera una función heurística sobre el grafo para intentar reducir la cantidad de evaluaciones innecesarias. El algoritmo anteriormente mencionado se basa fuertemente en al ecuación: $f(n) = g(n) + h'(n)$ en donde $g(n)$ representa el costo para alcanzar el nodo n y $h'(n)$ representa un costo aproximado para alcanzar el destino desde el nodo n. Dado lo anterior, este algoritmo depende fuertemente de que la heurística proporcionada sea buena, de lo contrario el algoritmo toma un orden exponencial.

III. Metodología Experimental

En la presente sección se describe la metodología experimental utilizada para replicar los experimentos de la publicación que se tomo como base para el informe.

III-A. Herramientas Utilizadas

Como ya se adelantaba en la introducción los motores de base de datos utilizados en los experimentos fueron Neo4j para el modelo de grafos y PostgresSQL para modelo relacional. Esto supone un reto en primera instancia ya que se deben extraer los datos fuente con formato *OSM*[3] para luego persistirlos, ejecutar el algoritmo de camino más corto en ambas bases de datos y recabar la información necesaria sobre tiempo, uso de memoria y cache en cada caso. Para resolver lo anterior se utilizaron un conjunto de herramientas listadas a continuación:

- **Osm for Neo4j**[4]**:** Herramienta colaborativa creada por el equipo de Neo4j que permite importar los datos de geografía desde un formato *OSM* hacia un modelo de grafos en la base de datos.
- **Neo4j Spatial**[5]**:** También del equipo de Neo4j, esta librería permite añadir meta datos necesarios para el cálculo de rutas como pueden ser intersecciones de calles o puntos de interés.

[3]https://www.openstreetmap.org
[4]https://github.com/neo4j-contrib/osm
[5]https://github.com/neo4j-contrib/spatial

- **Neo4j Apoc**[6]**:** Librería de Neo4j que permite ejecutar un conjunto de operaciones extra a las que ofrece Cypher[7] nativo, de esta librería se usaron los procedimientos para el cálculo de la ruta mas corta mediante los algoritmos de Dijktra y A*.
- **PostGIS**[8]**:** Extensión de PostgreSQL que agrega funciones para cálculo espacial tales como distancia, área y en específico tipos de datos geométricos.
- **PgRouting**[9]**:** Extensión desarrollada sobre PostGIS para proveer funcionalidades de ruteo geo-espacial.
- **Osm2pgrouting**[10]**:** Herramienta desarrollada por el equipo de PgRouting para transformar los archivos de mapas OSM hacia una base de datos de PostGIS con la información necesaria para realizar cuentas de enrutamiento.
- **Python**[11]**:** Se utilizó el lenguaje de programación para desarrollar un sistema de evaluación de desempeño estándar haciendo uso de las bibliotecas *psycopg2* y *neo4j* para conexión a las bases de datos. El mismo se encarga de realizar las pruebas, recopilar información y persistir los datos para su análisis posterior.

III-B. Motores de enrutamiento

Este informe también plantea un objetivo secundario de la investigación dirigido a profundizar en los conceptos que hacen posibles las soluciones de motores de ruteo sobre tecnologías de bases de datos relaciones y bases de datos de grafos. Ninguna de las bases de datos presentadas como componente principal de estos motores de ruteo fue concebida originalmente para este cometido.

Esta sección se enfoca en describir las adaptaciones necesarias a las mismas para poder hacer posible dichos motores de ruteo.

Para el caso de PostGIS, la adaptación necesaria se realiza en la forma de un nuevo tipo de datos. El tipo de datos Geométrico [12]. Este tipo de dato permite representar de forma atómica diferentes formas geométricas como puntos, líneas, polígonos, etc. En específico, tiene la capacidad de modelar objetos del mundo real (como rutas o caminos) utilizando una resolución constante para su representación. Esta resolución viene dada en la forma del SRID - Sistema Espacial de referencia -. De esta forma, definiendo un subtipo *ST_SRID(geometry)* es posible conseguir la identificación espacial de cualquier objeto.

De esta forma, las bases de datos relacionales realizadas sobre PostGIS para utilizar con PgRouting cuentan con una columna geométrica de caminos de los cuáles es posible conseguir su distancia, forma y posición en el mapa. Estos caminos también cuentan con sus "anclas" de comienzo y fin mediante las cuáles PgRouting logra definir las intersecciones existentes para generar el camino más corto.

[6]https://neo4j.com/developer/neo4j-apoc/
[7]https://neo4j.com/developer/cypher/
[8]https://postgis.net/documentation/
[9]https://docs.pgrouting.org/latest/en/index.html
[10]https://pgrouting.org/docs/tools/osm2pgrouting.html#features
[11]https://docs.python.org/3/
[12]http://postgis.net/workshops/postgis-intro/geometries.html

Estas bases de datos no cuentan con ningún tipo de jerarquía en su información más allá de la distinción entre punto y camino las cuales están definidas simplemente con diferentes tablas.

Neo4j logra un modelado más simple del problema ya que cada nodo del grafo puede modelarse como una intersección con latitud y longitud y las relaciones entre intersecciones cuentan con la distancia entre estas. También pueden existir nodos intermedios que únicamente forman parte de un camino, definiendo su forma pero sin aportar información al algoritmo de camino más corto.

III-C. Conjunto de datos

Para los experimentos presentados se utilizaron datos el mapa de Uruguay provisto por Open Street Map como fuente de datos. Esta fuente modela una geografía mediante la descripción de todos los puntos relevantes en el espacio (con su latitud y longitud correspondiente) y luego referencia los mismos para construir una entidad mayor como puede ser una calle, un camino, un río, entre otras cosas. Dentro de la entidades anteriormente mencionadas existe un parámetro *tag* con un fin descriptivo, esto permite utilizar esta información para definir puntos de interés, o servir de referencia en el mapa.

Para las cargas de las bases de datos se utilizaron múltiples herramientas mencionadas en la sección Herramientas Utilizadas.

Para Neo4j por ejemplo se utilizan *Osm for Neo4j*, herramienta la cual permite extrapolar las entidades del archivo OSM (mencionadas anteriormente) a una base de datos de grafos. Cabe destacar que esta herramienta no genera un grafo de enrutamiento funcional luego de su ejecución, simplemente realiza una correspondencia directa entre los datos de la fuente OSM y el grafo.

Para esto se utiliza La ayuda de *Neo4j Spatial* y *APOC* las cuales son de utilidad para calcular distancia entre nodos (en km) y ejecutar de forma iterativa la misma consulta. Esto es útil ya que para crear el grafo de enrutamiento se deben inferir las intersecciones entre las calles, calcular las distancias entre las mismas y repetir lo anterior para todo el grafo, que de tener un tamaño considerable, sería una tarea virtualmente imposible para realizar de forma manual o realizarlo en una sola ejecución ya que los recursos de hardware lo imposibilitan.

En el caso de PgRouting el funcionamiento de la herramienta de cargado se encuentra totalmente encapsulada en la herramienta *osm2pgrouting*. La misma comienza por analizar todas las entradas XML para luego realizar el mapeo de nodos a intersecciones, creando las tablas necesarias en la base de datos con los tipos correspondientes.

III-C1. Red de transporte uruguaya: En específico, la fuente utilizada en el presente informe tiene un peso de 636 MB al persistirla en un formato XML. Representando la red de transporte del territorio uruguayo mediante 3141961 nodos (puntos con latitud y longitud), 178031 caminos, 1041287 etiquetas y 134876 Intersecciones entre calles.

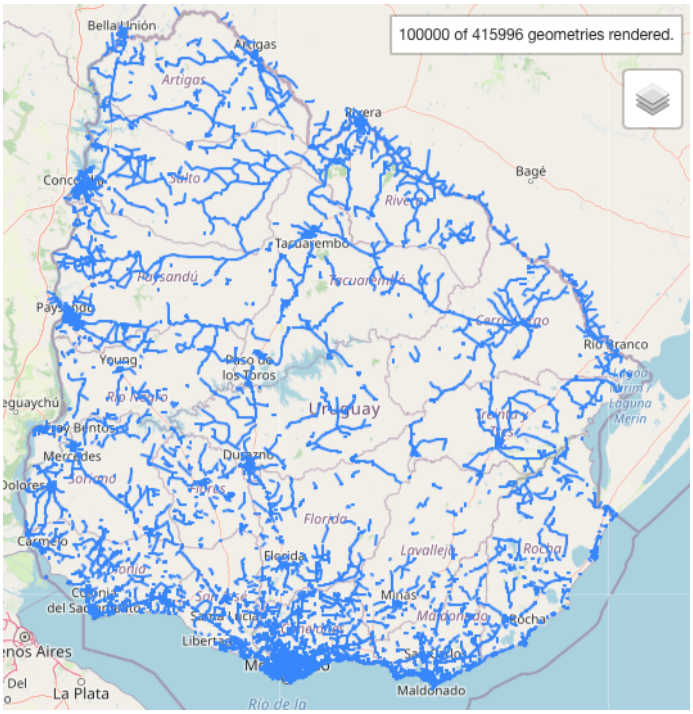

Figura 1: 24 % de la red de transporte utilizada para la experimentación.

La Figura 1 presenta el resultado de visualizar el **24 %** de los vías disponibles en la base de datos relacional.

III-D. Entorno de pruebas

Con el fin de replicar los experimentos del articulo de la forma mas verosímil posible, se configura un máquina virtual con especificaciones similares a las que se plantean en el artículo fuente. Las especificaciones de hardware se listan a continuación:

- **Procesador:** Intel Core I7-8550U (4 Núcleos asignados)
- **RAM:** 6 GB RAM
- **Disco Duro:** 30 GB SSD M2
- **SO:** Ubuntu 20.04

Configurar un máquina virtual permite recabar resultados fidedignos de los experimentos. Al ser un sistema operativo limpio se minimiza la probabilidad de fuga de recursos hacia otros procesos que no formen parte de las pruebas.

Respecto a las versiones de motores de base de datos utilizados, para PostgresSQL se utiliza la versión 13.3 y para Neo4j la versión 4.3.2 en su variante Empresarial (*Enterprise*).

Para la evaluación se utiliza un sistema de evaluación de performance desarrollado por el equipo específicamente para esta investigación [13]. El mismo contiene dos interfaces unificadas para la obtención del camino más corto entre dos nodos dados, una para conectar con la base de datos de Neo4j y otra para conectar con la base de datos de PostgreSQL.

[13]https://gitlab.fing.edu.uy/guillermo.ripa/bdnr-final

Su funcionalidad principal es activar la corrida de un conjutno de experimentación dada una base objetivo, un algoritmo de camino más corto y la cantidad de hilos concurrentes a utilizar. Una vez invocada la misma realiza las conexiones a la base de datos especificada y realiza las pruebas de desempeño para la base de datos en frío y luego repite los experimentos para la base de datos en caliente. Una vez finalizados los experimentos la misma persiste los resultados obtenidos para cada camino así como el tiempo de ejecución requerido para cada uno, para cada etapa y para la ejecución en general en formato CSV. Así mismo, lleva un registro en paralelo del consumo de memoria del sistema, con la posibilidad de especificarle un proceso en particular (el proceso de las bases de datos) para realizar su seguimiento.

En el mismo repositorio se encuentran también las secuencia de comandos utilizadas para la visualización de información.

III-E. Plan de pruebas

El plan de pruebas planteado tiene como meta replicar los experimentos realizados en la publicación académica base.

Una variante del plan realizado en el articulo es que también se ejecutaron pruebas con el algoritmo de enrutamiento A*, este algoritmo no estaba disponible al momento de la publicación del articulo por lo que no se incluyo en los estudios, pero dado que el articulo abarca una problemática global como el camino mas corto y no esta fuertemente atado al algoritmo de Dijkstra, se considero de interés incluir A* en las pruebas.

El plan consiste en tomar 190 pares de puntos al azar y calcular la ruta del camino más corto para cada par, manteniendo este conjunto de pares a lo largo de la evaluación. Para cada ejecución se cronometrara su tiempo y uso de memoria, tanto es su *cold run* como en su *hot run*. Una particularidad propia del plan es que se prueba la ejecución variando el numero de conexiones concurrentes entre 1 y 16 y los experimentos se replican de forma exacta tanto para el algoritmo Dijkstra como para su variante A*. Para el caso de Neo4j se establecieron ajustes de desempeño para mejorar el comportamiento de las consultas concurrentes y optimizar el uso de cache, dichos ajustes se trataran en detalle en la sección Ajustes de desempeño.

En la Figura 4 se puede visualizar una relación fuertemente lineal entre la cantidad de nodos de los caminos analizados y su distancia total. Abarcando un extendido universo de caminos.

También se incluyen en los listados 1 y 2 las consultas utilizadas para obtener los caminos más cortos utilizando el algoritmo de dijksta. Las mismas son muy similares para el algoritmo A*.

Listado 1 (Dijkstra) Consulta de camino más corto para PostgreSQL

```
1  SELECT * FROM pgr_dijkstra('
2              SELECT gid as id,
3                     source_osm::bigint as source,
4                     target_osm::bigint as target,
5                     cost::double precision as cost
6                    FROM ways',
7              source_id, target_id, false);
```

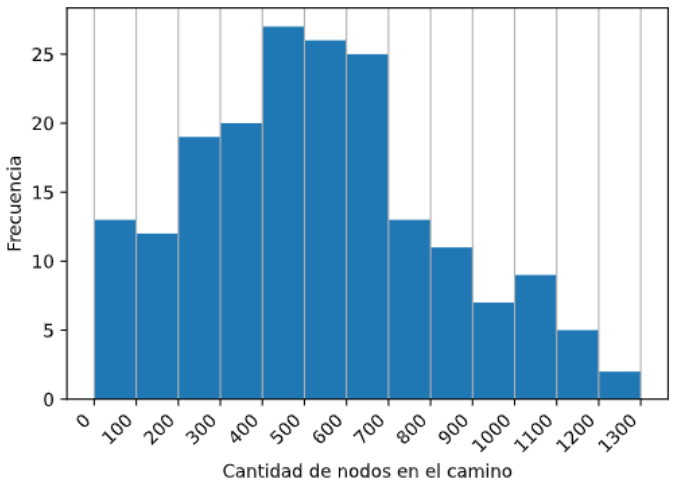

Figura 2: Histograma de distribución de la cantidad de nodos en los caminos más cortos entre pares.

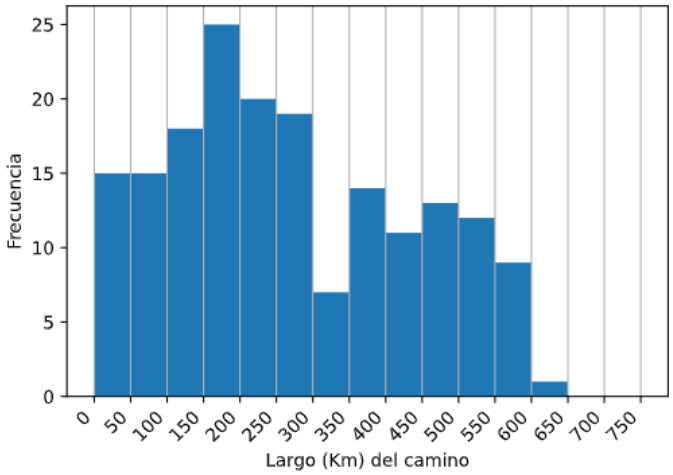

Figura 3: Histograma distancia en km de los caminos más cortos entre pares.

Listado 2 (Dijkstra) Consulta de camino más corto para Neo4j

```
MATCH (n:Intersection)
   WHERE n.node_osm_id = $source_gid
MATCH (p:Intersection)
   WHERE p.node_osm_id = $target_gid
CALL apoc.algo.dijkstra(n,p,'ROUTE','distance')
YIELD path AS j
RETURN j
```

III-F. Ajustes de desempeño

Tal y como lo muestra en el articulo, se necesitan algunos ajustes de desempeño en el caso de Neo4j para lograr mejores resultados en términos de tiempo. En dicho articulo se prueba el uso de 4 tipos de cache siendo estas: *gcr*, *soft*, *weak* y *strong*. Al indagar en la documentación oficial de Neo4j no se encuentran referencias hacia las mismas, por lo que el equipo concluyó que dichos parámetros de configuración fueron discontinuados. Sin embargo existe una configuración

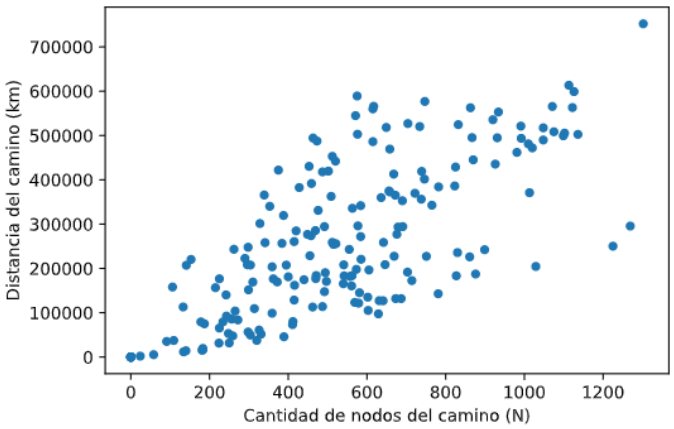

Figura 4: Relación entre distancia total y cantidad de nodos entre caminos.

de Neo4j que permite lograr el mismo resultado, dicha configuración permite realizar una pre-carga de la base de datos en memoria cache según los nodos que se correspondan con la expresión regular del argumento $warmup.preload.allowlist$, si este parámetro se deja con el valor $.*$ (o no se le da un valor especifico), la base tendrá almacenado todo el grafo en memoria chache, y esto es igual que lo que lograba mediante el tipo de cache *strong*, si se le especifica una expresión regular un poco menos abarcativa replicaría el comportamiento de la cache *weak* y de esta manera se pueden replicar los 4 tipos de cache de las versiones iniciales de Neo4j de una forma más exacta, especificando que tipos de nodos son de interés para tener en memoria cache.

Para la pre-carga del grafo anteriormente mencionada se deben cumplir (además de lo anteriormente mencionados) dos requerimientos más. El primero de ellos implica tener la suficiente Memoria RAM para poder mantener el grafo entero en memoria (dicho requerimiento es alcanzado ya que la base de datos pesa aproximadamente 1.8 GB y la VM cuenta con 6GB de RAM), y el segundo es que el tamaño de página de cache configurada en Neo4j debe ser lo suficientemente grande para poder almacenar el grafo en memoria, esto ultimo se soluciona modificando el parámetro $dbms.memory.pagecache.size$ y colocándole un valor por encima del que pesa la base de datos entera (3GB para el conjunto de datos presentado).

Otros ajustes de desempeño fueron evaluados tales como cambiar el *heap_size* por defecto de Neo4j. Configurar un gran tamaño de pila es útil en el caso de que de manera concurrente o en un lapso de tiempo muy corto se realice la misma consulta a la base de datos, por tanto si se tiene un pila grande las consultas y sus resultados se almacenaran en lo que se llama *heap_memory* (por si se realiza la misma consulta en un corto plazo). Esto no es útil en la realidad planteada ya que en el plan de pruebas se hacen consultas concurrentes pero tomando en cuenta 190 pares de puntos distintos. Por tanto ninguna consulta se va a repetir haciendo inútil los datos que se tiene en la *heap_memory*.

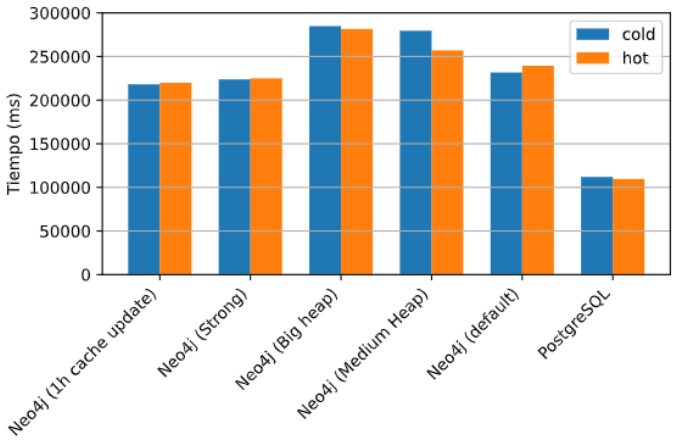

Figura 5: Duración de tiempos de prueba en frío y caliente para Neo4j por parámetros de memoria.

A la vista de los resultados presentados en la Figura 5 es visible que tener un heap_cache grande no es un opción viable ya que además de lo mencionado recientemente tiene el problema de que reserva memoria RAM para un uso que no es aprovechado, teniendo un impacto negativo en términos de tiempos.

Dados los resultados de experimentos para mejorar el rendimiento se decidió tomar la configuración a continuación de Neo4j para las pruebas de la sección anterior.

Listado 3 Configuración de Neo4j

```
dbms.memory.pagecache.warmup.enable=true
dbms.memory.pagecache.warmup.preload=true
dbms.memory.pagecache.warmup.profile.interval=1h
dbms.memory.pagecache.size=3g
```

La configuración final también incluye la propiedad $warmup.profile.interval$ con valor una hora, esto sirve para que las tasas de refresco de cache sean cada una hora, de esa manera evitando que se actualice el cache innecesariamente.

IV. Resultados de la investigación

El artículo original plantea expectativas positivas a corto plazo para la base de datos de grafos Neo4j como motor de enrutamiento y por el contrario plantea de las soluciones relacionales cuentan con poca capacidad de mejora en este aspecto. Ambas hipótesis se basan en el mismo concepto. Como motor de enrutamiento, Neo4j parece ser una solución más prometedora porque, más allá haber sido desarrollada para casos de uso de recorridas locales del grafo, sus funcionalidades para el cálculo del camino más corto están atadas directamente a la velocidad con la que Neo4j logra recorrer el grafo. PgRouting, por su parte, sólo depende de la infraestructura de PostgresSQL para la lectura de las aristas, mientras que el cálculo de rutas se realiza de forma totalmente independiente, obteniendo así una base de tiempo de la cual no tiene control.

Los experimentos realizados muestran que algunos de estos razonamientos han sido sustentados por el correr del tiempo. A pesar de esto, Neo4j muestra no ser la solución ideal para todo tipo de problemas.

Este informe no presenta comparaciones valor a valor con el artículo original debido a que la diferencia en tecnologías y topologías del mapa le quitan su valor referencial. De todos modos, los resultados originales se presentan al final del informe.

En una primera instancia se evalúan los resultados relacionados al algoritmo de camino más corto Dijkstra dado que fue el usado en la publicación original. De todos modos, dada la existencia de la implementación del algoritmo A* en ambas tecnologías, también se presentan los resultados de esta segunda iteración.

Como se muestra en la Figura 6 el uso de memoria para el caso de Neo4j se mantiene constante, con una variación máxima del **10 %** independientemente de las consultas realizadas ya que la totalidad del mapa se encuentra cargado en memoria. Mientras tanto, el uso de memoria de PostgreSQL se presenta levemente afectado por la concurrencia de conexiones con una diferencia del **40 %** entre los pedidos con un único hilo y con dieciséis pero utilizando **54 %** de la memoria en relación a Neo4j en la diferencia más pequeña.

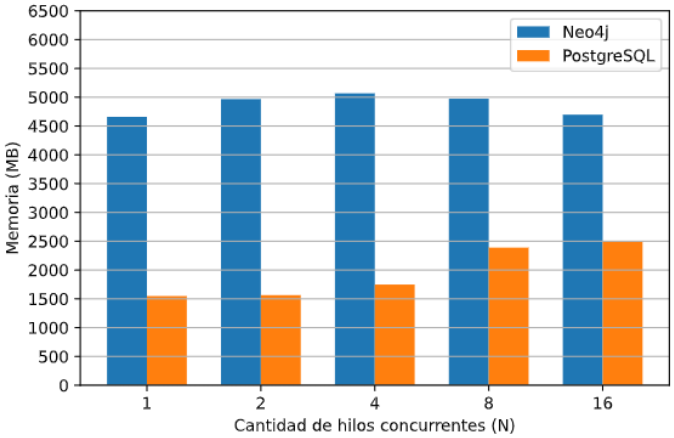

Figura 6: (Dijkstra) Uso máximo de memoria por número de hilos.

La Figura 7 presenta la primer sorpresa hallada por el equipo. La búsqueda de los 190 caminos resulta drásticamente más lenta en la arquitectura de grafos de Neo4j en comparación con la arquitectura relacional, utilizando el algoritmo de Dijkstra. De todos modos, en ambos casos se ven mejoras una vez "caliente" la base de datos y en específico la base de datos de grafos logra aprovechar de mejor manera la concurrencia de hilos. El artículo original presenta resultados donde la arquitectura de grafos se superaba en desempeño a la arquitectura relacional mientras que ambas mejoraban en relación a los hilos concurrentes. El equipo cree que esta mejora en relación a la cantidad de hilos se puede deber a la plataforma de pruebas desarrollada para el presente informe,

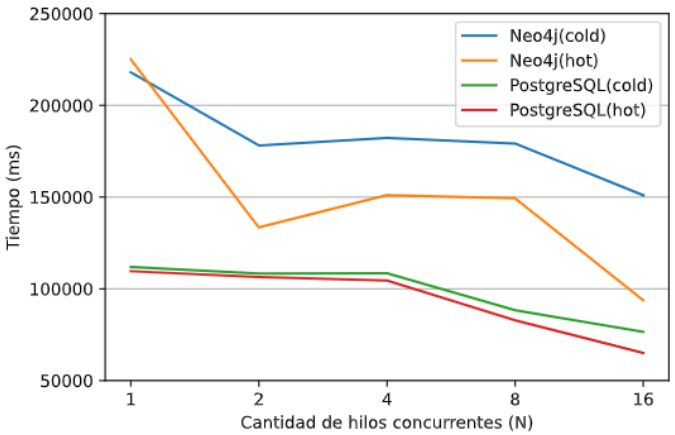

Figura 7: (Dijkstra) Tiempo requerido para finalizar experimentos en frío y caliente para Neo4j y PostgreSQL.

ya que el artículo original realiza las pruebas en base a la plataforma JMeter [14] y el equipo utilizó la biblioteca de Neo4j para Python para realizar los pedidos y la biblioteca de hilos de Python [15] para procesarlos pudiendo generar gastos generales al procesar todas las respuestas en formato JSON en distintos hilos pero en un mismo proceso.

Por último, al evaluar los primeros 20 pedidos a ambas bases se detecta una constancia luego del primer par de consultas en la arquitectura relacional la cuál no está presente en las pruebas de grafos. De todos modos estos datos efectivamente concuerdan con los datos presentados en el artículo original. Tanto la Figura 10 como en la Figura 12 se muestra el mismo comportamiento inestable para los tiempos de Neo4j y constantes para PostgreSQL con la diferencia de una mayor velocidad en el "calentamiento" de las bases en las pruebas actuales.

Una vez analizados los resultados del algoritmo de Dijkstra el equipo incursionó en al investigación del nuevo algoritmo A* (II-B). Es aquí dónde las hipotesis desarrolladas por Miler et al. cobran nuevo valor. A pesar de que el uso de memoria se mantenga constante en relación a los resultados previos (Figura 8), es posible notar como el algoritmo A* aprovecha de forma remarcable la relación con el recorrido del grafo para reducir el tiempo de ejecución en Neo4j incluso por debajo del resultado del mismo algoritmo desarrollado sobre PostgreSQL.

El algoritmo A* utiliza una heurística relacionada a las distancias entre el nodo evaluado y el nodo destino para reducir el universo de caminos posibles dónde se haya el camino más corto. De esta forma, una vez conocidas las coordenadas de los nodos es menor la cantidad de procesamiento requerido entre etapa y etapa. Como se menciona al principio de la sección, PostgreSQL solo podrá utilizar esta heurística una vez ya leídos todos los nodos posibles, mientras que Neo4j logra reducir la cantidad de grafo a recorrer, utilizando su ventaja arquitectónica y reduciendo así los tiempos de forma dramática.

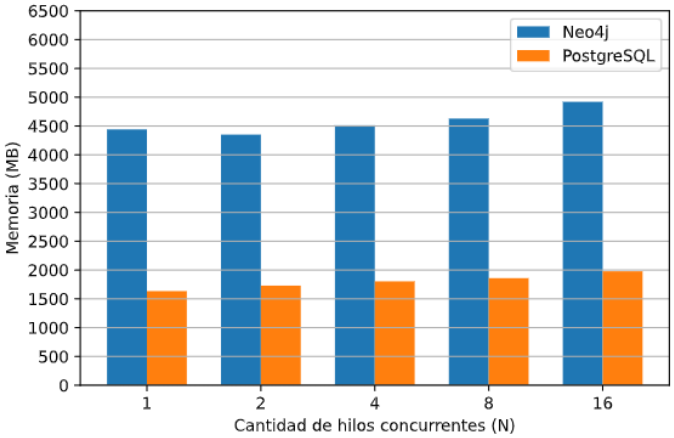

Figura 8: (A*) Uso máximo de memoria por número de hilos.

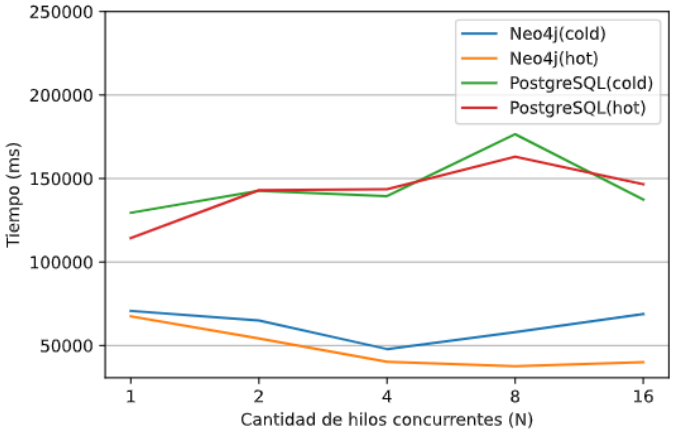

Figura 9: (A*) Tiempo requerido para finalizar experimentos en frío y caliente para Neo4j y PostgreSQL.

Un ejemplo de lo planteado es la Figura 9 que presenta tiempos similares para PostgreSQL utilizando Dijkstra (Figura 7) mientras que Neo4j se presenta por debajo tanto de su contrapartida en Neo4j con Dijkstra como de la prueba de PostgreSQL con A*, en frío como en caliente. La misma relación es también visible en las figuras Figura 10 y Figura 11.

[14] https://jmeter.apache.org
[15] https://docs.python.org/3/library/threading.html

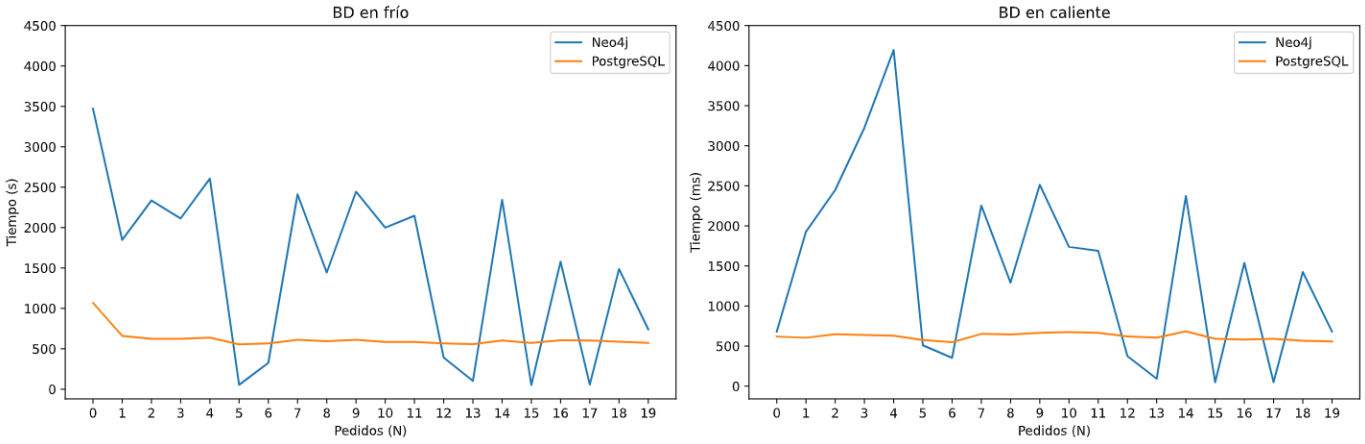

Figura 10: (Dijkstra) Duración de tiempos de prueba en frío y caliente para las primeras 20 consultas con un único thread.

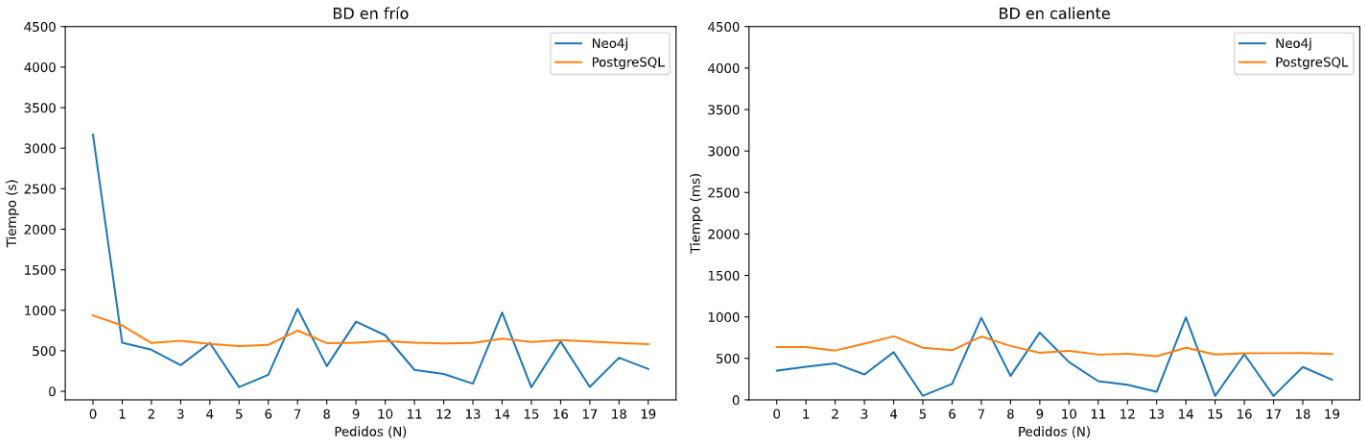

Figura 11: (A*) Duración de tiempos de prueba en frío y caliente para las primeras 20 consultas con un único thread.

V. Conclusión

En primera instancia es destacable el comportamiento que tiene PgRouting al ejecutar el algoritmo de Dijkstra, ya que en todas las pruebas tanto de tiempo como de memoria se posiciona adelante de Neo4j, esto se repite en todos los escenarios sin importar cuantos hilos se manejen. En este punto se tiene un disidencia con el artículo original ya que el mismo posiciona a Neo4j por encima en términos temporales. En uso de memoria ocurre que tanto Neo4j como PgRouting se mantienen relativamente constantes, siendo PgRouting considerablemente menor en uso de memoria RAM.

Al ejecutar la variante A* de Dijkstra el comportamiento cambió notablemente. En este caso Neo4j obtuvo mejores resultados en términos de tiempo, tanto en el escenario de un hilo como en el multi-hilo llegando a ser 4 veces mas rápido que PgRouting para la ejecución del test del plan de pruebas. Esto puede estar dado ya que A* logra aprovechar la relación de velocidad de recorrida del grafo en Neo4j. Ventaja con la que PgRouting no cuenta dada su implementación independiente de PostgreSQL. El uso de memoria en este caso es muy similar al caso de Dijkstra teniendo consumos muy próximos tanto para Neo4j como para PgRouting.

Al igual que en el artículo base, es posible concluir que Neo4j no es un motor de base de datos que resulte óptimo para el recorrido global de un grafo. PostgresSQL fue superior a Neo4j en todas la pruebas usando el algoritmo de Dijkstra. De todas maneras, Neo4j continua siendo la mejor forma de resolver el problema que trata este informe. Esto debido a que con el algoritmo A* Neo4j logra superar ampliamente a PostgreSQL en todos los escenarios excepto el de memoria. Por tanto si la memoria que consumen los procedimientos de Neo4j no son un problema, Neo4j se resulta la mejor alternativa.

De todos modos cabe remarcar que ninguna de las soluciones fue diseñada originalmente como un motor de enrutamiento, por lo que tiene sentido adoptar las mismas mayoritariamente en casos donde se aprovechen las demás funcionalidades de filtrado y agregación así como la utilización de la metadata presente en los sistemas geográficos actuales.

Referencias

[1] D. M. Mario Miler, Dražen Odobašić, "The shortest path algorithm performance comparison in graph and relational database on a transportation network," *Promet - Traffic - Traffico*, vol. 26, pp. 75–82, 02 2013.

[2] A. K. J. M. U. Nitin Gupta, Kapil Mangla, "Applying Dijkstra's Algorithm in Routing Process," 2016. [Online]. Available: https://www.ijntr.org/download_data/IJNTR02050040.pdf

[3] A. Patel, "Amit's Thoughts on Pathfinding," 2020. [Online]. Available: http://theory.stanford.edu/~amitp/GameProgramming/AStarComparison.html

Apéndice

A continuación se incluyen las figuras presentadas en el artículo original a modo de comparación [1]:

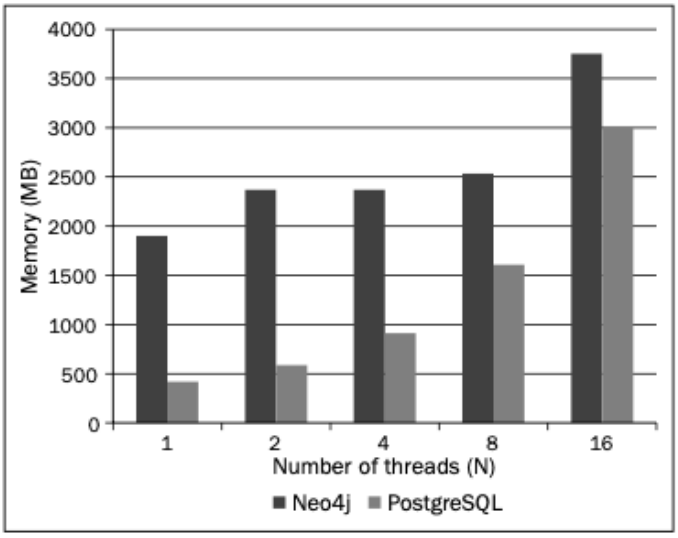

Figura 13: Uso máximo de memoria por número de hilos. [1]

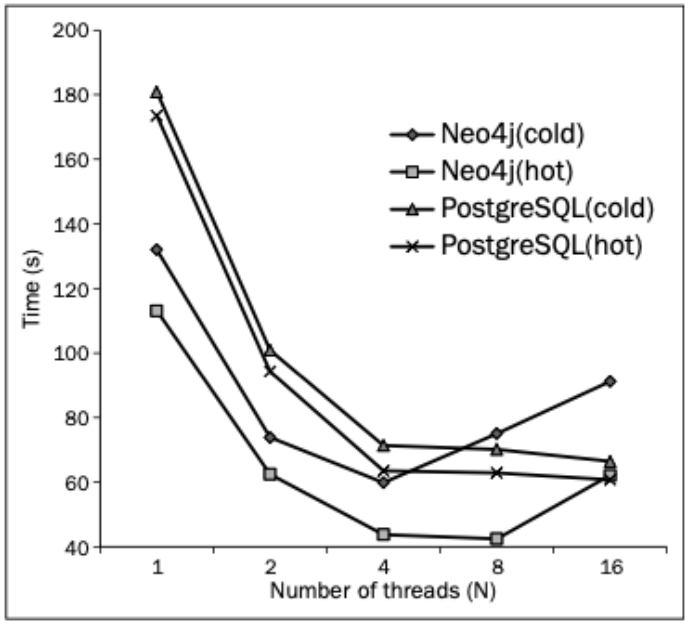

Figura 14: (Dijkstra) Tiempo requerido para finalizar experimentos en frío y caliente para Neo4j y PostgreSQL. [1]

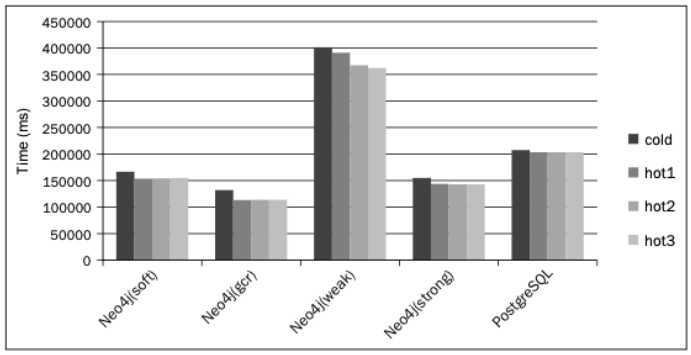

Figura 15: Duración de tiempos de prueba en frío y caliente para Neo4j por parámetros de memoria. [1]

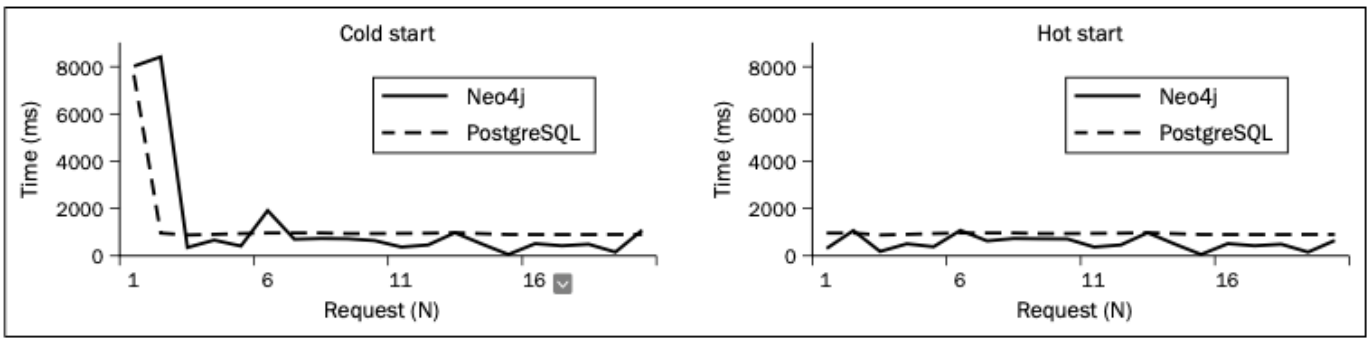

Figura 12: Duración de tiempos de prueba en frío y caliente en las primeras 20 consultas con un único thread. [1]