Administración y Recuperación de Información en Bases de Datos Masivas

Luis Britos, Fernando Kasián, Verónica Ludueña, Diego Olivares,
Marcela Printista, Nora Reyes, Patricia Roggero, Gabriela Romano, Pablo Samat
LIDIC, Dpto. de Informática, Fac. de Cs. Físico Matemáticas y Naturales, Universidad Nacional de San Luis
{*fkasian, vlud, mprinti, nreyes, proggero*}@unsl.edu.ar,
{*ldoorz, olivarestello.diego, gabrielasindaromano, samatpablo*}@gmail.com
Karina Figueroa
Universidad Michoacana de San Nicolás de Hidalgo, México
karina@fismat.umich.mx
Claudia Deco
Facultad de Ciencias Exactas, Ingeniería y Agrimensura, Universidad Nacional de Rosario
deco@fceia.unr.edu.ar

Resumen

Como se ha vuelto evidente durante los últimos tiempos, debido al uso masivo de dispositivos electrónicos, la cantidad de datos multimedia disponibles crece exponencialmente. Estos datos por su naturaleza ya son digitales, aunque sean de diversos tipos, y se acumulan en grandes repositorios de tipos de datos no estructurados. En particular, estos repositorios son difíciles de administrar bajo el modelo relacional de base de datos, por lo que para su administración y recuperación se hacen necesarias herramientas particulares, porque deben modelizarse sobre otros tipos de base de datos que no son las tradicionales.

Más aún, si consideramos el volumen de datos no estructurados que deben administrarse, se hace aún más evidente la necesidad de considerar nuevos modelos y herramientas para su administración y su indexación, considerando las operaciones de interés, porque es necesario resolverlas de manera eficiente, considerando la existencia de la jerarquía de memorias y la posibilidad de hacer uso de computación de alto desempeño.

Palabras Claves: *bases de datos masivas, computación de alto desempeño, recuperación de información.*

1. Contexto

La línea de investigación "Recuperación de Datos e Información" se enmarca dentro del Proyecto Consolidado 3-03-2018 de la Universidad Nacional de San Luis (UNSL - Res. CS 126/18) y en el Programa de Incentivos (Código 22/F834): "Tecnologías Avanzadas Aplicadas al Procesamiento de Datos Masivos", se desarrolla en el Laboratorio de Investigación y Desarrollo en Inteligencia Computacional (LIDIC) de la UNSL, finalizado en 2022. Actualmente se ha realizado una nueva presentación.

Esta línea cuenta con 4 docentes-investigadores y una estudiante de maestría y tiene como objetivo el desarrollo de herramientas eficientes que permitan no sólo administrar grandes bases de datos que almacenan datos no estructurados, sino también poder recuperar datos e información sobre ellas. Con este propósito se analizan nuevas técnicas que provean una buena interacción con el usuario y se desarrollan nuevas estructuras de datos capaces de indexar un gran volumen de datos no estructurados, que permitan resolver eficientemente los tipos de consultas de interés sobre estos tipos de datos (texto, secuencias de ADN, huellas digitales, audio, vídeo, etc.), que sean escalables y sean adecuadas para memorias jerárquicas. Para mayor eficiencia se pueden aplicar técnicas de computación de alto desempeño (HPC).

2. Introducción

Debido al uso masivo de internet, a la gran disponibilidad de dispositivos electrónicos y a la evolución de las tecnologías de información y comunicación, se están generando una cantidad masiva de datos digitales. Además, por provenir de fuentes muy diversas, los tipos de datos generados son también muy variados. En este contexto de problemas de "big data" (porque aparecen sus dos características importantes, que son: el volumen de datos y variedad de los mismos) se hace imposible restringir las consultas a búsquedas sobre datos estructurados tradicionales, porque obligaría a representar una visión parcial del problema. Entonces, para no perder información relevante, es necesario redefinir las

técnicas de procesamiento, análisis y obtención de información útil y formular nuevas metodologías.

Un enfoque significativo para sistemas de recuperación de información es *la búsqueda basada en contenidos*, donde se usa el dato no estructurado en sí mismo para describir lo que se está buscando. En este escenario los tipos de búsqueda más generales y aplicables son las búsquedas por similitud y para resolverlas eficientemente, es necesario utilizar *métodos de acceso* o *índices métricos* [3].

Por lo tanto, al considerar grandes cantidades de datos no estructurados y ser necesario responder consultas de recuperación de información, se pueden utilizar índices métricos para lograr eficiencia en las respuestas. Por otra parte, cuando se consideran aplicaciones reales, estos índices además de eficientes deben permitir actualizaciones (altas y bajas de elementos) y ser escalables; es decir, que su desempeño no se vea afectado muy negativamente a medida que crece el volumen de datos a indexar.

En este escenario, los *espacios métricos*, compuestos por un *universo* $\mathcal{U}$ de objetos y una *función de distancia* $d : \mathcal{U} \times \mathcal{U} \longrightarrow \mathbb{R}^+$, resultan adecuados para modelizar las búsquedas por similitud. La función d mide la disimilitud entre dos objetos y cumple con las propiedades de reflexividad, positividad estricta, simetría y desigualdad triangular.

Entre las búsquedas por similitud, las más utilizadas son: la *búsqueda por rango* y la *búsqueda de los k vecinos más cercanos*. Una de las principales ventajas de este modelo es que, además de brindar un marco formal, es independiente del dominio de la aplicación. Al considerar una base de datos $S \subseteq \mathcal{U}$ tal que $\mid \mathcal{S} \mid = n$, cualquier consulta se resuelve de manera trivial calculando n evaluaciones de distancia. El cálculo de distancia sobre algunos tipos de datos no estructurados puede ser muy costoso (por ej.: comparación de huellas digitales). Por lo tanto, un objetivo importante es aprovechar los índices para ahorrar cálculos de distancia durante las búsquedas. Así, en la mayoría de las aplicaciones sobre datos masivos no es posible prescindir de un índice y resolver una consulta comparando el objeto de interés contra toda la base de datos (búsqueda exhaustiva). Para construir un índice se preprocesa la base de datos S, si se conocen de antemano sus elementos, sino se debe crear incrementalmente.

Sin embargo, en muchos casos es probable que la base de datos, el índice, o ambos, deban alojarse en memoria secundaria, porque no caben en memoria principal. Como se sabe, esto puede incrementar mucho los costos de las búsquedas, porque las operaciones de E/S en memoria secundaria suelen ser muy costosas. Por lo tanto, para lograr eficiencia en estos casos, no basta con realizar pocos cálculos de d sino que también se debe minimizar el número de operaciones de E/S. Una solución adecuada en esta situación debe considerar el nivel de la jerarquía de memorias sobre la que se trabaja, utilzar técnicas de computación de alto desempeño (HPC) y, en algunos casos, admitir respuestas no exactas [5].

En esta línea, se busca desarrollar nuevas herramientas que soporten la interacción con el usuario, diseñar índices que permitan la manipulación eficiente de grandes volúmenes de datos no estructurados y faciliten la realización de diferentes tipos de consultas, para lograr recuperación de información eficiente sobre conjuntos masivos de datos. De esta manera, se espera contribuir al desarrollo de soluciones a problemas de big data para aplicaciones reales.

3. Líneas de Investigación

Dado que en esta línea se pretende contribuir a distintos aspectos de los sistemas de recuperación de información sobre grandes volúmenes de datos no estructurados, se ha considerado inicialmente optimizar índices existentes, diseñar nuevos índices, resolver distintas clases de consultas, y lograr eficiencia y escalabilidad para grandes volúmenes de datos; para luego incorporarlos en un sistema administrador para este tipo de bases de datos.

Métodos de Acceso Métricos

Como se ha mencionado, el modelo para estas bases de datos masivas que contienen datos no estructurados son las *bases de datos métricas*, en las cuales interesa responder a consultas por similitud. Para acelerar la respuesta a estas consultas, minimizando los costos involucrados en el proceso, se utilizan métodos de acceso métricos (MAMs) [3].

En general, los MAMs sacan partido de las distancias almacenadas en el índice y aprovechan que la distancia satisface la propiedad de desigualdad triangular, para ahorrar cálculos de distancia y también tiempo. Habitualmente, los MAMs almacenan algunas distancias precalculadas o referencias obtenidas al comparar los elementos de la base de datos respecto de un conjunto de objetos distinguidos llamados *centros*, *pivotes* o *permutantes*.

Durante las búsquedas es posible estimar la distancia entre cualquier objeto de consulta q y los elementos de la base de datos, aprovechando las distancias o referencias almacenadas y la desigualdad

triangular. Así, es posible determinar qué objetos no son relevantes y evitar cálculos de d.

Cuando se diseñan MAMs, se deben considerar varios aspectos: dinamismo (que soporten altas y bajas), en qué nivel de la jerarquía de memorias deben almacenarse, si es posible aplicar técnicas de HPC para mejorar los tiempos de respuesta, si son de respuesta exacta o de respuesta aproximada y la dimensionalidad del espacio métrico considerado.

Por el volumen de los datos con los que se trabaja, se debe considerar que posiblemente los índices no puedan almacenarse en memoria principal. En general, se considera directamente su almacenamiento en memoria secundaria y sólo se dispone de memoria principal para mantener información muy útil que permita determinar que partes del índice deben recuperarse desde el disco. Por ello, para lograr eficiencia en las búsquedas, se debe minimizar el número de cálculos de distancia y de operaciones de E/S.

Así, una parte de esta línea se dedica a diseñar MAMs, preferentemente dinámicos y adaptados para memoria secundaria, cuyo desempeño en las búsquedas sea adecuado para las aplicaciones de interés. Entre los índices que se han desarrollado se encuentran algunos completamente dinámicos y especialmente diseñados para trabajar sobre grandes volúmenes de datos, basados en la *Lista de Clusters*[1] por (*LC*) [2]. Uno de ellos, el *Conjunto Dinámico de Clusters* (*DSC*) [9], que tiene una buena ocupación de página y operaciones eficientes tanto en cálculos de distancia como en operaciones de E/S, logra un buen desempeño en espacios de alta dimensión. El *DSC* mantiene los clusters en memoria secundaria y organiza los centros de clusters en un *DSAT* [9] en memoria principal. La información en el *DSAT* permite bajar los costos en cálculos de distancia, y mantener bajos los costos de E/S.

Sin embargo, cada inserción en un *DSC* debe localizar el cluster donde se insertará el nuevo elemento. Luego, se lee el cluster desde el disco y se verifica si el nuevo objeto podría ser un mejor centro para el cluster; haciendo, si es posible, el cluster más compacto. Finalmente, se graba nuevamente el cluster en el disco y se actualiza el *DSAT* de centros si el nuevo objeto se transformó en el nuevo centro. Esta operación, además de los cálculos de distancia necesarios, realiza pocas operaciones de E/S en el disco. Para mejorar más respecto de la E/S se está analizando una variante que en lugar de insertar los elementos en el índice a medida que lleguen, demore la inserción de cada nuevo objeto a un cluster hasta tener varios elementos y determinar un mejor agrupamiento de los mismos. Así, se mejoran los costos de búsqueda, al lograr clusters aún más compactos y se asegura una total ocupación de la página del disco, achicando el tamaño del archivo y reduciendo los tiempos de acceso. Esta opción, podría reducir significativamente el costo de construcción, por amortizar el costo de la escritura de *un* cluster en disco, luego de varias inserciones.

Así, se ha diseñado el *Buffered On Line Dynamic Set of Clusters* (*BOLDSC*), una variante del *DSC*, que agrega un espacio en memoria para mantener los objetos que son insertados (bolsa), además de un índice auxiliar (de pivotes) sobre estos elementos. Como pivotes se eligen elementos de la bolsa, y cuando ésta se llena, se selecciona el pivote que necesita el menor radio de cobertura para encerrar la cantidad de elementos que caben en un cluster y todos éstos se sacan de la bolsa y se graban en un cluster completo. La grabación de clusters llenos mejora la cantidad de E/S, pero puede provocar que objetos cercanos, que se insertan en diferentes momentos, queden en clusters diferentes, ocasionando degradación en las búsquedas. Por ello, se está estudiando una variante del *DSC* que graba clusters con una cantidad de objetos que está en función de un factor de carga ρ (casi llenos). Como los clusters no estarán llenos, habrá espacio libre en cada cluster. Entonces, al insertar un nuevo objeto se verifica si hay clusters que lo puedan incorporar y si hay se elige al más cercano a él. Se verifica si el nuevo elemento sería un mejor centro para el cluster y si es así, se actualiza el *DSAT* de centros. Si no es posible incorporarlo a un cluster (todos llenos), va a la bolsa y se actualiza el índice de pivotes. Así, los clusters se adaptan para mejorar los costos de búsqueda. Esta variante está en etapa inicial de implementación.

El *DSC* basa su buen desempeño en la calidad de la partición generada por los centros de sus clusters. Así, otro aspecto a considerar es mejorar la poda en estos agrupamientos. Cada cluster en *DSC* está determinado por su centro y su radio. Sin embargo, puede haber "huecos" dentro de ellos. Por esta razón, es posible seleccionar un conjunto de "pivotes" globales que permitan caracterizar las zonas de los clusters dentro de las cuales existan efectivamente elementos [8], manteniendo para cada cluster información sobre la mínima y la máxima distancia a la que cada pivote encuentra elementos de

[1] Usaremos la palabra *cluster* en inglés, en lugar de agrupamiento en espanõl, ya que es de uso habitual en computación.

dicho cluster. Esta información identifica los "huecos" del cluster para que, durante las consultas, se pueda descartar un cluster si la bola de la consulta no posee intersección efectiva con la zona del cluster que realmente contiene elementos de interés. En las búsquedas se recupera desde memoria secundaria cada cluster que intersecta la bola de consulta con centro q y radio r. Al identificar las zonas del cluster que no contienen elementos, si la consulta intersecta al cluster en su zona "hueca" se lo puede descartar y ahorrar cálculos de distancia y lecturas a disco. Esta variante está en etapa avanzada de implementación.

Por otra parte, el *Árbol de Aproximación Espacial Distal* (*DiSAT*) [1], es un índice estático y muy competitivo respecto del estado del arte. Se ha demostrado que la razón principal de su buen desempeño en las búsquedas es por la partición que induce el árbol sobre la base de datos. Por ello, se está analizando utilizar la información de la partición que induce un *DiSAT* sobre la base de datos para obtener una *representación binaria compacta basada en esquemas* y realizar búsquedas por similitud aproximadas, en las que se admite intercambiar precisión en la respuesta por velocidad. Esta representación de secuencia de bits identifica la partición a la que pertenece el objeto y reduce el espacio de almacenamiento del índice. Durante las búsquedas, se puede obtener la representación binaria de la consulta q y compararla eficientemente con las representaciones de los objetos de la base de datos para determinar cuales parecen ser similares a q (sus representaciones deben ser similares entre sí). Luego, la consulta q se debe comparar usando d para determinar si esos elementos realmente son relevantes. Así, aunque podrían no recuperarse todos los objetos relevantes, por la calidad de la partición del DiSAT, se espera disminuir el costo de las búsquedas y obtener buena calidad de respuesta. Este nuevo índice está en etapa de diseño.

Otra manera de acelerar las búsquedas es paralelizar el índice usando técnicas de HPC. Se ha implementado un *DSC* paralelo, que no sólo baje la cantidad de cálculos de distancia y operaciones de E/S necesarias para responder a una consulta, sino resuelva más consultas simultáneas, aprovechando al máximo las operaciones de E/S que se realicen. Esta implementación considera un índice distribuido local. Cada nodo mantiene su propio índice local para los datos que almacena. La implementación se compone de: un broker, una cola de mensajería y los nodos del cluster conteniendo los índices locales. El broker se encarga de envíar los objetos al momento de crear el árbol y al momento de hacer las búsquedas, dejando distintos mensajes en los tópicos correspondientes. Cada nodo lee los objetos del tópico de creación y agrega el objeto al índice local. También lee los objetos de búsqueda del tópico, busca en el índice local y deja los resultados en ese tópico.

Para la comunicación entre el broker y los nodos se utiliza Apache Kafka. Los datos se almacenan en Kafka en forma de tópicos. Al momento de la creación, el broker va a dejar un objeto en el tópico que corresponda, el nodo correspondiente leerá de ese tópico e insertará el elemento en su índice local. Así, se construyen los diferentes índices locales para cada nodo. En el caso de las consultas, el broker será el encargado de leer las respuestas del tópico, unir los resultados, y devolver la respuesta que corresponde a cada consulta. Para identificar qué respuesta corresponde a qué consulta, al realizarla se deja junto con el objeto un mensaje en el tópico con un identificador único. Así, la respuesta irá al tópico que corresponda y el broker será capaz de unir todos los resultados que correspondan al mismo identificador de tópico. Finalmente, para considerar aspectos de portabilidad, aislamiento, escalabilidad y disponibilidad y rendimiento de los nodos de cluster cada componente se ubica en un contenedor ("docker") y corren en un cluster de Kubernetes. Actualmente, se está optimizando y evaluando la implementación.

DBMS para Bases de Datos Multimedia

Como ya se mencionó, las operaciones más comunes sobre bases de datos multimedia son las búsquedas por rango o de k-vecinos más cercanos. Sin embargo, no son las únicas operaciones a considerar en un sistema administrador para bases de datos multimedia [11]. En este caso, se debe incorporar también la operación de *join* por similitud [4].

Las variantes del join por similitud entre dos bases de datos A y B, con $A, B \subseteq \mathcal{U}$, dependen del criterio de similitud Φ utilizado. Cualquiera de ellas devuelve es un conjunto de pares (x, y), $x \in A$ e $y \in B$, tales que se satisface Φ entre x e y. Algunas de las variantes más conocidas son: el join por rango, el join de k-vecinos más cercanos y el join de k pares de vecinos más cercanos [10]. Formalmente, dadas $A, B \subseteq \mathcal{U}$, el *join por similitud* entre A y B es $(A \underset{\Phi}{\bowtie} B) = \{(x, y)/x \in A \wedge y \in B \wedge \Phi(x, y)\}$. Si $A = B$, la operación se denomina *auto-join* [4]. Un DBMS para estas bases de datos, con datos métricos, debe resolver eficientemente las consultas.

PostgreSQL es un DBMS que permite realizar

consultas por similitud sobre algunos atributos. Por ejemplo, si se han creado índices *KNN-GiST* sobre texto, se podrían resolver búsquedas de k-vecinos más cercanos, pero no son aplicables sobre todo tipo de datos métricos. Además, estos índices brindan plantillas sólo para crear *árboles balanceados*, que no son adecuados para espacios métricos generales. Por ello, es importante incorporar en un DBMS la capacidad de manejar distintos tipos de datos no estructurados y las operaciones de interés.

Como las consultas de join son muy costosas, para acelerar los tiempos de respuesta se pueden disminuir los tamaños de los conjuntos obtenidos diversificando las respuestas [12]. El resultado es así un conjunto más pequeño de pares, con respuestas útiles y diversificadas, y en menor tiempo. Otra manera de acelerar la respuesta es dando una respuesta aproximada a la consulta. Por ello, se ha propuesto un algoritmo simple y eficiente para responder consultas de auto-join por similitud de k vecinos más cercanos aproximados, con una razonable precisión en la respuesta [4]. Por lo tanto, se espera lograr que *PostgreSQL* se transforme en un DBMS más aplicable sobre datos multimedia masivos.

4. Resultados

Recientemente, además de una estrategia para obtener buenas listas de candidatos para las búsquedas por similitud usando permutaciones en un índice invertido [5], se ha publicado una manera de acortar dichas listas de candidatos para acelerar las búsquedas [6], considerando una manera acortar las pemutaciones y de evaluar las similitudes ellas [7].

Por otro lado, se está evaluando experimentalmente la versión paralela del índice *DSC* y se espera publicar estos resultados en el corto plazo. Además, se encuentran en etapa de publicación los resultados obtenidos con *BOLDSC* y en etapa de diseño e implementación las otras mejoras al *DSC*.

La extensión de *PostgreSQL* está en la etapa final para ponerla disponible y publicarla.

5. Formación de Recursos

Se están realizando tesis de Maestría: (1) "Estructuras Eficientes sobre Datos Masivos para Búsquedas en Espacios (UNSL) , (2) "Sistema Administrador para Bases de Datos Métricas" (UNSL), (3) "Índices Métricos – Optimización del DSC usando Cortes de Regiones" (UNSJ), (4) "Optimización del BOLDSC por la Mejora de la Densidad y Solapamiento de los Clusters"(UNSJ) y (5) "Representación Basada en Esquemas para Búsquedas por Similitud Eficientes" (UNSL).

Referencias

[1] E. Chávez, V. Ludueña, N. Reyes, and P. Roggero. Faster proximity searching with the distal sat. *Inf. Systems*, 59:15 – 47, 2016.

[2] E. Chávez and G. Navarro. A compact space decomposition for effective metric indexing. *Pattern Recognition Letters*, 26(9):1363–1376, 2005.

[3] L. Chen, Y. Gao, X. Song, Z. Li, X Miao, and C. Jensen. Indexing metric spaces for exact similarity search. *ACM Comput. Surv.*, 55(6), 2022.

[4] S. Ferrada, B. Bustos, and N. Reyes. An efficient algorithm for approximated self-similarity joins in metric spaces. *Inf. Systems*, 91:101510, 2020.

[5] K. Figueroa, N. Reyes, and A. Camarena-Ibarrola. Candidate list obtained from metric inverted index for similarity searching. *Adv. in Comp. Intelligence*, 29–38, 2020. Springer.

[6] K. Figueroa, A. Camarena-Ibarrola,and N. Reyes. Shortening the candidate list for similarity searching using inverted index. *Pattern Recognition*, 89–97, 2021. Springer.

[7] K. Figueroa, A. Camarena-Ibarrola, N. Reyes, R. Paredes, and B. Hernández Martínez. A hybrid approach to boost the permutation index for similarity searching. In *Actas del XXVIII CACIC 2022*, 458–467, 2022.

[8] J. Lokoč, J. Moško, P. Čech, and T. Skopal. On indexing metric spaces using cut-regions. *Inf. Systems*, 43:1–19, 2014.

[9] G. Navarro and N. Reyes. New dynamic metric indices for secondary memory. *Inf. Systems*, 59:48 – 78, 2016.

[10] R. Paredes and N. Reyes. Solving similarity joins and range queries in metric spaces with the list of twin clusters. *JDA*, 7:18–35, 2009.

[11] C. Rong, C. Lin, Y. N. Silva, J. Wang, W. Lu, and X. Du. Fast and scalable distributed set similarity joins for big data analytics. In *2017 IEEE 33rd International Conference on Data Engineering*, pages 1059–1070, 2017.

[12] L. Santos, L. Carvalho, W. Oliveira, A. Traina, and C. Jr. Traina. Diversity in similarity joins. *Similarity Search and Applications*, volume 9371 of *LNCS*, 42–53. Springer, 2015.