Modelos, Algoritmos y Aplicaciones en Búsquedas a Gran Escala

Gabriel H. Tolosa[1,2], Agustín Marrone[1], Andrés Giordano[1], Agustín Gonzalez[1], Tomás Jurán[1,2] , Esteban A. Ríssola[1,3]
{tolosoft, eamarrone, agiordano, agonzalez, tjuran}@unlu.edu.ar
esteban.andres.rissola@usi.ch

[1]Departamento de Ciencias Básicas, Universidad Nacional de Luján
[2]CIDETIC, Universidad Nacional de Luján
[3]Faculty of Informatics, Università della Svizzera italiana

Resumen

La publicación de información digital crece día a día a tasas exponenciales. Esto exige mayores capacidades de hardware a los proveedores de servicios, e impone restricciones a los usuarios en cuanto a la facilidad de acceso. Además, teniendo en cuenta que los usuarios requieren información relevante lo más rápido posible, la alta tasa de aparición de contenido desafía a las herramientas de búsqueda, las cuales deben considerar y manejar eficientemente el tamaño, la complejidad y el dinamismo de las fuentes actuales de información digital.

En el caso del procesamiento de colecciones masivas de documentos, uno de los desafíos en cuanto a la eficiencia está dado por analizar la menor cantidad de documentos posible para satisfacer una consulta. Por otro lado, si los documentos ocurren en tiempo real (flujos) se requieren estrategias eficientes de ruteo hacia los nodos de búsquedas y de indexación incremental.

Estos problemas requieren, en general, procesamiento distribuido, paralelo y algoritmos altamente eficientes. En la mayoría de los casos, la partición del problema y la distribución de la carga de trabajo son aspectos de las estrategias que requieren ser optimizados de acuerdo al problema.

Este trabajo presenta las líneas de investigación en el contexto de los problemas de búsquedas aplicados a datos masivos en colecciones o flujos de documentos. Las propuestas abarcan el estudio, diseño y evaluación de estructuras de datos y algoritmos con énfasis en la eficiencia y el uso racional de los recursos de hardware.

Palabras clave: algoritmos eficientes, motores de búsqueda, estructuras de datos, grandes datos.

Contexto

Esta presentación se encuentra enmarcada en el proyecto de investigación "Estrategias y Algoritmos para Problemas de Búsquedas a Gran Escala" (Disposición CD-CB N° 350/19) del Departamento de Ciencias Básicas (UNLu).

Introducción

La generación y publicación de información digital crece día a día a tasas exponenciales, lo cual impone restricciones a los usuarios en cuanto a la facilidad de acceso. Considerando la necesidad de acceder a información relevante, esta alta tasa de aparición de contenido (de diversas formas) genera la necesidad de contar con herramientas de búsqueda que puedan manejar el tamaño, complejidad y dinamismo de las fuentes de información digital actuales [25].

El área de Recuperación de Información (RI) es una disciplina dentro de la Ciencias de la Computación que trata con la representación, el almacenamiento y la recuperación de información relevante para los usuarios [2]. Desde las últimas décadas, RI ha experimentado un crecimiento importante en cuanto a sus alcances debido a la utilización de muchas de sus técnicas en los motores de búsqueda para la web (*"web search engines"* o WSE), los cuales han impuesto múltiples desafíos a la comunidad de investigación en el tema.

De forma sintética, un WSE es una aplicación distribuida, que se ejecuta en un *cluster* de computadoras y que debe responder a las consultas de los usuarios (*queries*) con estrictas restricciones de tiempo, en general, en unos pocos milise-

gundos. Pero, además, debe: (1) procesar el repositorio de documentos más grande que se conoce, la web, (2) mantener estructuras de datos específicas e (3) implementar sofisticados algoritmos para lograr satisfacer eficaz y eficientemente los requerimientos. Se puede considerar a los WSE como una de las primeras aplicaciones de procesamiento de datos masivos, variados y dinámicos, características de las hoy llamadas aplicaciones de Big Data [13].

Garantizar la efectividad y la eficiencia es primordial para el despliegue práctico de la web y otras aplicaciones que requieren búsquedas (procesamiento) a gran escala. En los últimos años, esta tarea se ha complejizado por el constante crecimiento del tamaño del problema (cantidad de documentos) y requerimientos de los sistemas (y usuarios). Se han incorporado a la disciplina nuevas estrategias y nuevos escenarios. Por ejemplo, en el primer caso, se utilizan técnicas basadas en aprendizaje automático para aprender a clasificar o rankear [22] y, en el segundo, aparecen requerimientos de búsqueda en redes sociales, las cuales poseen como estructura subyacente un grafo masivo [1]. Este ecosistema ofrece oportunidades para que la comunidad de RI investigue nuevas arquitecturas, se revisen los supuestos acerca de los modelos utilizados, se consideren los costos asociados a las soluciones y, en general, se mejore el *tradeoff* entre la eficacia y la eficiencia y puedan escalar a más datos y usuarios. Además, surgen nuevos problemas que requieren soluciones prácticas.

En el caso del procesamiento de colecciones masivas de documentos, uno de los desafíos en cuanto a la eficiencia está dado por analizar la menor cantidad de documentos posible para satisfacer una consulta. La arquitectura interna de un WSE está compuesta, básicamente, por un nodo *Broker* que recibe las consultas de los usuarios y las dirige a *Search Nodes* (*Back-End*) los cuales poseen las estructuras de datos (índices) necesarias para responder eficientemente. En el *Broker* se pueden implementar algoritmos de selección de recursos (para determinar a cuáles *Search Nodes* enviar la consulta), estrategias de ranking (para ofrecer una mejor respuesta a los usuarios) y caching (para disminuir el procesamiento en el *Back-End*), principalmente. La estructura de datos comúnmente utilizada para soportar la recuperación eficiente es el índice invertido. De forma simple, está compuesta por un vocabulario (V) con todos los términos extraídos de los documentos y, por cada uno de éstos, una lista de los documentos donde aparece dicho término junto con información de frecuencia (*posting list*).

El procesamiento de consultas es uno de los desafíos más difíciles de manejar en un sistema de búsquedas debido al constante crecimiento tanto en datos como en usuarios [15]. Además, el tema de la eficiencia es contínuamente identificado como uno de los más importantes en RI [8]. Por lo tanto, en este proyecto se abordan, principalmente, problemas de eficiencia relacionados con búsquedas a gran escala en dos ámbitos puntuales: colecciones y flujos de documentos.

En sentido amplio, la idea general de aumentar la eficiencia en las búsquedas permite procesar mayor cantidad de datos con menos recursos. Por ejemplo, en un datacenter con 25.000 equipos (nodos), una reducción de sólo un 2% en el tiempo de ejecución significa que se requieren utilizar 500 nodos menos, lo que impacta positivamente en el mantenimiento de éstos, el consumo energético y la generación de calor. Esto, además, permite disponer de nodos ociosos para asignar a otras tareas o, simplemente, mantenerlos para mejorar la tolerancia a fallas del sistema completo.

Estos problemas requieren procesamiento distribuido, paralelo y algoritmos altamente eficientes [5]. Para su abordaje, se utilizan plataformas de almacenamiento y procesamiento no tradicionales [13], tales como Hadoop [27] y Spark [29], sobre las cuales se situan capas de software que implementan algoritmos de búsquedas, aprendizaje automático y optimización. En la mayoría de los casos, la partición del problema y la distribución de la carga de trabajo son aspectos de las estrategias que requieren ser optimizados de acuerdo al problema [26].

Líneas de I+D

Las líneas de I+D del grupo se basan en mejorar la eficiencia en la recuperación de información de gran escala y el procesamiento de grandes volúmenes de datos, con énfasis en:

a. Algoritmos para Poda Dinámica

Una de las áreas importantes cuyas soluciones impactan en la eficiencia son los algoritmos para *top-k*. Como se mencionó, se utiliza un índice invertido como estructura de datos el cual puede (o no) residir completamente en memoria principal y sobre el cual se utilizan dos enfoques principales para

el recorrido de las *posting lists* de los términos que forman la consulta: DAAT (Document-at-a-Time) y TAAT (Term-at-a-Time). A partir de un *query* formado por n términos ($q = \{t_1, t_2 \ldots t_n\}$) DAAT recorre las n listas en paralelo intentando determinar en qué momento detener la evaluación sin llegar al final de todas (*early termination* o *dynamic pruning*) garantizando que el conjunto final de respuesta es el definitivo y equivalente a una búsqueda exhaustiva, pero con mucho menos costo computacional. En el caso de TAAT, las listas de los términos se procesan una a la vez, siguiendo la misma idea.

En la actualidad, las dos estrategias predominantes siguen el criterio DAAT y son Maxscore [23] y WAND [3]. En ambos casos, la idea subyacente es contar con un valor umbral (*upper bound*) que permite determinar en qué momento finalizar la evaluación. Sobre esta base se han realizado varias extensiones y propuestas, en algunos casos combinando la estrategia con una estructura de índice particular, como Block-Max [9]. En este sentdio, un avance relevante es el trabajo de Mallia et al. [14] en el cual se propone utilizar bloques de tamaño variable en la estructura de datos. El particionado de las listas en bloques se modela como un problema de optimización, introduciendo una estructura comprimida para representar la información de los bloques. Sus resultados muestran una mejora de la performance de aproximadamente $2x$ respecto del mejor algoritmo hasta el momento (Block-Max-WAND). En el mismo sentido, un muy reciente trabajo [15] muestra que una estrategia de saltos de bloques múltiples (*long skipping*) permite obtener mejoras para casos particulares como *queries* cortos de hasta un 37%. Por el lado de Maxscore, Jian et al. [10] proponen realizar saltos (*skipping*) también sobre las listas de términos *esenciales* y muestran que el algoritmo procesa significativamente menos documentos.

En esta línea se investigan los algoritmos DAAT que forman parte del estado del arte y, adicionalmente, se propone uno nuevo como extensión de MaxScore. En éste, en vez de almacenar sólo un *upperbound* global, se almacena un conjunto de ellos en una estructura similar a una *skip list* [7], dotando al algoritmo de más información para mejorar la eficiencia del procesamiento. Resultados preliminares muestran que la evaluación de documentos en postings se puede reducir hasta un 50% favoreciendo a consultas con términos muy populares. No obstante, también se ha advertido que se requiere una implementación altamente eficiente para compensar el overhead del procesamiento de los *upper-bounds*.

b. Procesamiento Mixto de Posting Lists

Las técnicas de procesamiento de consultas TAAT forman parte de una línea de investigación opacada por la eficiencia de DAAT en grandes bases de datos textuales como la Web, donde éstas últimas permiten evitar analizar un gran número de documentos. Además, TAAT requiere de una cantidad de *acumuladores* de *scores* (puntaje que determina su relevancia) parciales de cada documento hasta poder calcular el *score* final que indica su orden en el ranking.

Sin embargo, como bajo ciertas circunstancias no se presentan estos problemas descritos, en la búsqueda de estrategias más eficientes de procesar consultas, se propone combinar estrategias TAAT con Maxscore, una de las técnicas DAAT previamente mencionadas.

Se utilizan como base la técnica TAAT descrita por Persin [20] (Filtros de Persin) y la versión de Maxscore propuesta por Jonassen & Bratsberg [11], derivada del trabajo de Turtle & Flood [23].

El desarrollo principal propone crear una estructura de datos donde la *posting list* completa está divida en dos secciones: una primera sección con documentos ordenados por la frecuencia del término en el documento y una segunda sección ordenada por identificador de documento.

Luego, generar un algoritmo que recorra esta estructura, procesando la primer sección utilizando los Filtros de Persin y finalmente, si fuera necesario, realizar MaxScore con los documentos restantes, manteniendo los *scores* parciales calculados. De esta manera, se aprovecha la eficiencia de TAAT en colecciones pequeñas y la ventaja de DAAT en colecciones grandes.

Para esto se requiere mantener la sección de documentos ordenados por frecuencia lo suficientemente extensa para resolver la consulta sin utilizar MaxScore, pero lo suficientemente corta para no perder la eficiencia. Parte de esto involucra definir un umbral que divida la *posting list* entre sus dos secciones y cuyo *overhead* no presente un deterioro significativo en la eficiencia del procesamiento. Dado que la generación de la estructura mixta y sus parámetros auxiliares puede realizarse *offline*, el trabajo se enfoca en optimizar el tiempo de procesamiento de la consulta con los algoritmos descriptos.

c. Búsquedas sobre Flujos en Tiempo Real

Complementando los algoritmos de búsqueda mencionados en la línea anterior, otra estrategia para acelerar el proceso está basada en la partición de la colección de documentos en porciones, P, (*shards*) de acuerdo a algún criterio (por ejemplo, temático) de manera tal de enviar las consultas solamente a un número reducido n de nodos ($n << P$) que contengan particiones de la colección que potencialmente pueden satisfacer la consulta. Dichas estrategias son desafiadas cuando los documentos aparecen en flujos en tiempo real como, por ejemplo, las publicaciones en las redes sociales [21]. Un caso paradigmático son las publicaciones en Twitter, en la cual millones de usuarios alrededor del mundo publican "documentos cortos"(*tweets*) desde diferentes tipos de dispositivos (generalmente, móviles), los cuales deben estar disponibles casi de inmediato (segundos) por lo que las estructuras de datos deben soportar un alto dinamismo [4].

Los problemas principales a abordar aquí abarcan tanto la asignación de los documentos *nuevos* a las particiones como la indexación incremental. El primero de los casos puede ser abordado con técnicas de clustering combinadas con ruteo de documentos. Aquí, es necesario analizar (y redefinir) estrategias efectivas para documentos cortos.

En esta linea de investigación se propone combinar la problemática de las búsquedas en tiempo real utilizando una arquitectura basada en búsquedas selectivas donde los criterios de actualización del índice invertidos por partición, las estrategias de caché a implementar y el algoritmo de búsqueda final impactan en la performance que se pretende optimizar (eficiencia y/o efectividad). Los objetivos generales perseguidos son el desarrollo de técnicas y algoritmos para la asignación de flujos de documentos a diferentes particiones de un índice para luego soportar búsquedas sobre un subconjunto de éstas, maximizando la performance (tanto eficiencia como efectividad).

d. Compresión del índice

El tamaño de una colección de documentos influye directamente en el espacio que su representación lógica ocupa en disco condicionando la posibilidad de ubicarlo en memoria principal. Por ello, a gran escala es necesario usar técnicas de compresión sin pérdida. Sin embargo, el hecho de utilizar una estructura comprimida añade un tiempo de descompresión durante la resolución de una consulta, el cual puede ser compensado al transferir menor cantidad de datos [16]. Además, una compresión eficiente permite que más datos se almacenen en memoria principal [31, 16, 6, 12].

En sentido estricto, las listas invertidas que conforman el índice son secuencias de enteros que pueden comprimirse utilizando métodos específicos.

Entre los diversos esquemas de compresión, se pueden mencionar a los libres de parámetro, que comprimen (y descomprimen) cada entero de forma individual (*Unario* y *Gamma* [16] , o *Variable Byte* [6, 12, 30]); y a los adaptativos a lista, que comprimen una secuencia o bloque de enteros. Además, si se tiene en cuenta la relación costo-beneficio entre tamaño resultante y tiempo de descompresión, se destacan *PFor* (y sus variantes) [30, 12, 28] y *Elias-Fano* [24], en sus versiones particionadas [19]. A su vez, otra codificación interesante es *Interpolative* [17], que a pesar de ofrecer el mejor *compression ratio*, posee una velocidad de descompresión ineficiente para tratar el problema [12, 15].

En general, las listas que lo conforman se suelen dividir en particiones (o *chunks*) de 128 elementos cada una [12, 30, 19]. Asimismo, debido a que también pueden producirse ineficiencias de compresión cuando la cantidad de elementos a comprimir es menor al tamaño de bloque [6], suelen utilizarse métodos alternativos tales como *Variable Byte* (eficiente en tiempo) o *Interpolative* (eficiente en espacio).

Con todo este ecosistema de métodos disponibles, sus variantes y sus ventajas y desventajas en esta línea de investigación se propone modelar un esquema multicompresión que combina diversas codificaciones de acuerdo a las propiedades particulares de cada una de las particiones de las listas invertidas, contemplando también una solución de compromiso entre tasa de compresión y velocidad de resolución de consultas. Siguiendo algunos trabajos prelimimares [18] se trabaja en el uso de codificaciones adicionales y de otros tamaños de partición, abordando también el problema del *overhead* requerido.

Resultados y objetivos

La eficiencia sigue siendo un tema de significativa importancia en la comunidad de RI y se lo reco-

noce como uno de los que siguen ofreciendo desafíos importantes [8]. Principalmente, la necesidad de escalar en datos y hardware ofrece múltiples oportunidades para desarrollos científico/tecnológicos.

Por lo tanto, el objetivo principal del proyecto es estudiar, desarrollar, aplicar, validar y transferir modelos, algoritmos y técnicas que aborden el problema de las búsquedas sobre datos masivos, tanto en documentos como en grafos. Se pretende estudiar los problemas mencionados relacionados con técnicas de optimización para aplicaciones de búsqueda a partir de propiedades de los datos. En particular:

- Proponer, diseñar y evaluar variaciones de los algoritmos para poda dinámica para recuperación *top-k* basados en múltiples *upper-bounds*, minimizando la cantidad de postings procesadas.
- Desarrollar una estructura de datos y un algoritmo de recorrido que considere un ordenamiento mixto por identificadores de documentos o por frecuencias de acuerdo a diferentes criterios.
- Desarrollar un esquema de multicompresión de las posting lists que consideren diferentes *codecs* para cada bloque, de acuerdo a propiedades de los datos en éstos.
- Diseñar y evaluar estrategias de indexación distribuida y resolución de consultas para flujos de documentos que ocurren en tiempo real (por ejemplo, publicaciones en redes sociales).

Formación de Recursos Humanos

En el marco de estas líneas de investigación se están dirigiendo tres tesis de Licenciatura en Sistemas de Información (UNLu). Además, asociados al proyecto de investigación hay una estancia de investigación de la Secretaría de CyT (UNLu), una Beca Estímulo a las Vocaciones Científicas (CIN) y una pasantía interna UNLu.

Referencias

[1] N. K. Ahmed, N. Duffield, T. L. Willke, and R. A. Rossi. On sampling from massive graph streams. *Proc. VLDB Endow.*, 10(11):1430–1441, Aug. 2017.

[2] R. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval: The Concepts and Technology Behind Search.* Addison-Wesley Publishing Company, 2nd edition, 2011.

[3] A. Z. Broder, D. Carmel, M. Herscovici, A. Soffer, and J. Zien. Efficient query evaluation using a two-level retrieval process. In *Proc. of the Twelfth Int. Conf. on Information and Knowledge Management*, CIKM '03, pages 426–434. ACM, 2003.

[4] M. Busch, K. Gade, B. Larson, P. Lok, S. Luckenbill, and J. Lin. Earlybird: Real-time search at twitter. In *Proc. of the 28th Int. Conf. on Data Engineering*, ICDE '12. IEEE, 2012.

[5] B. B. Cambazoglu and R. A. Baeza-Yates. Scalability and efficiency challenges in large-scale web search engines. In *Proc. of the Eighth ACM Int. Conf. on Web Search and Data Mining, WSDM*, 2015.

[6] M. Catena, C. Macdonald, and I. Ounis. On inverted index compression for search engine efficiency. In *Proc. of the 36th European Conf. on IR Research on Advances in Information Retrieval - Volume 8416*, page 359–371. ACM, 2014.

[7] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to algorithms.* MIT press, 2009.

[8] J. S. Culpepper, F. Diaz, and M. D. Smucker. Research frontiers in information retrieval: Report from the third strategic workshop on information retrieval in lorne (swirl 2018). *SIGIR Forum*, 52(1):34–90, Aug. 2018.

[9] S. Ding and T. Suel. Faster top-k document retrieval using block-max indexes. In *Proc. of the 34th Int. Conf. on Research and Development in Information Retrieval*, pages 993–1002. ACM, 2011.

[10] K. Jiang and Y. Yang. Faster maxscore query processing with essential list skipping. In X. Luo, J. X. Yu, and Z. Li, editors, *Advanced Data Mining and Applications*, pages 549–559. Int. Publishing, 2014.

[11] S. Jonassen and S. E. Bratsberg. Efficient compressed inverted index skipping for disjunctive text-queries. In *Proc. of the 33rd European*

Conf. on Advances in Information Retrieval - Volume 6611, page 530–542. ACM, 2011.

[12] D. Lemire and L. Boytsov. Decoding billions of integers per second through vectorization. *Softw. Pract. Exper.*, 45(1):1–29, Jan. 2015.

[13] S. Madden. From databases to big data. *IEEE Internet Computing*, 16(3):4–6, 2012.

[14] A. Mallia, G. Ottaviano, E. Porciani, N. Tonellotto, and R. Venturini. Faster blockmax wand with variable-sized blocks. In *Proc. of the 40th Int. Conf. on Research and Development in Information Retrieval*, pages 625–634. ACM, 2017.

[15] A. Mallia and E. Porciani. Faster blockmax wand with longer skipping. In L. Azzopardi, B. Stein, N. Fuhr, P. Mayr, C. Hauff, and D. Hiemstra, editors, *Advances in Information Retrieval*, pages 771–778. Int. Publishing, 2019.

[16] C. D. Manning, P. Raghavan, and H. Schütze. *Introduction to Information Retrieval*. Cambridge University Press, 2008.

[17] A. Moffat and L. Stuiver. Binary interpolative coding for effective index compression. *Inf. Retr.*, 3(1):25–47, July 2000.

[18] G. Ottaviano, N. Tonellotto, and R. Venturini. Optimal space-time tradeoffs for inverted indexes. In *Proc. of the Eighth ACM Int. Conf. on Web Search and Data Mining*, WSDM '15, pages 47–56. ACM, 2015.

[19] G. Ottaviano and R. Venturini. Partitioned elias-fano indexes. In *Proc. of the 37th Int. Conf. on Research & Development in Information Retrieval*, page 273–282. ACM, 2014.

[20] M. Persin. Document filtering for fast ranking. In *Proc. of the 17th Annual Int. Conf. on Research and Development in Information Retrieval*, page 339–348. ACM, 1994.

[21] E. Rissola and G. Tolosa. Improving real time search performance using inverted index entries invalidation strategies. *Journal of Computer Science & Technology*, 16(1), 2016.

[22] N. Tax, S. Bockting, and D. Hiemstra. A cross-benchmark comparison of 87 learning to rank methods. *Information Processing & Management*, 51, 11 2015.

[23] H. Turtle and J. Flood. Query evaluation: Strategies and optimizations. *Inf. Process. Manage.*, 31(6):831–850, Nov. 1995.

[24] S. Vigna. Quasi-succinct indices. In *Proc. of the Sixth ACM Int. Conf. on Web Search and Data Mining*, page 83–92. ACM, 2013.

[25] Y. Wang, L. Wu, L. Luo, Y. Zhang, and G. Dong. Short-term internet search using makes people rely on search engines when facing unknown issues. *PLOS ONE*, 12(4):1–9, 2017.

[26] Y. Wang, L. Wu, L. Luo, Y. Zhang, and G. Dong. Short-term internet search using makes people rely on search engines when facing unknown issues. *PloS one*, 12(4), 2017.

[27] T. White. *Hadoop: The Definitive Guide*. O'Reilly Media, Inc., 1st edition, 2009.

[28] H. Yan, S. Ding, and T. Suel. Inverted index compression and query processing with optimized document ordering. In *Proc. of the 18th Int. Conf. on World Wide Web*, WWW '09, page 401–410. ACM, 2009.

[29] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica. Spark : Cluster Computing with Working Sets. *Proc. of the 2nd USENIX Conf. on Hot topics in cloud computing*, page 10, 2010.

[30] J. Zhang, X. Long, and T. Suel. Performance of compressed inverted list caching in search engines. In *Proc. of the 17th Int. Conf. on World Wide Web*, WWW '08. ACM, 2008.

[31] M. Zukowski, S. Heman, N. Nes, and P. Boncz. Super-scalar ram-cpu cache compression. In *Proc. of the 22nd Int. Conf. on Data Engineering*, ICDE '06, page 59. IEEE, 2006.