UNIVERSIDAD DE CHILE

FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS

DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN

Caché para RDF Stores basado en k^2-tree dinámico y compacto

PROPUESTA DE TEMA DE TESIS PARA OPTAR AL GRADO DE MAGÍSTER EN CIENCIAS MENCIÓN COMPUTACIÓN

Autor:
Cristóbal Miranda Torres

Profesores Guía:
Diego Arroyuelo, Aidan Hogan, Gonzalo Navarro

8 de Noviembre, 2019

UNIVERSIDAD DE CHILE

1 Introducción

La Web Semántica es una iniciativa que se ha venido gestando desde principios de la década del 2000 y que pretende delegar a las máquinas el procesamiento del conocimiento que está inserto en la Web, complementándose al modelo original de esta, que solo utiliza a las máquinas para mostrar contenido a los humanos.

Para lograr este objetivo, la comunidad de la Web Semántica ha desarrollado el modelo Resource Description Framework, abreviadamente RDF, para almacenar relaciones entre los datos. Este es el modelo estándar en el contexto de la Web Semántica y se utiliza para construir cosas más complejas como Web Ontology Language (OWL), que proporciona información semántica aún más enriquecida con el establecimiento de estructuras lógicas sobre conjuntos de triples RDF, permitiendo inferir cosas sobre los datos.

Hoy en día, el volumen de datos con el que se cuenta en las bases de datos de triples RDF es grande y se espera que siga en aumento con los esfuerzos que hay puestos en la Web Semántica. Por ejemplo, una de las mayores gestoras de este tipo de datos es Wikidata, proyecto hermano de Wikipedia, cuyo objetivo principal es abstraer el conocimiento del idioma e integrarse a Wikipedia como motor descriptor de datos de esta, es decir, los datos de Wikipedia ser administrados por Wikidata, que es algo que ya se está haciendo desde hace unos años. Wikidata posee una gigantesca RDF Store que sigue creciendo continuamente gracias al aporte de su comunidad a la que puede unírsele cualquier persona en el mundo.

Este crecimiento en los datos implica más desafíos, entre estos está el ver cómo almacenar los datos y cómo son obtenidos a partir de consultas. También en muchas situaciones y en particular en la de Wikidata ocurre que hay alta concurrencia en las peticiones de resolución de consultas lo que impacta fuertemente en la capacidad de procesamiento de los servidores que se dedican a esto. Es por esto que surge la idea de aplicar métodos de caching en este contexto para alivianar la carga en el procesamiento de las consultas y tomando en cuenta de que hay consultas que ocurren frecuentemente.

En este trabajo se propone el uso de dos tipos de k^2-tree compactos para almacenar triples RDF. Un tipo corresponde a una estructura estática que se utiliza para indexar la totalidad de la base de datos y se reindexa periódicamente para utilizar la nueva información. El otro tipo es un k^2-tree dinámico, que permite la construcción de resultados de las consultas. La aplicación del primer tipo no es novedosa, pero si la del uso como resultados para el caché.

Con la ayuda de la estructura estática, se quiere estudiar el funcionamiento de caching basado en la versión dinámica. Se analizará el uso de la estructura dinámica tanto en el servidor como en el cliente, pues es interesante la aplicación en que el servidor delegue trabajo al cliente para ahorrarse procesamiento.

2 Problema

El problema a abordar no es uno solo, son varios que son provocados por la misma causa que es que el procesamiento de consultas SPARQL es costoso. Algunas consecuencias típicas de esto son que los resultados son erróneos, incompletos o que no se entregan resultados porque ocurren timeouts [8]. Una vía para evitar estas cosas es mediante el escalamiento de hardware, ya sea vertical u horizontal, que es perfectamente razonable, pero puede significar altos costos. Antes de pensar en agregar más hardware conviene estudiar la forma en que están organizados los datos y maximizar mejoras dentro de ese campo.

3 Trabajo Relacionado

El uso de k^2-tree [4] en la Web de Datos o la Web Semántica es algo que viene de hace algún tiempo. Se ha usado para representar RDF Stores con un k^2-tree dinámico y compacto, denominado K2triples+, porque además de tener el k^2-tree usa otras estructuras para disminuir el costo de algunas consultas [1]. El k^2-tree en esa estructura está basado en vectores de bits dinámicos usando la organización de nodos conocida como LOUDS (Level-Ordered Unary Degree Sequence) que cuyo recorrido secuencial en los vectores de bits representa un recorrido del árbol por nivel de profundidad. Esa representación la ocupan para los niveles superiores del árbol y cuando se llega a cierto nivel se comienza a utilizar DACs (Directly Addresable Codes) [5]. Otra estructura para representar RDF Stores, que se basa en un Suffix Array compacto y se llama RDFCSA [3], ocupa más espacio que los K2triples+, pero en cambio mejora los tiempos de las queries.

Muy recientemente se ha desarollado una representación de k^2-tree dinámica que mejora el k^2-tree que se utiliza en K2triples+ en cuanto a tiempo de las operaciones y manteniendo espacio similar. Esta usa una estructura de tries basada en códigos de Morton para recorrer el árbol en las operaciones de búsqueda e inserción [2]. A diferencia de K2triples+, la organización de nodos en vectores de bits es DFUDS (Depth-First Unary Degree Sequence) que al recorrer los bits secuencialmente, lo que se está haciendo es realmente un recorrido en profundidad en el árbol, es decir, al comenzar desde cierto nodo, primero se visitará todo su sub-árbol antes de visitar a su nodo hermano siguiente (si es que lo tiene). Por otra parte, los nodos en esta estructura se guardan en bloques que particionan el k^2-tree y que se conectan por nodos especiales llamados nodos frontera, donde cada nodo frontera tiene asociado un puntero que permite moverse entre bloques. Esta representación puede lograr mejores tiempos de operación que la de K2triples+, que al usar vectores de bits dinámicos las operaciones de rank y select ya no pueden hacerse en tiempo constante sino que ahora son en tiempo $O(\log v / \log \log v)$, con v la cantidad de nodos del árbol.

Con respecto al uso de caché en bases de datos RDF, existe una metodología de planeamiento de joins con programación dinámica (DPccp) [6], etiquetado canónico de consultas en SPARQL para resolver consultas isomorfas y con un controlador de caché que permite diferenciar cuáles son las queries más rentables de ser almacenadas [7]. En cuanto al etiquetado canónico, cada consulta se puede

interpretar como un grafo y si dos consultas tienen representaciones de grafos isomorfos, entonces la respuesta a esas consultas va a ser la misma.

4 Datos

Los datos con los cuales se trabaja son los de Wikidata [9], que almacena una gran cantidad de relaciones RDF y sigue creciendo con esfuerzos colaborativos. En la Figura 1 se muestra cantidad de ediciones por mes desde Octubre del 2012 hasta Agosto de 2019. Se ve que en Marzo de 2019 hubo en total 28.5 millones de ediciones a Wikidata.

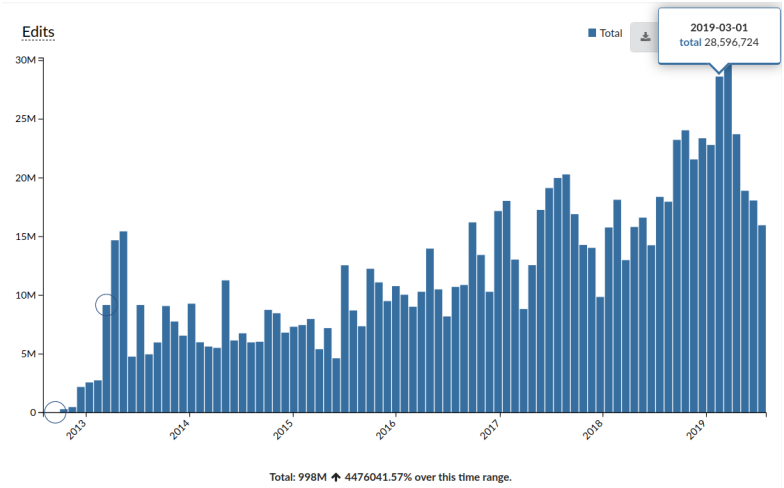

Figura 1: Ediciones mensuales en Wikidata desde Octubre del 2012 hasta Agosto de 2019.[1]

En la Figura 2 se indica la diferencia absoluta en bytes en el tiempo desde 2012 hasta 2019, que significa la suma de las diferencias entre los bytes de la página después y antes de ser editada en valor absoluto. Es decir, en vez de medir cuántas veces se editó, se mide la cantidad de información que se editó. Por ejemplo, se observa que entre Agosto y Septiembre del 2017 hubo 123.8 GB en modificaciones de páginas en total.

Los dumps de Wikidata son, a la fecha (segundo semestre de 2019), de alrededor de 30 GB comprimidos y 500 GB descomprimidos.

[1] `https://stats.wikimedia.org/v2/#/all-projects`

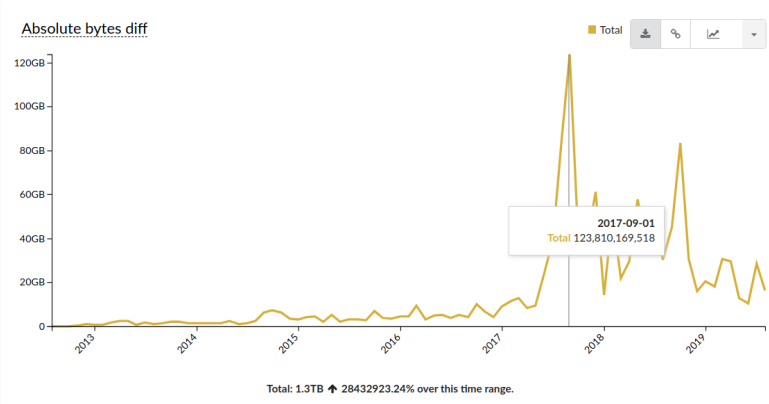

Figura 2: Diferencia absoluta en bytes mensual desde Octubre del 2012 hasta Agosto de 2019.[1]

5 Hipótesis

Con esta propuesta se logrará mejorar los tiempos de respuesta de consultas SPARQL a un caché de una RDF Store, usando el k^2-tree dinámico y compacto con códigos de Morton, descrito en este documento, para indexar resultados de queries en SPARQL dentro del caché, reemplazando al índice utilizado por Papailiou, N et al [7].

6 Preguntas de investigación

1. **Caché compacto vs no caché en el servidor**: Se quiere determinar si es que el caché compacto consigue mejores tiempos de respuesta que no usar el caché, con todo el procesamiento en el servidor y el cliente solo recibiendo los resultados.

2. **Caché compacto vs no caché en el cliente**: Por otro lado, saber cómo se comporta este caché en el cliente, que de no tener los resultados de subconsultas debe pedirlos al servidor y cómo se compara con obtener respuestas directas del servidor.

3. **Caché compacto vs no compacto**: También se quiere conocer cuál es el impacto de utilizar un caché compacto a diferencia de uno que no es compacto.

4. **Transferencia eficiente**: Por último se quiere saber qué tanto afecta en los tiempos de transferencia la representación compacta.

7 Objetivo General

Lo que se quiere de este trabajo es lograr disminuir los tiempos de respuesta de las consultas en SPARQL que se hacen a RDF Stores.

Lograr esto permitiría ayudar a resolver el problema de Wikidata de consultas eficientes sobre los datos estructurados que tiene y que se usan en sus proyectos hermanos, como Wikipedia.

8 Objetivos Específicos

8.1 Uso de caché

En primera instancia se quiere probar empíricamente que el uso del caché con k^2-tree dinámico mejora sustancialmente el tiempo de respuesta de una consulta SPARQL, en dos escenarios distintos. El primer escenario es que el servidor en el que se almacena la base de datos construye completamente la respuesta a la consulta y la envía al cliente. En el segundo escenario, el servidor le entrega al cliente partes con las que puede construir su respuesta.

8.2 Caché compacto

Por otro lado, se quiere mostrar la conveniencia de usar un caché compacto por sobre uno plano, tanto en términos de eficiencia como en el uso de espacio. Siendo lo segundo importante para lo primero, considerando la jerarquía de memoria.

8.3 Alivianar carga en servidor

Se quiere determinar cuándo es mejor darle al cliente la tarea de construir la respuesta, manteniendo un balance entre el trabajo que hace el servidor y el tiempo de respuesta de las consultas.

9 Metodología

9.1 Desarrollo de estructuras

La primera etapa del trabajo consiste en el desarrollo de las estructuras con las que se requiere hacer los experimentos. Se implementará una estructura K2triples+, pero en su versión estática y una estructura muy similar al K2triples+ que en vez de usar el k^2-tree regular se utilizará el que se recorre con códigos de Morton [2].

9.1.1 k^2-tree estático y compacto

La estructura inicial a considerar es un k^2-tree estático en la cual se almacenará todo el contenido del dump más reciente de Wikidata y la idea es reindexar todos esos datos periódicamente. Específicamente esta estructura se representa con vectores de bits y operaciones de rank y select para ir recorriendo el árbol, que tienen complejidad de tiempo constante.

9.1.2 k^2-tree dinámico y compacto

El segundo tipo de k^2-tree es dinámico y en este también se utilizan vectores de bits para representar el árbol, pero se opera de una manera distinta. En este caso no se utilizan operaciones de rank y select para recorrer el árbol. En cambio, se usan códigos de Morton, que codifican los puntos que están representados en el k^2-tree como cadenas binarias, permitiendo recorrerlo como un trie.

9.2 Desarrollo de algoritmos de consulta

Para hacer consultas sobre un k^2-tree, primero hay que parsear el lenguaje de SPARQL. Para esto se pretende buscar la mejor alternativa entre las herramientas existentes para hacer esto. Luego, las instrucciones obtenidas del parsing deben ser traducidas a operaciones en el k^2-tree.

9.3 Construcción de respuestas

Para construir las respuestas de las consultas se pretende ir insertando puntos al k^2-tree dinámico, a medida que se vayan obteniendo del k^2-tree estático.

9.4 Estrategia de caching

Cuando se tienen las respuestas se debe decidir si son cacheadas o no y cómo se mapean las queries a los resultados. Para esto se aplicará una estrategia que es ortogonal al trabajo a ser evaluado acá, pero de todas formas debe ser implementada. La forma más ingenua de hacerlo es cachear todas las respuestas y mapeando todas las queries que son iguales, excepto nombres de variable, al mismo resultado. Otra forma más elaborada es etiquetar las consultas canónicamente y las consultas con las mismas etiquetas asignarles la misma respuesta, que es equivalente a resolver el problema de isomorfismo de grafos, pues que el etiquetado sea canónico significa que se cumpla que dos grafos con la misma etiqueta sean isomorfos y viceversa. Una opción de este etiquetado para este trabajo es usando el algoritmo de etiquetado de grafos Bliss, que primero requiere como input un grafo dirigido con

nodos coloreados y para el contexto RDF se usa un procedimiento existente en la literatura que logra esto satisfactoriamente [7].

En este trabajo se usará el mismo planificador con programación dinámica DPccp y el mismo controlador de caché que utilizan en el trabajo de Papailiou, N et al [7]. En cambio, para el etiquetado de queries se usará el etiquetado sintáctico y canónico propuesto por Salas, J. y Hogan, A. que promete mejor utilización del caché puesto que detecta más queries equivalentes [8].

9.5 Estrategia de joins

Parte del trabajo consiste en estudiar cómo hacer joins entre grafos que sirven para armar una respuesta, hay veces que puede convenir hacerlo en el mismo servidor y hay veces que puede convenir enviar las partes al cliente para que este las arme, alivianando la carga del servidor. Sin embargo, puede haber ocasiones en las que hacer esto último puede resultar demasiado caro en tiempos de respuesta.

Para estudiar este punto, la metodología será hacer experimentos que midan los tiempos de respuesta de consultas, el tiempo de procesamiento en el servidor, el tamaño de procesamiento en el cliente y el tamaño de los datos enviados por la red, todo en cada escenario distinto. Cada escenario se diferencia según el tamaño de lo que se envía por la red, es decir, eso sería una de las variable independientes.

9.6 Estudio según consultas

Es posible clasificar las consultas según cuál es la entidad variable entre sujeto, predicado y objeto de un triple RDF. Además que se tiene en consideración que según este tipo de consulta se verá afectado el tiempo de respuesta, pues se ocupan procedimientos distintos para cada caso. Por lo que también se pretende considerar como variable independiente el tipo de consulta.

10 Resultados Esperados

Se espera que con la metodología descrita anteriormente se pueda, principalmente, satisfacer la hipótesis. También se quiere determinar cuales son los parámetros óptimos con los que funciona el caché para ver la factibilidad de ser usado como aplicación.

Está la posibilidad de que en la práctica no sea conveniente usar estas estructuras, aunque la teoría y otros trabajos anteriores similares indican que si debe ser conveniente. En el caso de tener un resultado negativo va a ser interesante estudiar las razones de ello si es que son teóricas y si son de

implementación se puede iterar en su desarrollo hasta llegar a un resultado positivo o darse cuenta de que si era un asunto teórico.

11 Trabajo Futuro

En el caso de que se tuviera éxito con este trabajo se podrían mejorar ciertos aspectos técnicos, como por ejemplo idear una estrategia para no tener que reindexar los datos o al menos que se pueda hacer a períodos más largos.

Una opción a seguir sería utilizar otras estructuras de índice resultados para probar el caché y encontrar la que tenga mayor ventajas para ese contexto.

También esta solución podría ser extendida a sistemas distribuidos, tanto para replicar como para particionar los datos. Con el fin de ser más tolerante a fallas, disminuir la latencia del usuario, alivianar la carga de las peticiones al caché y tener capacidad arbitraria para manejar el conjunto de datos que va creciendo en el tiempo.

Referencias

[1] Sandra Álvarez-García, Nieves Brisaboa, Javier D. Fernández, Miguel A. Martínez-Prieto, and Gonzalo Navarro. "Compressed vertical partitioning for efficient RDF management". In: *Knowledge and Information Systems* 44.2 (Aug. 2014), pages 439–474 (cited on page 2).

[2] Diego Arroyuelo, Guillermo de Bernardo, Travis Gagie, and Gonzalo Navarro. "Faster Dynamic Compressed d-ary Relations". In: *String Processing and Information Retrieval*. Springer International Publishing, 2019, pages 419–433 (cited on pages 2, 5).

[3] Nieves R. Brisaboa, Ana Cerdeira-Pena, Antonio Fariña, and Gonzalo Navarro. "A Compact RDF Store Using Suffix Arrays". In: *String Processing and Information Retrieval*. Springer International Publishing, 2015, pages 103–115 (cited on page 2).

[4] Nieves R. Brisaboa, Susana Ladra, and Gonzalo Navarro. "Compact representation of Web graphs with extended functionality". In: *Information Systems* 39 (Jan. 2014), pages 152–174 (cited on page 2).

[5] Nieves Brisaboa, Susana Ladra, and Gonzalo Navarro. "DACs: Bringing direct access to variable-length codes". In: *Information Processing & Management* 49 (Jan. 2013), pages 392–404 (cited on page 2).

[6] Guido Moerkotte et al. "Analysis of Two Existing and One New Dynamic Programming Algorithm for the Generation of Optimal Bushy Join Trees without Cross Products". In: *Proceedings of the 32nd International Conference on Very Large Data Bases (VLDB 2006), ACM, 930-941 (2006)* (Jan. 2006) (cited on page 2).

[7] Nikolaos Papailiou, Dimitrios Tsoumakos, Panagiotis Karras, and Nectarios Koziris. “Graph-Aware, Workload-Adaptive SPARQL Query Caching”. In: *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data - SIGMOD ’15*. ACM Press, 2015 (cited on pages 2, 4, 7).

[8] Jaime Salas and Aidan Hogan. “Canonicalisation of Monotone SPARQL Queries”. In: *Lecture Notes in Computer Science*. Springer International Publishing, 2018, pages 600–616 (cited on pages 2, 7).

[9] Denny Vrandečić and Markus Krötzsch. “Wikidata”. In: *Communications of the ACM* 57.10 (Sept. 2014), pages 78–85 (cited on page 3).