Graph Analytics en Bases de Datos de Grafos

Graph Analytics in Graph Databases

Benjamín Farías[1], alumno 4º año.
Juan Romero[1] alumno 4º año.
Adrián Soto[1], alumno de doctorado.
Juan Reutter[1], profesor asociado.

[1]Departamento de Ciencias de la Computación, Escuela de Ingeniería, Pontificia Universidad Católica de Chile.

*Autor para correspondencia: jlreutte@ing.puc.cl

Benjamín Farías[1], 4th year student.
Juan Romero[1], 4th year student.
Adrián Soto[1], graduate student.
Juan Reutter[1], associate professor.

[1]Department of Computer Science, School of Engineering, Pontificia Universidad Católica de Chile.

*Corresponding author: jlreutte@ing.puc.cl

RESUMEN

Actualmente existe un problema de compatibilidad entre las consultas sobre bases de datos de grafos y la aplicación de diversos análisis algorítmicos en dichos grafos. No existen lenguajes de consultas que sean capaces de soportar ambos tipos de tareas al mismo tiempo, por lo que en esta investigación se propone el nuevo lenguaje **SPARQAL** (Hogan, Reutter y Soto, 2020).
Dicho lenguaje está diseñado con el fin de combinar las consultas sobre bases de datos de grafos con el análisis de los componentes de dichos grafos, todo mediante una semántica declarativa y una sintaxis legible y fácil de comprender. Para validar el funcionamiento y la aplicabilidad de **SPARQAL**, se realizaron diversos tests sobre grafos de distintos tamaños, comparando su rendimiento con el de un lenguaje de programación establecido en la actualidad.
Nosotros experimentos se basaron en el *benchmark* Graphalytics (LDBC, 2019), que incluye 6 algoritmos aplicables sobre grafos. Para poder tener una métrica de comparación, los grafos fueron cargados en **PYTHON**, donde la modelación fue propuesta por nosotros.
Los resultados obtenidos permiten evaluar la aplicabilidad de este nuevo lenguaje, además de entregar información sobre los aspectos que se deben mejorar para escalar la implementación y transformarla en una alternativa real a otros lenguajes de consultas sobre grafos.

Palabras clave: SPARQL, graph analytics, web semántica, RDF, metadatos

ABSTRACT

Currently, there is a compatibility problem between the queries on graph databases and the application of various algorithmic analyses on these graphs. There are no query languages capable of supporting both types of tasks at the same time therefore in this study we propose the use of a new language to achieve this purpose: **SPARQAL** (Hogan, Reutter and Soto, 2020). This language is designed to combine the queries on graph databases with the analysis of the components of these graphs through declarative semantics and a readable, easy to understand syntax. To validate the operation and applicability of **SPARQAL**, various tests will be carried out on graphs of different sizes, comparing their performance with that of a currently established programming language.

We perform experiments based on the Graphalytics benchmark (LDBC, 2019), which includes 6 algorithms applicable on graphs. In order to have a comparison metric, the graphs were loaded into **PYTHON**, where modeling was introduced by us.

The results allow evaluating whether this new language can be applied in practice, as well as providing information on the aspects that must be improved to scale the implementation and transform it into a real alternative to other query languages on graphs.

Keywords: SPARQL, graph analytics, semantic web, RDF, metadata

1. INTRODUCCIÓN

En la actualidad, existe un escaso desarrollo en cuanto al rendimiento y aplicabilidad que tienen los motores de bases de datos estructuradas como **GRAFOS**. Las innovaciones respecto a este ámbito se enfocan en el uso de una semántica declarativa y legible para el lenguaje de consultas, tales como Neo4j (Miller, 2013), pero están muy limitadas respecto a la ejecución de análisis sobre los componentes de dichos grafos.

SPARQL es un lenguaje estandarizado para las consultas sobre grafos **RDF**, que están diseñados para representar los datos de la web de forma legible, creando un esquema conocido como la **WEB SEMÁNTICA**. Este esquema busca añadir **METADATOS** a las páginas web, de forma que se pueda analizar su contenido y su relación con otras páginas a gran escala.

1. INTRODUCTION

At present, there is little development in terms of performance and applicability of engines for databases that are structured as **GRAPHS**. Innovations in this area focus on the use of declarative and readable semantics for the query language, such as Neo4j (Miller, 2013), but they are very limited regarding the execution of analyzes on the components of these graphs.

SPARQL is a standardized language for queries on **RDF** graphs, which are designed to represent the web data in a readable way, creating a scheme known as the **SEMANTIC WEB**. This scheme aims to add **METADATA** to web pages, so that their content and relationship with other pages can be analyzed at a large scale.
This study seeks to extend the **SPARQL** language with the aim of improving the performance of queries on

Esta investigación busca extender el lenguaje de **SPARQL**, con el objetivo de mejorar el rendimiento de las consultas sobre bases de datos basadas en grafos. Para esto, se propone un nuevo lenguaje denominado SPARQAL, que combina las consultas con el análisis dentro las bases de datos. La principal tarea a desarrollar será la implementación y ejecución de diversos algoritmos de interés para problemas modelados como grafos, comparando el rendimiento de este nuevo lenguaje con el de un lenguaje de programación ya establecido, que sería normalmente utilizado para realizar análisis sobre los grafos.

2. METODOLOGÍA

La experimentación consistió en comparar los tiempos que tomaba el nuevo lenguaje SPARQAL con el tiempo que le tomaba a un lenguaje preestablecido (Python) para completar la ejecución de una misma tarea sobre un grafo de tamaño determinado. Los algoritmos implementados en Python tenían la propiedad de indexar los vértices del grafo mediante un **HASH INDEX**, guardando en memoria el grafo completo (se asumió que este cabría en memoria).

Esto se realizó haciendo uso de diversas estructuras de datos incluidas en Python, tales como **DICCIONARIOS** y **SETS** basados en hashing, de forma que el acceso a los datos relevantes de cada vértice, así como sus conexiones con otros vértices, fuera lo más eficiente posible.

Para ambos lenguajes, el tiempo de ejecución se consideró desde la lectura de los archivos del grafo hasta que se termina de escribir el resultado en disco.

Los algoritmos a evaluar en ambos lenguajes fueron los siguientes:

1. ***Breadth First Search*** **(BFS):** Busca recorrer todo el grafo a partir de un vértice inicial, visitando cada uno de los otros vértices 1 sola vez, para finalmente entregar la menor cantidad de aristas que requieren ser recorridas para llegar a cada uno de estos vértices (LDBC, 2019). Su utilidad para la web semántica recae en poder analizar qué tan relacionadas están ciertas páginas con otras, reflejado en la cantidad de aristas que separan a 2 páginas en el grafo.
2. ***Weakly Connected Components*** **(WCC):** Separa los nodos del grafo en clases, de modo que dos nodos u y v pertenecen a la misma clase si y sólo si existe un camino para ir de u a v o bien de v a u (LDBC, 2019). Puede ser utilizado para encontrar grupos de páginas web relacionadas bajo algún criterio.
3. ***Local Clustering Coefficient*** **(LCC):** Calcula un coeficiente que indica el grado de CLUSTERING entre los vecinos de un vértice (LDBC, 2019). Este valor puede ser utilizado para estudiar la correlación entre diversas páginas, determinando si las visitas de la gente son producidas por azar o por algún contenido relacionado.
4. ***Page Rank*** **(PR):** Computa para cada vértice del grafo un valor asociado, llamado Page Rank. Este valor

graph-based databases. To this end, a new language called **SPARQAL** is proposed, which combines queries with analysis within databases. The main task to be developed will be the implementation and execution of various algorithms of interest for problems modeled as graphs, comparing the performance of this new language with that of an already established programming language that is normally used to perform analysis on graphs.

2. METHODOLOGY

We compared the time necessary to complete the execution of the same task on a graph of a given size between the new **SPARQAL** language and a pre-established language (Python). The algorithms implemented in Python had the property of indexing the graph's nodes by means of a **HASH INDEX**, storing the complete graph in memory (assuming that it would fit in memory).

This was achieved by using various data structures included in Python, such as **DICTIONARIES** and **SETS** based on hashing, so that access to relevant data for each node, as well as its connections with other nodes, was as efficient as possible.

For both languages, the execution time was determined since the graph files were read until the result was written to disk.

The algorithms evaluated in both languages were the following:

1. **Breadth First Search (BFS):** It goes through the entire graph starting from an initial node, visiting each of the other nodes only once, to finally deliver the least amount of edges that need to be traveled to reach each of these nodes (LDBC, 2019). Its usefulness for the semantic web lies in being able to analyze how related certain pages are to others, indicated by the number of edges that separate 2 pages in the graph.
2. **Weakly Connected Components (WCC):** It separates the nodes of the graph into classes, so that two nodes, u and v, belong to the same class if and only if there is a path to go from u to v or from v to u (LDBC, 2019). It can be used to find groups of related web pages under some criteria.
3. **Local Clustering Coefficient (LCC):** It calculates a coefficient that indicates the degree of CLUSTERING between the neighbors of a node (LDBC, 2019). This value can be used to study the correlation between different pages, determining whether people's visits are produced by chance or by some related content.
4. **Page Rank (PR):** For each node of the graph it computes an associated value, called Page Rank. This value allows to differentiate the nodes according to their relative importance. The algorithm consists of initially assigning the same Page Rank to all nodes, and then iteratively each node will transfer part of its Page Rank to its neighbors (LDBC, 2019). This algorithm is used on the semantic web

permite diferenciar a los vértices según su importancia. El algoritmo consiste en asignar inicialmente a todos los vértices del grafo el mismo Page Rank, y luego de manera iterativa cada vértice transferirá parte de su Page Rank a sus vecinos (LDBC, 2019). La utilidad de este algoritmo en la web semántica es que permite ordenar las páginas según su importancia, ayudando al motor de búsqueda del navegador web a entregar páginas más relevantes al inicio.

5. *Single Source Shortest Paths* (SSSP): Dado un vértice inicial en un grafo, este algoritmo computa la ruta más corta desde dicho vértice hacia todos los demás (LDBC, 2019). Tiene utilidad para estudiar la navegación de las páginas web.

6. *Community Detection using Label Propagation* (CDLP): Al igual que WCC, busca separar en clases los vértices del grafo. Para esto, inicialmente cada vértice tendrá su propia etiqueta y en cada iteración del algoritmo un vértice cambiará su etiqueta por la que tenga mayor presencia en sus vecinos. La idea es que, en regiones densamente conectadas, una etiqueta se propaga más rápido y todos los vértices de esa región terminarán con la misma etiqueta, mientras que en regiones débilmente conectadas le será más difícil a la etiqueta propagarse (LDBC, 2019). Tiene utilidad para el estudio de la relación entre distintas páginas web conectadas entre sí.

Cada uno de estos algoritmos fue ejecutado en grafos de diversos tamaños, abarcando archivos desde 1 KB a 20 GB de información. Se midió el tiempo de ejecución de todas estas configuraciones tanto para Python como para SPARQAL.

3. RESULTADOS Y DISCUSIÓN

Tras la experimentación, se logró demostrar que la extensión SPARQAL desarrollada es capaz de ejecutar consultas correctamente sobre múltiples grafos. A continuación, se muestran las características de los grafos de prueba utilizados, así como el tiempo de ejecución de los algoritmos implementados en SPARQAL:

to sort pages according to their importance, helping the web browser search engine to deliver more relevant pages at the top.

5. Single Source Shortest Paths (SSSP): Given an initial node in a graph, this algorithm computes the shortest path from that node to all the others (LDBC, 2019). It is useful to study the navigation of web pages.

6. Community Detection using Label Propagation (CDLP): Like WCC, it separates the nodes of the graph into classes. Initially, each node will have its own label and in each iteration of the algorithm a node will change its label for the one that has the greatest presence among its neighbors. The idea is that, in densely connected regions, a tag spreads faster therefore all nodes in that region will end up with the same tag, whereas in loosely connected regions it will be more difficult for the tag to spread (LDBC, 2019). This is useful for studying the relationship between different web pages connected to each other.

Each of these algorithms was run on graphs of various sizes, covering files from 1 KB to 20 GB of information. All configurations were clocked for both Python and SPARQAL.

3. RESULTS AND DISCUSSION

After experimentation, it was possible to demonstrate that the developed SPARQAL extension is capable of correctly executing queries on multiple graphs. The features of the test graphs together with the execution time of the algorithms implemented in SPARQAL are shown below:

Id	Nodes	Edges
Q1	93	172
Q2	3057	38738
Q3	480	766
Q4	266	211
Q5	7194	8719
Q6	627	996

Tabla 1. Número de vértices y aristas en los grafos de prueba (Hogan, Reutter y Soto, 2020).

Table 1. Number of nodes and edges in test graphs (Hogan, Reutter and Soto, 2020).

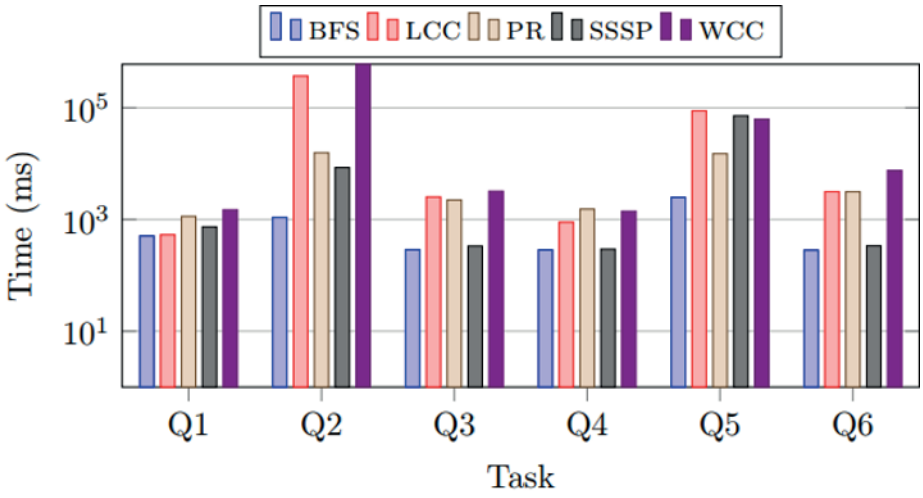

Figura 1. Tiempo de ejecución de algoritmos con SPARQAL en milisegundos (Hogan, Reutter y Soto, 2020).

Figure 1. Execution time of algorithms with SPARQAL in milliseconds (Hogan, Reutter and Soto, 2020).

Como se puede observar, el lenguaje sí fue capaz de entregar un resultado para estos algoritmos, lo que demuestra de manera empírica que SPARQAL es capaz de realizar las tareas de análisis deseadas sobre un grafo. Sin embargo, otro resultado importante fue la eficiencia de esta implementación. Los tests fueron ejecutados en grafos que, en comparación al orden de tamaño de los datos de la web semántica, son relativamente pequeños. A continuación, se muestran los tiempos de ejecución sobre uno de los grafos de prueba de mayor tamaño:

It can be observed that the language was able to deliver a result for these algorithms, showing empirically that SPARQAL is capable of performing the desired analysis tasks on a graph. However, another important result was the efficiency of this implementation. Tests were executed in graphs that, compared to the order of magnitude of the data in the semantic web, are relatively small. Execution times for one of the largest test graphs are shown below:

Id	Nodes	Edges
Q1	93	172
Q2	3057	38738
Q3	480	766
Q4	266	211
Q5	7194	8719
Q6	627	996

Tabla 2. Tiempo de ejecución de cada algoritmo en minutos (Hogan, Reutter y Soto, 2020).

Table 2. Execution time of each algorithm in minutes (Hogan, Reutter and Soto, 2020).

Estos resultados corresponden a la ejecución de estos algoritmos sobre un grafo con 3.774.768 vértices y 16.518.947 aristas. Como se puede apreciar en la tabla, la implementación en Python es mucho más eficiente que la implementación de SPARQAL. Esto implica que, para poder ser utilizado
en la práctica, se deberán considerar nuevas técnicas de procesamiento y optimizaciones dentro de la implementación de este lenguaje.

These results correspond to the execution of the algorithms on a graph with 3,774,768 nodes and 16,518,947 edges. As can be seen from the table, the Python implementation is much more efficient than SPARQAL, implying that, in order to be useful in practice, new processing techniques and optimizations should be considered in the implementation of this language.

4. CONCLUSIONES

A través de la investigación realizada logramos concluir que un lenguaje extendido como SPARQAL sí es capaz de realizar las tareas solicitadas sobre un grafo, mediante un lenguaje declarativo. Además, sentó las bases teóricas que debe tener dicho lenguaje.

Por otra parte, si bien SPARQAL resuelve de manera correcta las tareas pedidas, los resultados muestran que presenta serios problemas para escalar a grafos más grandes, que son justamente los que en un futuro se desearía utilizar con este tipo de base de datos. Con esto se abre una nueva línea de investigación para mejorar el rendimiento de SPARQAL, de forma de que sea capaz de competir frente a lenguajes globalmente establecidos, tales como Python.

Agradecimientos

Agradecemos a los profesores Juan Reutter y Adrián Soto por darnos la posibilidad de participar en esta investigación.

4. CONCLUSIONS

Considering the results of this investigation, we can conclude that an extended language such as SPARQAL is capable of performing the required tasks on a graph through a declarative language. In addition, it laid the theoretical foundations on which this type of language should be based on.

On the other hand, although SPARQAL solves these tasks correctly, the results show that it has serious issues when scaling up to larger graphs, which are precisely the most relevant targets in this type of database that we would like to tackle in the future. This opens a new line of research to improve SPARQAL performance in order to make it competitive with globally established languages like Python.

Acknowledgements

We thank professors Juan Reutter and Adrián Soto for giving us the opportunity to participate in this research.

GLOSARIO

CLUSTERING: Agrupación de elementos de un conjunto de datos bajo cierta característica.
DICCIONARIOS: Estructura de datos que los agrupa de forma clave-valor. El acceso a los datos se realiza entregando la clave y el dato en sí mismo es el valor.
GRAFOS: Estructura de datos teórica, formada por vértices que contienen información de interés y se relacionan entre sí mediante aristas.
HASH INDEX: Técnica para acceder y agrupar un conjunto de datos utilizando funciones de hash que mapean datos de tamaño arbitrario a valores de tamaño definido, de tal forma que mejora el rendimiento de las operaciones sobre los elementos de dicho conjunto.
METADATOS: Conjunto de datos que describen un conjunto más grande de datos.
PYTHON: Lenguaje de programación interpretado, caracterizado por la legibilidad de su código.
RDF: Resource Description Framework, modelo de datos utilizado en la web basado en el uso de metadatos para describir las páginas y relacionarlas entre sí.
SETS: Estructura de datos que simula un conjunto matemático con una colección de datos, proporcionando una manera eficiente de realizar las operaciones de conjuntos (unión, intersección, diferencia, etc.…).
SPARQL: Lenguaje estandarizado para realizar consultas sobre grafos RDF.
WEB SEMÁNTICA: Esquema de la web que busca añadir metadatos semánticos y ontológicos a cada página, modelando el internet como un grafo RDF.

GLOSSARY

CLUSTERING: Grouping of elements of a data set under a certain feature.
DICTIONARY: Data structure that groups items in a key-value relationship. Access to data is achieved by entering the key with the data itself being the value.
GRAPHS: Theoretical data structure, formed by nodes that contain information of interest and are related to each other through edges.
HASH INDEX: Technique to access and group a data set using hash functions, which map data of arbitrary size to values of fixed size to improve the performance of operations on the elements of the set.
METADATA: A data set that describes a larger data set.
PYTHON: Interpreted programming language, characterized by the readability of its code.
RDF: Resource Description Framework, a data model used on the web based on the application of metadata to describe the pages and relate them to each other.
SETS: Data structure that simulates a mathematical set with a collection of data, providing an efficient way to perform set operations (union, intersection, difference, etc.).
SPARQL: Standardized language for querying RDF graphs.
SEMANTIC WEB: Schema of the web that adds semantic and ontological metadata to each page, modeling the internet as an RDF graph.

REFERENCES

[1] Aidan Hogan, Juan Reutter y Adrián Soto. 2020. Recursive SPARQL for Graph Analytics. En arXiv:2004.01816, 1-10.
[2] LDBC. 2019. Graphalytics Benchmark Suite. https://graphalytics.org/.
[3] Justin J. Miller. 2013. Graph Database Applications and Concepts with Neo4j. In Southern Association for Information Systems Conference (SAIS). AIS eLibrary.