# *Benchmark* Proyecto Terminal II

## Objetivo

Conocer en qué situación se encuentra el *triple store* Apache Marmotta respecto a sus símiles y determinar sus fortalezas y debilidades respecto a ellos mediante consultas federadas geoespaciales y la evaluación de tiempo que existe entre solicitud y respuesta.

## Métricas

La métrica usada en este *benchmark* es el tiempo que transcurre entre la petición HTTP de un cliente y la respuesta que el mismo cliente obtiene del servidor.

Debido a la limitada cantidad de memoria RAM que se posee en el equipo de pruebas, la cantidad máxima de tripletas contempladas en las consultas es de máximo 5000 tripletas por lo que la métrica de completitud de respuesta no puede ser cubierta en este *benchmark* sin embargo, para enfocar este *benchmark* a un fin meramente descriptivo, de las 10 consultas usadas en este *benchmark*, se hace uso de la misma consulta 100 veces con el fin de generar datos para realizar un análisis estadístico de los *triple stores* similares a Apache Marmotta.

## Desarrollo

Para realizar este *benchmark* se realizó una búsqueda en Internet con la intención de encontrar datos geoespaciales en formato WKT. Se extrajeron más de 250 candidatos de 3 sitios que recopilan *SPARQL endpoints* y son:

* *LODAtlas* [1]*.*
* *WiKiData* [2]*.*
* *The Linked Open Data Cloud* [3]*.*

De los más de 250 sitios analizados, se usaron 2 *SPARQL* *endpoints* en específico ya que poseen datos con las características mencionadas.

* LinkedGeoData
* Foodie Cloud.

A continuación, se muestra en la tabla 1 los 2 *SPARQL endpoints* con sus características de cada uno.

Tabla 1 Datos sobre los SPARQL endpoints usados en el Benchmark.

|  |  |  |
| --- | --- | --- |
|  | LinkedGeoData | Smart Point of Interest |
|  |  |  |
| Tripletas disponibles |  |  |
|  | http://linkedgeodata.org/sparql | https://www.foodie-cloud.org/sparql |
| Cantidad de tipo de datos distintos disponibles. |  |  |
| Prefijos disponibles |  |  |

Para la extracción de *SPARQL endpoints* de los 3 sitios mencionados se tuvo que hacer distintos procedimientos. En el caso de LODAtlas se desarrolló un *script* que funcionara como un cliente que consumiera datos de la API de LODAtlas para buscar *datasets* mediante el uso de 156 palabras con el prefijo *geo.* Después que se extrajeran los *datasets* disponibles en LODAtlas, se extrajo la *URL* asociada a cada *dataset* en caso de que se tuviera. Finalmente, la cantidad de *SPARQL* *endpoint* extraídos fueron 126. Para el caso de *WikiData* solamente se visitó y copiaron los *SPARQL endpoints* disponibles al día 20 de abril del 2020. Finalmente, para el caso del sitio *The Linked Open Data Cloud* se realizó un *Web Crawler* para extraer los enlaces de los distintos *datasets* disponibles para luego extraer información sobre ellos.

En los 3 casos se guardaron los resultados en archivos tipo Excel que después fueron utilizados para detectar cuáles de ellos realmente contenían información geoespacial en formato *WKT*. Mediante el uso del módulo de consultas federadas geoespaciales implementado en Apache Marmotta y su API, se desarrolló un cliente que leyera cada uno de los *SPARQL endpoints* registrados en los archivos Excel generados previamente para buscar tipos de dato de tipo geoespacial en cada uno de ellos. El fragmento de código 1 muestra la consulta necesaria para realizar la búsqueda.

Captura de pantalla de un celular con texto e imagen

Descripción generada automáticamente

Fragmento de código 1 Consulta federada para buscar datos geoespaciales.

Después de un análisis de los resultados obtenidos, se decidió usar los *SPARQL endpoints* *LinkedGeoData* y *Smart Point of Interest* ya que *LinkedGeoData* proporciona información sobre los objetos que hay en tierra y *Smart Point of Interest* proporciona diversas geometrías que representan los mares en el mundo. Además para complementar las consultas, se utilizaron las geometrías del país de México [1] y de la CDXM [2] extraídos de 2 repositorios de GitHub y fueron cargados en mediante un archivo de tipo *Turtle* en los repositorios.

Con base en el *paper An Evaluation of Approaches to Federated Query Processing over Linked Data* [6]*,* los autores proponen 3 enfoques de federación de datos los cuales son mostrados en la figura 1 de los cuales en este *benchmark* se utilizan principalmente 2: la federación sobre múltiples repositorios simples cuyos datos fueron cargados de *SPARQL endpoints* remotos y la federación sobre múltiples *SPARQL endpoints* remotos. En este *benchmark* y en todo el proyecto se propuso una combinación de B y de C ya que los *SPARQL endpoints* fueron usados para cargar datos en los repositorios simples con diferentes temáticas cada uno de ellos.

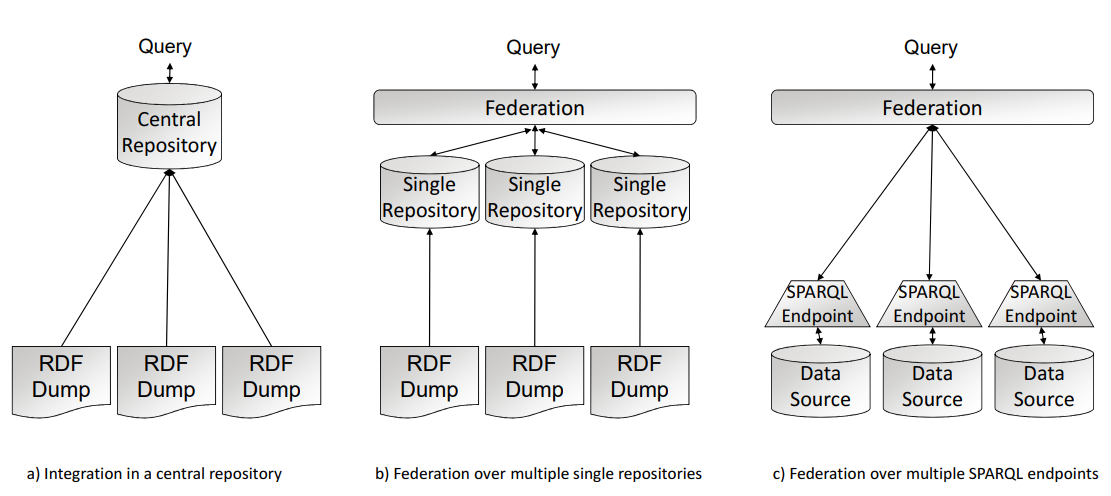


Figura 1 Enfoques de federación de datos en el contexto de la Web del Linked Data.

A continuación, se presentan las consultas y métodos usados para la evaluación de 5 diferentes *triple stores* enfocándose en carga de resultados de consultas federadas en grafos y evaluación de datos geoespaciales basándose en el protocolo *GeoSPARQL.*

Los *triple stores* evaluados comparten la característica de tener la capacidad de procesar consultas geoespaciales y la capacidad de hacer consultas a *SPARQL endpoints* remotos y son los mostrados en la tabla 1.

Tabla 2 Triple stores que usan en el presente benchmark.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Nombre | Consultas Federadas | GeoSPARQL | Status | Libre o pago |
| *Parliament* | Si | Si, en versión 2.7.4 | Activo | Libre |
| *GraphDB* | Si | Si | Activo | Ambos |
| *OpenLink Virtuoso* | Si | Si | Activo | Ambos |
| *Apache Jena* | Si | Si | Activo | Libre |
| *Apache Marmotta* | Si | Si | Activo | Libre |

Para poder hacer las consultas, se utilizaron las geometrías del país de México [1] y de la CDXM [2] extraídos de 2 repositorios de GitHub y fueron cargados en mediante un archivo de tipo *Turtle* en los repositorios.

En el *paper Geographica: A Benchmark for Geospatial RDF Stores* [7] los autores proponen un *benchmark* para la evaluación de *triple stores* usando consultas tipo *GeoSPARQL*. Este *benchmark* está inspirado en el *paper* al tomar ideas de qué tipo de consultas hacer y hacer uso de su plantilla de generación de consultas tanto para la selección como uniones espaciales. En el fragmento de código se muestra la plantilla para selecciones espaciales.



Fragmento de código 2 Plantilla para generar consultas de selección espacial.

Y para la generación de consultas de unión espacial se muestra en el fragmento de código 3 la plantilla para generarlas. Cabe decir que para el caso de este *benchmark* las uniones espaciales se hacen sobre distintos repositorios simples y no uno como lo muestra el fragmento de código 3.

Captura de pantalla de un celular con letras

Descripción generada automáticamente

Fragmento de código 3 PLantilla para generar consultas de unión espacial.

Para el *benchmark* se usó el esquema de federación previamente mencionado para el proceso de cargar datos federados geoespaciales en grafos de manera local y luego hacer consultas tipo *GeoSPARQL.* A continuación, se muestra el banco de 10 consultas diferentes que evalúan lo siguientes aspectos:

* Tiempos de consulta a un *SPARQL endpoint* remoto requiriendo información sobre universidades (Fragmento de código 2).

Captura de pantalla de un celular con letras

Descripción generada automáticamente

Fragmento de código 4 Consulta SPARQL #1.

* Tiempos de consulta a un *SPARQL endpoint* remoto requiriendo información sobre cines(Fragmento de código 3).

Captura de pantalla de un celular con texto

Descripción generada automáticamente

Fragmento de código 5 Consulta SPARQL #2.

* Tiempo para cargar los resultados de consultas federadas geoespaciales en un grafo (Fragmento de código 4).

Captura de pantalla de un celular con texto

Descripción generada automáticamente

Fragmento de código 6 Consulta SPARQL #3.

* Consulta GeoSPARQL *Simple features: Contains* (Fragmento de código 5)*.*

*Captura de pantalla de un celular

Descripción generada automáticamente*

Fragmento de código 7 Consulta SPARQL #4.

* Consulta GeoSPARQL *Egenhoffer: Disjoint* (Fragmento de código 6)*.*

*Captura de pantalla de un celular con texto

Descripción generada automáticamente*

Fragmento de código 8 Consulta SPARQL #5.

* Consulta GeoSPARQL *RCC8: Equals* (Fragmento de código 7).

*Captura de pantalla de un celular con texto

Descripción generada automáticamente*

Fragmento de código 9 Consulta SPARQL #6.

* Consulta no topológica que halla la geometría convexa de la unión entre un punto y los demás del mismo dataset (Fragmento de código 8).

Captura de pantalla de un celular con texto

Descripción generada automáticamente

Fragmento de código 10 Consulta SPARQL #7.

* Consulta que evalúa la igualdad geoespacial mediante una función *RCC8* (Fragmento de código 9).

Captura de pantalla de un celular con letras

Descripción generada automáticamente

Fragmento de código 11 Consulta SPARQL #8.

* Consulta que muestra los hospitales que estén en el borde de un mar (Fragmento de código 10).

Captura de pantalla de un celular con letras

Descripción generada automáticamente

Fragmento de código 12 Consulta SPARQL #9.

* Consulta que une el resultado de 2 consultas federadas (Fragmento de código 11).

Captura de pantalla de un celular con texto

Descripción generada automáticamente

Fragmento de código 13 Consulta SPARQL #10.

Cada consulta es evaluada tiene 5 versiones en la que cada versión se identifica por la cantidad de datos por considerar. Los límites son delimitados mediante la palabra reservada en *SPARQL* llamada *limit* y esos límites son de 250, 500, 1000, 2500 y 5000 datos.

Las consultas están almacenadas en un archivo *XLSX* donde cada columna representa una consulta y cada fila de la columna la versión distinta con base al párrafo anterior. La ejecución de cada consulta se hace mediante un *script* en Python que se encarga de recorrer cada consulta y cada versión de consulta para medir el tiempo transcurrido entre la petición y la respuesta por cada *triple store*. Finalmente, cuando el *script* termina de hacer las consultas a los distintos *triple stores*, se guarda el resultado en otro archivo de tipo *XLSX*.

## Implementación

Tal como se comentó en la sección anterior, se usó un *script* en Python para poder llevar a cabo el *benchmarking* mediante peticiones HTTP donde las cabeceras de las peticiones varían según el *triple store* evaluado. El siguiente fragmento de código muestra el proceso que debe de usarse para la evaluación de los *triple stores*.

Captura de pantalla de un celular con letras

Descripción generada automáticamente

Fragmento de código 14 Código para iterar el excel con consultas.

Donde:

* *tripleStores* es una lista de nombres de los *triple stores* evaluados.
* *URL\_select* es una lista de URL de tipo *SELECT* asociados a la API de los *triple stores* evaluados.
* *URL\_update* es una lista de URL de tipo *UPDATE* asociados a la API de los *triple stores* evaluados.
* *headers* es una lista de cabeceras asociadas a la API de los *triple stores* evaluados.
* *listaConsultas* son las columnas del archivo *XLSX* que contiene las consultas.
* *consultas* es el *Dataframe* que almacena todo el contenido del archivo *XLSX*.
* *variacion* es una lista de números con los límites que deberá de considerar cada consulta.
* *corridas* es un número que indica la cantidad de veces que se usa la misma consulta contra el mismo *triple store*.
* *ejecutaConsulta* es una función que se encarga de ejecutar *corridas* veces la consulta en cuestión

Cabe decir que la consulta “*Q3*” es evaluada de manera distinta ya que se tiene que utilizar un URL distinta al ser una operación de escritura y no de consulta por lo que el argumento de la URL de la función *ejecutaConsulta* es distinto en ese caso. Además, otro aspecto por destacar es que los objetos *tripleStores, URL\_select, URL\_update y headers* son listas del mismo tamaño y cada elemento de cada lista corresponde al elemento de las otras listas en la misma posición.

La función ejecuta consulta recibe como parámetros el índice de la consulta, la consulta, la URL del *triple store* y la cabecera respectiva del método HTTP. Según la cantidad de corridas establecidas, en este caso son 100, se ejecuta la misma consulta contra el mismo *triple store* y se mide el tiempo desde que se hace la petición POST hasta que se reciba los resultados de la consulta. Si la consulta se lleva a cabo, se guarda el tiempo verdadero y si no, se guarda un valor que no es un número. Antes de finalizar, si la consulta realizada es la *“Q3”*, implica que hay que borrar el grafo para que después pueda ser escrito nuevamente. Al final, se devuelve un vector de tiempos de longitud igual a la cantidad de corridas realizadas sobre la misma consulta. El siguiente fragmento de código se muestra la función *ejecutaConsulta*.

Captura de pantalla de un celular

Descripción generada automáticamente

Fragmento de código 15 Código que efectua las consultas contra el triple store.

## Resultados

Se cuenta con un conjunto de datos de 3000 registros por cada *triple store* por lo que se hace un análisis descriptivo por límite de tripletas y otro por consulta*.*

Usando el archivo *XLSX* que contiene todos los registros de tiempo de los *triple stores*, se desarrolló un *script* en R que se encarga de extraer la media y desviación estándar de manera genera y por límite de tiempo o consulta. El siguiente fragmento de código muestra el proceso para el cálculo de la media y de la desviación estándar por límite de tripletas. Este código aplica igual para la realizar los mismos cálculos para consultas, pero modificando la sección de *filter*.

Una captura de pantalla de un celular con letras

Descripción generada automáticamente

Fragmento de código 16 Código para calcular la media y la desviación estándar por límte de tripletas.

En la tabla 2 se muestra por cada triple store, el promedio de tiempo en general y el límite de tripletas, así como la desviación estándar y la desviación estándar por límite de tripletas.

Tabla 3 Estadística descriptiva por límite de tripletas.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 𝑇𝑟𝑖𝑝𝑙𝑒 𝑆𝑡𝑜𝑟𝑒 |  |  |  |  |  |
| Apache Marmotta | 0.3813625 | 0.4005342 | 250 | 0.1897614 | 0.3378772 |
| 500 | 0.1976510 | 0.1727636 |
| 1000 | 0.2765171 | 0.2238653 |
| 2500 | 0.4591795 | 0.3070222 |
| 5000 | 0.7837034 | 0.5148006 |
| Parliament |  |  | 250 |  |  |
| 500 |  |  |
| 1000 |  |  |
| 2500 |  |  |
| 5000 |  |  |
| GraphDB |  |  | 250 |  |  |
| 500 |  |  |
| 1000 |  |  |
| 2500 |  |  |
| 5000 |  |  |
| OpenLink Virtuoso |  |  | 250 |  |  |
| 500 |  |  |
| 1000 |  |  |
| 2500 |  |  |
| 5000 |  |  |
| Apache Jena |  |  | 250 |  |  |
| 500 |  |  |
| 1000 |  |  |
| 2500 |  |  |
| 5000 |  |  |

En la tabla 3 se muestra por cada triple store, el promedio de tiempo en general y el límite de tripletas, así como la desviación estándar y la desviación estándar por límite de tripletas.

Tabla 3 Estadística descriptiva por límite de tripletas.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |
| Apache Marmotta | 0.3813625 | 0.4005342 | Q1 | 0.494193496 | 0.2628391184 |
| Q2 | 0.510401007 | 0.4300727411 |
| Q3 | 0.814365381 | 0.4634338891 |
| Q4 | 0.004491567 | 0.0003036705 |
| Q5 | 0.220866272 | 0.2027187209 |
| Q6 | 0.243857244 | 0.2233387063 |
| Parliament |  |  | Q1 |  |  |
| Q2 |  |  |
| Q3 |  |  |
| Q4 |  |  |
| Q5 |  |  |
| Q6 |  |  |
| GraphDB |  |  | Q1 |  |  |
| Q2 |  |  |
| Q3 |  |  |
| Q4 |  |  |
| Q5 |  |  |
| Q6 |  |  |
| OpenLink Virtuoso |  |  | Q1 |  |  |
| Q2 |  |  |
| Q3 |  |  |
| Q4 |  |  |
| Q5 |  |  |
| Q6 |  |  |
| Apache Jena |  |  | Q1 |  |  |
| Q2 |  |  |
| Q3 |  |  |
| Q4 |  |  |
| Q5 |  |  |
| Q6 |  |  |

### Gráficas

En la figura 1 se muestra una gráfica de pastel indicando cuanto tarda cada consulta en ser procesada en los *triple store* de manera general.

Captura de pantalla de un celular con letras

Descripción generada automáticamente

Figura 2 Distribución de tiempos por consulta.

En la figura 2 se muestra un histograma que representa la distribución de tiempos en todo el análisis.

Imagen que contiene captura de pantalla

Descripción generada automáticamente

Figura 3 Histograma de tiempos.

Para hacer filtrado de datos en las gráficas por *triple store*, consulta y por límite de tripletas, se sugiere visitar el siguiente [enlace](https://public.tableau.com/profile/p.ez.oswaldo#!/vizhome/ResultadosBenchmarking/Dashboard1) que lo redirige a un *dashboard* donde podrá interactuar con los resultados.

## Conclusiones