HDT·EndPoints: una Arquitectura Eficiente para la Web de Datos *

Javier D. Fernández[1], Miguel A. Martínez-Prieto[1,2], Mario Arias Gallego[1], and Claudio Gutierrez[2]

[1] Departamento de Informática, Universidad de Valladolid (Spain)
{jfergar,migumar2}@infor.uva.es, mario.arias@gmail.com
[2] Departmento de Ciencias de la Computación, Universidad de Chile (Chile)
cgutierr@dcc.uchile.cl

Resumen El actual "diluvio de datos" está inundando la Web con grandes volúmenes de información representados en RDF, dando lugar a la *Web de Datos*. Los puntos de acceso a esta red semántica son los *SPARQL endpoints*, servicios que interpretan el lenguaje de consulta SPARQL. El rendimiento de esta infraestructura se ve claramente afectado por los formatos verbosos utilizados para la representación de RDF, cuyo procesamiento e intercambio introduce importantes retrasos en la comunicación al operar con grandes conjuntos de datos. En este trabajo introducimos los principios básicos de la arquitectura **HDT·EndPoints**. Esta propuesta se fundamenta en el uso de una representación compacta de RDF, denominada HDT, sobre la que se diseñan servicios para el descubrimiento, consulta e intercambio de información en la Web de Datos.

1. Introducción

La Web, como ente alimentado social y tecnológicamente, se encuentra en constante evolución. Actualmente, la versión más utópica de la Web Semántica está viéndose superada por una versión menos estricta conocida como la "Web de Datos". Esta surge al considerar una visión "centrada en datos" frente al tradicional punto de vista "basado en documentos". Su pilar fundamental es RDF (*Resource Description Framework*[3]), un modelo de datos semi-estructurado que permite expresar sentencias como triples $(sujeto, predicado, objeto)$ que pueden verse como grafos etiquetados $s \xrightarrow{p} o$. El principal valedor de dicho modelo, y de la Web de Datos, es la filosofía *Linked Open Data*[4], que aboga por disponer de datos RDF abiertos y enlazados con otras fuentes de datos. SPARQL[5] es el lenguaje de consulta estándar para RDF. Los conjuntos de datos se exponen para su consulta, en la Web de Datos, a través de *SPARQL endpoints* que, típicamente, entregan sus resultados en un formato procesable automáticamente.

* Financiado por MICINN (TIN2009-14009-C02-02), Instituto de Dinámica Celular y Biotecnología (ICDB) (ICM P05-001-F), y Fondecyt (1090565 y 1110287). El primer autor posee una beca de la Junta de Castilla y León y el Fondo Social Europeo.

[3] http://www.w3.org/TR/rdf-primer/
[4] http://linkeddata.org/
[5] http://www.w3.org/TR/rdf-sparql-query/

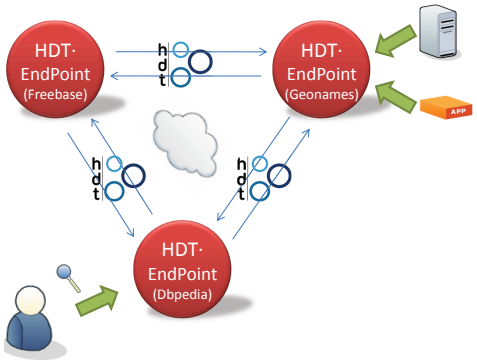

Figura 1. Arquitectura de comunicación HDT·EndPoint.

Actualmente se publican en RDF grandes cantidades de datos procedentes de diferentes áreas, como datos biológicos (Uniprot), estadísticos (2001 US Census), o geográficos (GeoNames). Sin embargo, el diseño original del modelo RDF se centró en la descripción de pequeños documentos o recursos. Esto se traduce en formatos de serialización, como RDF/XML[6] o N3[7], excesivamente verbosos. Este hecho ocasiona graves problemas de escalabilidad a la hora de publicar e intercambiar las colecciones disponibles en la Web de Datos. HDT[8] [2] es un formato de serialización RDF que acomete la problemática anterior.

Este trabajo contempla las propiedades de HDT como base para el desarrollo de una nueva arquitectura (denominada **HDT·EndPoints**) que afronta el consumo de grandes colecciones RDF. Esta propuesta modela la Web de Datos como una red (ver Figura 1) cuyos nodos almacenan, sirven y se comunican utilizando HDT, facilitando que los usuarios (no necesariamente humanos) interactúen de forma eficiente sobre las colecciones potencialmente distribuidas.

La organización del trabajo plantea un resumen conciso del estado del arte (Sección 2), una descripción de la arquitectura HDT·EndPoints y de sus servicios (Sección 3) y, finalmente, una breve discusión (Sección 4).

2. Estado del Arte

El modelo RDF no está adscrito a ningún formato de serialización particular. RDF/XML, a pesar de su verbosidad, es una buena alternativa para el intercambio de datos a pequeña escala. Otras notaciones como Turtle[9] o N3, permiten abreviar algunas construcciones, aunque ninguna de ellas considera, entre sus objetivos principales, el volumen de información a representar.

[6] `http://www.w3.org/TR/REC-rdf-syntax/`
[7] `http://www.w3.org/DesignIssues/Notation3/`
[8] El formato `HDT` ha sido recientemente publicado como W3C Member Submission: `http://www.w3.org/Submission/2011/03/`
[9] `http://www.w3.org/TeamSubmission/turtle/`

De igual modo, si bien existen múltiples sistemas de indexación RDF, la resolución eficiente y escalable de consultas SPARQL es un problema abierto. Algunas propuestas emplean una Base de Datos relacional para almacenar RDF, resolviendo SPARQL a través de SQL [6]. Otras, emplean índices específicos para RDF para servir SPARQL de manera nativa [4].

El concepto de SPARQL endpoint, como interfaz pública de consulta SPARQL, es clave en la actual Web de Datos. El rendimiento de los endpoints está claramente afectado por los factores mencionados anteriormente: (i) El **tiempo de consulta**, influenciado por el balance de accesos a disco y memoria (y por consiguiente del tamaño del índice), y (ii) el **tiempo de transmisión** de los resultados de una consulta, que depende del formato de serialización.

Un SPARQL endpoint permite, habitualmente, el acceso a un único conjunto de datos, necesitando mecanismos adicionales para expandir la consulta en la Web de Datos. Las dos aproximaciones principales se basan en 1) federación de consultas y 2) centralización de datos. En la primera se establece una capa superior de abstracción con acceso a varios SPARQL endpoint, y se ofrecen servicios para la descomposición de consultas en subconsultas SPARQL [5]. Por contra, la centralización de datos se basa en replicar varios conjuntos de datos (fuentes) en un único punto de acceso[10]. Otras propuestas, que vienen a suplir la carencia de descubrimiento dinámico de las anteriores, incrementan dinámicamente los nodos de consulta explorando las URIs obtenidas en cada subconsulta [3].

3. Arquitectura HDT·EndPoints

3.1. HDT

HDT [2] es un formato binario para la representación compacta de RDF que modela un conjunto de datos mediante tres componentes: (i) la cabecera (*Header*) contiene metadatos sobre la colección y su organización para facilitar su descubrimiento y consumo, (ii) el diccionario (*Dictionary*) organiza el vocabulario utilizado en la colección RDF asignando un identificador numérico único a cada elemento, y (iii) los triples (*Triples*) representan la topología del grafo.

HDT permite representar un conjunto de datos RDF en un 15 % del espacio utilizado por los formatos tradicionales de RDF [2], llegando a cotas cercanas al 4 % si se combina con compresores universales. Una implementación del componente Triples [1], basada en estructuras de datos compactas, indexa la topología del grafo de forma comprimida y permite, a su vez, la resolución eficiente de consultas sobre el mismo.

3.2. Descripción de la Arquitectura

La revisión del estado del arte descubre la necesidad de disponer de una arquitectura que facilite que los procesos de comunicación y consulta se desarrollen

[10] `http://sindice.com/`

```
1 CONSTRUCT { <http://dbpedia.org/resource/Berlin> ?predicado ?objeto .
2             ?objeto ?predicadoObjRelacionado ?objetoRelacionado .
3             ?sujRelacionado ?predSujRelacionado <http://dbpedia.org/resource/Berlin> .
4 } WHERE {
5   { <http://dbpedia.org/resource/Berlin> ?predicado ?objeto .
6     OPTIONAL{ ?objeto ?predicadoObjRelacionado ?objetoRelacionado .}
7   } UNION{ ?sujRelacionado ?predSujRelacionado <http://dbpedia.org/resource/Berlin> .}
8 }
```

Figura 2. Ejemplo de consulta SPARQL sobre la arquitectura.

de forma óptima. Esto se traduce, principalmente, en disponer de (i) mecanismos eficientes y escalables para el almacenamiento e indexación de colecciones de datos de tamaños cada vez mayores; (ii) formatos compactos para el intercambio de datos y (iii) un protocolo adecuado para la comunicación y el descubrimientos de nuevos recursos. Nuestra arquitectura integra todas estas propiedades a través del concepto de HDT·EndPoint:

Definición 1 (HDT EndPoint) *Un nodo HDT EndPoint es un elemento (i) conforme a la especificación del protocolo SPARQL para RDF (SPROT)*[11]*, (ii) que extiende su funcionalidad para descubrir y comunicarse con otros HDT EndPoints, y (iii) utiliza HDT como formato de intercambio de datos.*

La Figura 1 muestra un ejemplo de arquitectura de comunicación. En el mismo, se modela una red mediante tres HDT·EndPoints que almacenan tres conjuntos de datos[12] presentes en el mundo Linked Data; DBpedia es el equivalente de Wikipedia representado en RDF, Freebase aglutina diversas fuentes de datos (*mashup*) y Geonames describe datos geográficos. Supongamos un cliente que desea toda la información sobre "Berlin". La consulta SPARQL sería semejante a la referida en la Figura 2; se construye un grafo RDF con todos los triples que tienen a Berlin como sujeto (línea 5) en DBpedia; si existen, se añaden también aquellos triples relacionados con los anteriores (línea 6). Junto con ellos, se incorporan los triples que tengan a Berlin como objeto (línea 7).

La resolución de esta consulta implica, no sólo una resolución local en DBpedia, sino una comunicación distribuida con otros nodos que pueda aportar información a la consulta. En este ejemplo, la información de Berlin en DBpedia contiene enlaces tanto hacia Freebase como Geonames (y viceversa), con cientos de miles de triples como resultado total.

La Figura 3 muestra la estructura y el flujo de comunicación en la arquitectura que permite resolver esta y otras consultas distribuidas y/o que supongan un gran volumen de datos. En el ejemplo anterior, tras recibir la consulta del cliente, el HDT·EndPoint intentaría resolverla localmente, ya que extiende la funcionalidad de un SPARQL endpoint tradicional, sobre la base de un *RDF Store*. Para aquellas partes que no es capaz de resolver completamente, consulta su Tabla de Servicios Remotos. Como se verá a continuación, esta tabla permite conocer qué HDT·EndPoints pueden albergar cierta información. Tras consultar la tabla, el HDT·EndPoint de DBpedia descubrirá los de Freebase y Geonames,

[11] `http://www.w3.org/TR/rdf-sparql-protocol/`

[12] `http://dbpedia.org, http://www.freebase.com/, http://www.geonames.org/`

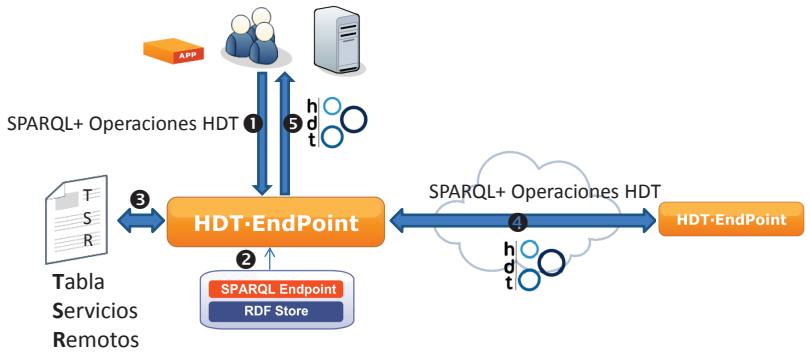

Figura 3. Estructura y Flujo de Comunicación de HDT·EndPoints.

realizando la subconsulta apropiada. Tanto las respuestas de las mismas como el resultado final servido al cliente se realizan en HDT, para que sean eficientes tanto en espacio/ancho de banda como en tiempo de respuesta.

3.3. Servicios

Como se aprecia en la Figura 3, al margen de SPARQL, los HDT·EndPoints permiten servir otras operaciones, denominadas *Operaciones HDT*, que están relacionadas con los servicios que ofrece la arquitectura.

Descubrimiento. Para proporcionar un servicio de descubrimiento, se dota a cada HDT·EndPoint de una Tabla de Servicios Remotos (TSR) como la mostrada en la Tabla 1. Ésta puede verse como una tabla de enrutamiento, en la cual se posee una entrada por cada conjunto de datos almacenado en un HDT·EndPoint. Así, para cada conjunto de datos (*Dataset*), se mantienen sus espacios de nombres (*Namespaces*) y la URI del HDT·EndPoint que lo aloja. Adicionalmente se puede incluir el componente cabecera de dicho conjunto de datos y una marca de tiempo (*Timestamp*) para conocer la caducidad de la información. Suponemos que la tabla TSR se inicializa en cada HDT·EndPoint con sus conjuntos de datos locales y con aquellos HDT·EndPoints que albergan información referenciada en los datos locales (*out-links*). Para permitir una política de actualización dinámica, los HDT·EndPoints deben servir dos operaciones adicionales:

- *1. ObtenerCabecera(dataset)*: recuperar la cabecera (Header) de un dataset albergado en el HDT·EndPoint, o incluido en su tabla TSR.
- *2. ObtenerDescripción()*: recuperar la descripción de los conjuntos de datos que gestiona, esto es, su identificador, *namespaces* y política de actualización.

Consulta. Un HDT·EndPoint, además de servir SPARQL, incrementa la capacidad de consulta gracias a la filosofía HDT subyacente:

- *3. ObtenerDiccionario(dataset [,formato,subconjunto])*: permite recuperar el diccionario de un conjunto de datos albergado en el HDT·EndPoint. Opcionalmente, se puede especificar un formato de diccionario o un subconjunto del diccionario a recuperar (predicados, sujetos-objetos compartidos, etc.)

Dataset	Namespaces	HDT·EndPoint	Header	Timestamp
dbpedia	http://dbpedia.org	localhost	-	-
geonames	http://www.geonames.org; http://sws.geonames.org	http://· · · /GeoPoint	GeoHeader.H	2011-11-09 T 11:20
freebase	http://rdf.freebase.org	http://· · · /FreePoint	FreeB.H	-

Tabla 1. Ejemplo de Tabla de Servicios Remotos (DBPedia).

- *4. ObtenerTriples(dataset [,formato])*: se recupera la estructura de triples de un conjunto de datos. Permite concretar el formato de la devolución.

Intercambio. El servicio de intercambio se delega en la capacidad eficiente de compactación que proporciona HDT. Las consultas SPARQL se extienden para incluir la opción de especificar el *formato* HDT que se pretende recibir como resultado, ya que existen múltiples posibilidades de codificación.

4. Discusión

La arquitectura HDT·EndPoints permite paliar uno de los mayores problemas actuales de la Web de Datos: la *escalabilidad*. Esta arquitectura hace uso del formato HDT para reducir el ancho de banda necesario para intercambiar información RDF, permitiendo manejar volúmenes de información más amplios con menores latencias, lo que favorece una mayor distribución de la información en la red. Asimismo, el hecho de que los resultados se intercambien en un formato compacto y buscable permite un consumo directo de los resultados obtenidos. Estas características abren nuevas oportunidades de aplicaciones en la Web de Datos, como la aparición de clientes ligeros (dispositivos móviles) que puedan consultar y recibir de manera compacta grandes volúmenes de información distribuida, y sirvan sólo aquella parte de interés en un momento dado. Actualmente estamos trabajando en una implementación abierta para la comunidad, que permita explotar todo este potencial.

Referencias

1. S. Álvarez García, N. Brisaboa, J.D. Fernández, and M.A. Martínez-Prieto. Compressed k2-Triples for Full-In-Memory RDF Engines. In *AMCIS 2011. Disponible en* `http://arxiv.org/abs/1105.4004`, 2011.
2. J.D. Fernández, M.A. Martínez-Prieto, and C. Gutierrez. Compact Representation of Large RDF Data Sets for Publishing and Exchange. In *ISWC'2010*, 2010.
3. O. Hartig, C. Bizer, and J.C. Freytag. Executing SPARQL Queries over the Web of Linked Data. In *ISWC 2009*, volume 5823, pages 293–309, 2009.
4. T. Neumann and G. Weikum. The RDF-3X Engine for Scalable Management of RDF data. *The VLDB Journal*, 19(1):91–113, 2010.
5. B. Quilitz and U. Leser. Querying Distributed RDF Data Sources with SPARQL. In *ESWC'2008*, pages 524–538, 2008.
6. L. Sidirourgos, R. Goncalves, M. Kersten, N. Nes, and S. Manegold. Column-store Support for RDF Data Management: not All Swans are White. *Proc. of the VLDB Endowment*, 1(2):1553–1563, 2008.