Revista Venezolana de Computación
ISSN: 2244-7040
http://www.svc.net.ve/revecom
Vol. 2, No. 1, pp. 95-106, Diciembre 2015
Selección de los Mejores Artículos de CoNCISa 2015

Aplicación para Dispositivos Móviles Android con Capacidad de Procesamiento de Consultas Realizadas sobre Documentos RDF Almacenados en la Nube

Jesús Soto, Amadís Martínez
jesusotox.35@outlook.com, aamartin@uc.edu.ve

Departamento de Computación, FaCyT, Universidad de Carabobo, Valencia, Venezuela

Resumen: Los dispositivos móviles se han convertido en parte de nuestra cotidianidad, permitiendo la realización de tareas habituales de manera sencilla y práctica. La plataforma Android ha crecido conjuntamente con los dispositivos móviles al estar presente en gran parte de los mismos. Por otro lado, la *Web* Semántica brinda la posibilidad de crear *software* capaz de procesar su contenido y realizar deducciones lógicas. Para ello, el W3C propone *Resource Description Framework* (RDF), un lenguaje para expresar metadatos, y SPARQL, un lenguaje de consultas sobre RDF. Los dispositivos móviles cuentan con capacidades limitadas de procesamiento, memoria y energía, dificultando el almacenamiento de un repositorio RDF local. Este trabajo propone el desarrollo de un mecanismo para Android, que permita consultar repositorios RDF en la nube, para superar la limitación del almacenamiento local del repositorio. Para esto, se estudiaron los componentes del *framework* de Android para la implementación del mecanismo que permitiera un uso compartido del mismo. Igualmente, se analizó el impacto del mecanismo sobre la tarea de responder consultas. Los componentes diseñados se implementaron en un prototipo completamente funcional para consultas SPARQL a repositorios en la nube desde un dispositivo con Android, al igual que una aplicación para la demostración y evaluación del rendimiento del mecanismo. Finalmente, se realizaron pruebas experimentales para evaluar la calidad del modelo planteado, utilizando datos sintetizados y reales. Los resultados obtenidos muestran que, en promedio, este enfoque es viable para la creación de aplicaciones Android basadas en la *Web* Semántica a través de consultas a repositorios RDF.

Palabras Clave: Android; *Web* Semántica; RDF; SPARQL; Aplicación; Servicio.

Abstract: Mobile devices have become part of our daily life, making fast and easy to perform common tasks. The Android platform has grown along with the mobile devices, being present in wide variety of them. Furthermore, the Semantic Web provides the ability to create software capable of processing its own content and make logical deductions. For this, the W3C proposed the Resource Description Framework (RDF), a language for expressing meta-data, and SPARQL, a query language for RDF. Mobile devices possess limited processing, memory and energy capabilities, making difficult to store a RDF repository locally. This paper proposes the development of a mechanism for Android, which allows to access RDF repositories in the cloud, to overcome the limitation of the local storage of the repository. For this, the Android framework components were studied in order to implement a mechanism that allows a shared usage of it. Similarly, the impact of the mechanism was analyzed on the task of query answering. The designed components were implemented in a fully functional prototype to make SPARQL queries to cloud-stored repositories from a Android device, as well as an application for demonstration and performance evaluation of the mechanism. Finally, several tests were performed to assess the quality of proposed model using synthesized and real data. The results show that, on average, this approach is feasible for building Semantic Web based Android applications through queries to RDF repositories.

Keywords: Android; Semantic Web; RDF; SPARQL; Application; Service.

I. Introducción

Es muy notoria la relevancia que han adquirido los dispositivos móviles; en estos últimos años se han convertido aceleradamente en parte de la vida de cada uno de nosotros al ofrecer la posibilidad de ejecutar tareas, que solían ser algo complejas, ahora desde la palma de la mano. En particular, la plataforma Android [1] es una de las más populares y reconocidas debido a que ha logrado abarcar una gran cantidad de dispositivos en el mercado. Además, por ser una plataforma de código abierto, ha podido ser adaptada y optimizada para su uso en dispositivos de distintos fabricantes.

Android es un sistema operativo basado en el *kernel* de Linux,

actualmente es desarrollado y mantenido por la compañía Google Inc. Android consta de una interfaz de usuario basada en la manipulación directa, por lo que está diseñado primordialmente para pantallas táctiles de teléfonos inteligentes y *tablets*.

Por otra parte, la *World Wide Web* (WWW) es un sistema que permite el acceso a una enorme colección de documentos enlazados, los cuales se encuentran dispersos en millones de computadores a lo largo del mundo conectados a través de *Internet*. El funcionamiento de la *Web* ha sido posible por medio de un conjunto bien establecido de estándares, los cuales garantizan varios niveles de interoperabilidad: la existencia de un protocolo de comunicaciones para el intercambio de datos (*Transmission Control Protocol / Internet Protocol* – TCP/IP) garantiza el nivel de interoperabilidad técnica; la definición de un protocolo para la recuperación de documentos (*HyperText Transfer Protocol* – HTTP) y la existencia de una estructura para la presentación de contenidos (*HyperText Markup Language* – HTML) garantizan el nivel de interoperabilidad sintáctica [2].

La presentación de contenidos en la *Web* fue concebida originalmente para su procesamiento directo por usuarios humanos. Esto significa que, aunque la información que reside en los documentos de la *Web* es legible por las máquinas, no es entendible por ellas. Los usuarios humanos pueden utilizar las máquinas para realizar búsqueda de información en los documentos de la *Web*. Sin embargo, el resultado de esta búsqueda debe ser interpretado por el usuario humano: las máquinas pueden presentar la información al usuario, pero no pueden inferir cuál es la información relevante o útil con respecto a la búsqueda realizada [2].

La *Web* Semántica (*Semantic Web*) es una propuesta basada en la idea de construir una infraestructura de semántica legible por las máquinas para la información en la *Web*, con la intención de superar las limitaciones explicadas anteriormente. Para que esto sea posible, la *Web* Semántica requiere que se establezca el nivel de interoperabilidad semántica, el cual, a su vez, requiere la definición de un conjunto de nuevos estándares, no sólo para la forma sintáctica de los documentos, sino también para la descripción de su contenido semántico. En este sentido, el *WWW Consortium* (W3C) ha liderizado la elaboración de propuestas que sirven como base para esta infraestructura: *Resource Description Framework* (RDF) junto con RDF *Schema* (RDFS) proporcionan los mecanismos fundamentales para la descripción de interrelaciones semánticas entre recursos en la *Web* [3].

Para la plataforma Android, existen dos sistemas de administración de datos RDF que permiten crear aplicaciones basadas en la *Web* Semántica: AndroJena [4] y OpenRDF *Sesame* [5]. Estos son sistemas de propósito general para la creación, almacenamiento y manipulación de documentos RDF. Ambos soportan consultas a través del lenguaje SPARQL, el cual permite hacer búsquedas sobre los recursos de la *Web* Semántica utilizando distintas fuentes de datos.

Sin embargo, hablar de dispositivos móviles es sinónimo de recursos limitados, poder de procesamiento restringido, escasa memoria RAM, poca capacidad de almacenamiento secundario y limitada cantidad de energía para operar. Por lo tanto, lo que AndroJena y *Sesame* tienen en común, es que operan siempre y cuando el repositorio de datos esté almacenado en el mismo dispositivo, es decir, localmente. Dadas las pocas capacidades de los dispositivos móviles y los grandes tamaños que pueden llegar a tener los repositorios, es difícil pensar que se pueda tener alguno almacenado dentro de un dispositivo móvil, consumiendo toda o gran parte de la capacidad de almacenamiento, esto reduciría drásticamente la eficiencia del dispositivo.

En virtud de lo antes expuesto, el objetivo principal de esta investigación consiste en el desarrollo y evaluación de una aplicación para dispositivos móviles con Android que utilice un servicio Android compartido capaz de realizar consultas, utilizando el lenguaje SPARQL, a un repositorio RDF que no esté almacenado localmente en el dispositivo. En otras palabras, el repositorio estará almacenado en la nube, de modo que se obtengan sólo los resultados de interés sin necesidad de desperdiciar espacio en un repositorio completo, preservando mejor el rendimiento y la capacidad de almacenamiento del dispositivo móvil.

Los aportes de este trabajo se resumen a continuación:

- Un servicio Android compartido que permite a cualquier aplicación del dispositivo hacer uso de él para consultar, a través del lenguaje SPARQL, a repositorios RDF almacenados en la nube y utilizando HTTP como protocolo de comunicación, solventando así, la limitación del almacenamiento local del repositorio en el dispositivo móvil.
- Una aplicación Android que hace uso del servicio Android desarrollado para consultar un *endpoint* SPARQL, obtener los resultados mediante su descarga y posteriormente procesarlos.
- Una extensa fase de evaluación compuesta por un estudio experimental sobre la tarea de evaluar el rendimiento del mecanismo desarrollado al realizar consultas en SPARQL desde un dispositivo móvil a un *endpoint* SPARQL.

Este documento fue estructurado en cinco secciones, incluyendo la introducción. En la Sección II se presenta el marco teórico, los antecedentes y los trabajos relacionados con esta investigación. La Sección III describe el diseño del mecanismo planteado para obtener solución al problema de consultar repositorios RDF desde dispositivos móviles con Android. La Sección IV contiene los resultados experimentales obtenidos por el esquema desarrollado, presentando el estudio experimental de su rendimiento; además, se compara el rendimiento entre los diferentes formatos de respuesta a las consultas. Finalmente, en la Sección V se presentan las conclusiones de la investigación, las recomendaciones finales y los trabajos futuros.

II. Marco Teórico

A. Trabajos Relacionados

Resource Description Framework (RDF) es una propuesta del W3C para expresar metadatos acerca de información

de recursos que se encuentran en la *Web* direccionados por URIs. El objetivo de RDF consiste en describir recursos de información de una manera comprensible por máquinas, y que estas descripciones puedan ser procesadas por aplicaciones. En el modelo de datos de RDF, la estructura básica es la terna RDF (RDF *triple*), la cual es de la forma (s, p, o), donde s (sujeto) es el recurso que se está describiendo, p (predicado) es una propiedad del recurso s y o (objeto) es el valor de la propiedad p. Un conjunto de ternas RDF se denomina grafo RDF (RDF *graph*) y un conjunto de uno o más grafos RDF se denomina fuente de datos RDF (RDF *dataset*).

Un sistema de administración de datos RDF es un *software* de propósito general que permite la creación, almacenamiento persistente y manipulación de fuentes de datos RDF. Además, debe ofrecer soporte para dos tareas fundamentales: (1) responder consultas realizadas por usuarios y agentes de *software* sobre fuentes de datos RDF y (2) razonamiento semántico sobre fuentes de datos RDF, para descubrir interrelaciones entre los recursos descritos [2].

A medida que las fuentes de datos RDF han crecido en magnitud, así como la repercusión que han tenido los dispositivos móviles en la sociedad, se han incrementado los esfuerzos para buscar soluciones viables que permitan mejorar el desempeño y escalabilidad con respecto al almacenamiento y consulta de estas fuentes de datos RDF desde estos dispositivos. En el contexto de Android y la *Web* Semántica se han desarrollado dos sistemas de administración de datos RDF, con el objetivo de ofrecer acceso eficiente a fuentes de datos RDF:

1) AndroJena: AndroJena [4] es un sistema de administración de RDF de código abierto (*open source*), implementado en JAVA y es el *porting* de Jena para la plataforma Android de Google. Al ser un *porting* de Jena, AndroJena ofrece las mismas características adaptadas a la plataforma Android, como una interfaz de programación (API) para la manipulación de grafos RDF como conjuntos de ternas RDF. AndroJena fue extendido con ARQoid [6] y TDBoid para así, soportar de la misma manera consultas SPARQL [7] y almacenamiento persistente en dispositivos móviles con Android.

2) OpenRDF Sesame: Sesame [8] es un sistema de administración de RDF de código abierto (*open source*), implementado en JAVA y disponible para descarga [5]. Sesame cuenta con la cualidad de que puede ser ejecutado en dispositivos móviles con Android. Para la manipulación y consulta de grafos RDF, Sesame ofrece una interfaz de programación (API) llamada SAIL. A través de esta API, permite almacenar los grafos RDF en memoria principal, en memoria secundaria o en en un sistema manejador de bases de datos relacionales, utilizando una técnica basada en el modelo relacional tradicional denominada tabla de ternas (*triple table*).

Una vez almacenadas las ternas de un grafo RDF, Sesame ofrece tres operaciones básicas sobre la base de datos asociada con el grafo RDF original: (1) agregar (*add*), que inserta una terna RDF en la base de datos, (2) borrar (*remove*), que elimina una terna RDF del grafo y (3) obtener (*get*), que retorna todas las ternas RDF que coinciden con un patrón de acceso de la forma (s, p, o), donde s, p y o pueden ser constantes o variables. La versión más reciente ofrece soporte para consultas SPARQL.

Pese a que ambos sistemas de administración RDF (AndroJena [4] y *Sesame* [5]) están adaptados para su ejecución en la plataforma Android, permitiendo así almacenar un repositorio RDF en el dispositivo móvil y realizar consultas al mismo, tienen la desventaja que ambos asumen y requieren que el almacenamiento del repositorio sea local, y debido al gran tamaño que estos pueden ocupar, podrían consumir la totalidad del almacenamiento del dispositivo o incluso no ser posible su almacenamiento si el dispositivo es muy restringido.

B. Bases Teóricas

1) SPARQL: Es un lenguaje de consultas sobre RDF estandarizado por el W3C que se utiliza para expresar consultas que permiten interrogar diversas fuentes de datos. SPARQL permite a sus usuarios escribir consultas sobre datos que siguen las especificaciones RDF. SPARQL provee un conjunto completo de operaciones de consultas analíticas como lo son JOIN, SORT, AGGREGATE para datos en los que su esquema forma parte de los mismos datos en lugar de requerir una especificación externa del esquema. La información del esquema (ontología) por lo general es provista de manera externa, para permitir que distintas fuentes de datos puedan ser combinadas sin ambigüedades.

2) Endpoint SPARQL: Un *endpoint* SPARQL es un servicio *Web* que permite a los usuarios consultar fuentes de datos RDF haciendo uso del lenguaje SPARQL. Los resultados son normalmente retornados en uno o varios formatos procesables. De tal manera que un *endpoint* SPARQL es concebido como una interfaz hacia una fuente de datos RDF.

3) Jena: Jena [9][10] es un sistema de administración de RDF de código abierto (*open source*), implementado en JAVA y disponible para descarga [10]. Jena ofrece una interfaz de programación (API) para la manipulación de grafos RDF como conjuntos de ternas RDF. A través de esta API, Jena permite almacenar los grafos RDF en memoria principal o en memoria secundaria, utilizando dos técnicas basadas en el modelo relacional tradicional: (1) Tabla de ternas, y (2) Tabla de propiedades.

Jena fue extendido con dos componentes: (1) ARQ [11], una máquina de ejecución para consultas SPARQL que incluye optimización basada en costos y (2) *Tuple DataBase* (TDB) [12], un sub-sistema para almacenamiento persistente de grafos RDF que ofrece integración con ARQ para procesar consultas SPARQL.

4) Spring Boot y Spring Web MVC: El *framework* Spring [13][14] es una plataforma JAVA que ofrece soporte de infraestructura para el desarrollo de aplicaciones JAVA. Spring tiene como objetivo hacer que el desarrollo de aplicaciones JAVA más sencillo y promover las buenas prácticas de programación, permitiendo una programación basada en objetos planos de JAVA (*Plain Old* JAVA *Objects* – POJO).

Spring está compuesto por diversos módulos que ayudan a tareas específicas, uno de ellos es Spring *Web* MVC, que contiene componentes para construcción de aplicaciones *Web*

Spring autónomas con configuración automática y servicios *Web* REST.

Por otra parte, Spring Boot es la solución del *framework* Spring para crear aplicaciones con la mínima configuración inicial posible, debido a que posee librerías (tanto propias como de terceros), para poder iniciar la ejecución de la aplicación con el mínimo esfuerzo.

C. Servicios Android

Un servicio (*service*) [15] en Android es un componente de aplicación capaz de realizar operaciones de larga duración en segundo plano (*background*) y no provee una interfaz gráfica de usuario. Algún otro componente de la aplicación podría iniciar un servicio y éste continuará ejecutándose en segundo plano, incluso si el usuario cambia a otra aplicación.

Adicionalmente, un componente puede enlazarse (*bind*) a un servicio para interactuar con él e incluso desempeñar comunicación entre procesos. Existen esencialmente dos tipos de servicios en Android:

- INICIADO (*started*): Un servicio es "iniciado" cuando algún componente (como por ejemplo un *Activity*) lo inicia a través de la llamada al método `startService()`. Una vez iniciado, un servicio puede ejecutarse en segundo plano de manera indefinida, incluso si el componente que lo inició es destruido por el sistema. Por lo general, un servicio iniciado ejecuta operaciones concretas y no retorna resultado alguno al componente que lo inició.
- ENLAZADO (*bound*): Un servicio es "enlazado" cuando un componente se enlaza a él a través de la llamada al método `bindService()`. Un servicio enlazado ofrece una interfaz cliente-servidor (*client-server*) que permite a otros componentes interactuar con él, enviar peticiones (*requests*), obtener resultados e incluso desempeñar comunicación entre procesos. Un servicio enlazado se ejecuta sólo mientras exista algún otro componente enlazado a él. Múltiples componentes pueden enlazarse a un mismo servicio en paralelo, pero una vez que todos ellos se desenlacen, el servicio es destruido por el sistema.

Aunque existen estos dos tipos de servicios con características bien definidas, cabe la posibilidad de crear un servicio que funcione combinando características de los dos tipos, es decir, un servicio puede ser iniciado (*started*) para ejecutarse indefinidamente y al mismo tiempo permitir que otros componentes se enlacen (*bind*) a él.

Sin importar si un servicio es iniciado, enlazado, o ambos, por defecto los servicios en Android son privados de la aplicación que los crea, es decir, sólo los componentes de esa misma aplicación pueden hacer uso del servicio, y además son creados y ejecutados en el mismo proceso de la aplicación a la que pertenecen. Sin embargo, un servicio puede programarse de manera que pueda ser utilizado por componentes de cualquier aplicación del dispositivo móvil.

III. DISEÑO DE LA SOLUCIÓN

La Figura 1 presenta un diagrama en el que se destacan los módulos principales que componen la solución del problema

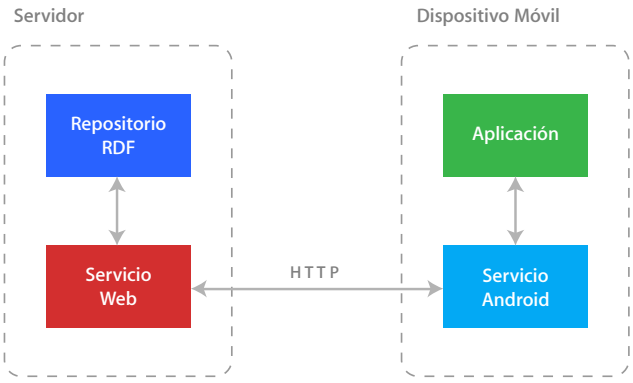

Figura 1: Componentes que Integran la Solución

planteado.

En el diagrama podemos notar que la arquitectura propuesta está compuesta por cuatro componentes principales, que cumplen un papel muy específico de la solución. En primer lugar, el *Repositorio RDF* es el encargado de almacenar y gestionar los documentos RDF. El *Servicio Web* ofrece la posibilidad de poder obtener los datos del repositorio RDF para exponerlos a través de un contenedor *Web*, que es donde reside el servicio. El *Servicio Android* es un componente en el dispositivo móvil que permite establecer la conexión con el servidor *Web* para enviarle la información necesaria, de modo que éste es capaz de responder las consultas, y posteriormente devolver los resultados al dispositivo móvil. Por último la *Aplicación Android* es quien hace uso del Servicio Android para poder consultar y obtener resultados del Repositorio RDF que toma lugar en la nube.

En las subsecciones siguientes se explican con detalle cada uno de los componentes que integran este mecanismo para el procesamiento y consulta de fuentes de datos RDF desde dispositivos móviles con Android.

A. Repositorio de Datos RDF

En esta subsección se presenta el *framework* RDF utilizado para el almacenamiento y procesamiento eficiente de documentos RDF.

Jena [10][9] es un sistema de administración de RDF de código abierto (*open source*), implementado en JAVA. En el ámbito de la solución planteada, lo primero se debe destacar es el hecho de que Jena esté implementado en JAVA, permitiendo una mejor interacción y una integración más directa entre el *framework* y el servicio *Web*. A pesar de que el lenguaje nativo de consultas de Jena es RDQL, se incluyó soporte para SPARQL a través de ARQ haciendo uso de heurísticas de optimización para lograr un procesamiento eficiente de consultas, y dado que las consultas a enviar desde el dispositivo móvil estarán en SPARQL, es primordial que el *framework* utilizado tenga capacidad de procesar las consultas en ese lenguaje.

Adicionalmente, Jena ofrece un eficiente mecanismo para el almacenamiento en memoria secundaria del repositorio RDF haciendo uso de TDB, este método de almacenamiento es el

ideal cuando se trata con fuentes de datos de gran tamaño. Además, al estar integrado con ARQ, soporta consultas en SPARQL directamente sobre el repositorio en memoria secundaria. Finalmente, Jena goza de una gran popularidad y documentación que permiten su fácil implantación, además de ser uno de los *frameworks* más eficientes y rápidos en responder consultas SPARQL.

B. Servicio Web

En esta subsección se presenta el *framework* utilizado para desplegar el servicio *Web* que forma parte del esquema de la solución, permitiendo la creación del *endpoint* SPARQL.

Spring Boot [16][17] es un *framework* de aplicación de código abierto (*open source*) y un contenedor de inversión de control (*inversion of control*) para el desarrollo de aplicaciones en la plataforma JAVA. Spring Boot es la solución del *framework* Spring [13][14] para crear aplicaciones con la mínima configuración inicial posible debido a que posee librerías, tanto propias como de terceros, para poder iniciar la ejecución de la aplicación con el mínimo esfuerzo usando la técnica de convención sobre configuración (*convention over configuration*).

Spring Boot provee la capacidad de desplegar (*deploy*) un servicio *Web* haciendo uso del *framework* Spring Web MVC [18], este ofrece una arquitectura Modelo-Vista-Controlador (*Model-View-Controller* – MVC) y componentes preparados para el desarrollo de aplicaciones *Web* flexibles. Las siguientes son algunas de las características más destacadas de Spring *Web* MVC que tienen lugar como parte del esquema de la solución planteada:

- Creación de aplicaciones Spring autónomas con configuración automática y servicios *Web* con API REST.
- Permite a los programadores desarrollar todo tipo de componentes de aplicación usando planos objetos de JAVA (*Plain Old Java Objects* – POJO).
- Provee herramientas diseñadas para la fase de producción como métricas, verificación de estado (*health check*) y configuración externalizada.

Este conjunto de características ayudan a identificar los motivos del uso de Spring Boot como *framework* para el servicio *Web* que forma parte del esquema de la solución. Spring Boot al ser un *framework* para aplicaciones de JAVA, permite una fácil, rápida y eficiente integración e interacción con Jena, por lo que Jena es utilizado como un conjunto de librerías añadidas a Spring Boot para permitir la gestión, almacenamiento y procesamiento de consultas con datos RDF.

De esta manera, gracias a que ambos *frameworks* comparten el lenguaje de programación JAVA, es posible realizar la integración de forma tal que el servicio *Web* pueda exponer el repositorio RDF a través de una API REST bien definida para el procesamiento de consultas en SPARQL. En el servicio *Web* que se ha implementado, el procesamiento de consultas SPARQL se logra una vez que el *DispatcherServlet* recibe la solicitud HTTP, extrae los datos, y los proporciona a Jena para la ejecución de la consulta en el repositorio RDF.

Una vez que el repositorio RDF (gestionado por Jena) ejecuta la consulta en el repositorio y obtiene los resultados, éstos se almacenan como un archivo dentro del servidor y se retorna al controlador la dirección del archivo recién creado, para ser devuelto en el cuerpo del mensaje de respuesta HTTP y así el archivo pueda ser luego descargado.

El formato del resultado de la consulta dependerá de lo especificado en la solicitud HTTP, actualmente se tiene soporte para RDF/XML [19], RDF/JSON [20] y texto plano. Por supuesto, el tamaño del resultado puede variar según el formato en que se exprese, pero éso no afecta el tiempo del procesamiento de la consulta.

C. Servicio Android

En esta subsección se presenta el mecanismo utilizado dentro del dispositivo móvil que permite la comunicación con el servidor *Web*, previamente descrito, para solicitar resultados de consultas escritas en lenguaje SPARQL desde el dispositivo y su posterior entrega a la aplicación que lo solicita.

Recordando un poco el marco teórico, un servicio (*service*) en Android es un componente de aplicación capaz de realizar operaciones de larga duración en segundo plano (*background*) y no provee una interfaz gráfica de usuario. Algún otro componente de la aplicación podría iniciar un servicio y éste continuará ejecutándose en segundo plano incluso si el usuario cambia a otra aplicación. Adicionalmente, un componente puede enlazarse (*bind*) a un servicio para interactuar con él e incluso desempeñar comunicación entre procesos (*Inter-Process Communication*) [15].

En líneas generales, un servicio en sí es bastante simple, ofreciendo dos funcionalidades principales:

- Facilidad para las aplicaciones de "informar" al sistema acerca de algún trabajo que se quiera ejecutar en segundo plano (incluso cuando el usuario no está directamente interactuando con la aplicación). Esto corresponde con los servicios "iniciados" (*started*) los cuales permiten programar alguna operación para ser ejecutada hasta que el mismo servicio o algún otro componente detenga su ejecución explícitamente.
- Facilidad para las aplicaciones de "exponer" algunas de sus funcionalidades a otras aplicaciones. Esto corresponde a los servicios "enlazados" (*bound*) los cuales permiten mantener una conexión consistente con algún componente a fin de interactuar con él.

Ahora bien, en el ámbito del esquema de la solución planteada, la principal tarea a cumplir por este componente es proporcionar una interfaz común en el sistema Android, que pueda ser utilizada por cualquier otro componente, para proveer la facilidad de realizar consultas y obtener sus resultados, usando el lenguaje SPARQL, sobre algún repositorio RDF que viva en la nube y no localmente.

Para lograr esto, el servicio desarrollado, que lleva por nombre *SparkleService*, combina características de los diferentes tipos de servicios. Es "enlazado" para poder exponer su funcionalidad a otros componentes de la aplicación y además proveer

una interfaz cliente-servidor que permita una comunicación consistente y directa entre el componente y SparkleService, de modo que la aplicación externa pueda enviar peticiones y obtener resultados de SparkleService. Y es "iniciado" para poder ejecutar las descargas de los resultados en segundo plano y de manera indefinida hasta que finalice y se detenga su ejecución.

Además, SparkleService es un servicio exportado, esto proporciona la capacidad de exponer su funcionalidad a todas las aplicaciones y componentes en el dispositivo móvil, sin importar si esos componentes pertenecen a la misma aplicación de la que forma parte el servicio.

SparkleService es el único componente de la aplicación a la que pertenece, de modo que al instalar dicha aplicación, sólo se estará instalando el servicio y nada más. La razón para esto es permitir que el servicio pueda ser instalado y gestionado de manera independiente a otras aplicaciones del dispositivo. Adicionalmente, al ser SparkleService el único componente, permite que éste se ejecute en un proceso diferente al de las aplicaciones externas que puedan hacer uso de él, esto tiene la ventaja de que no se vean afectadas mutuamente en caso de que alguno de los dos, por una u otra razón, sean destruidos por el sistema.

Debido a que SparkleService y la aplicación que desee usarlo se encuentran en procesos separados, es necesario utilizar mecanismos de comunicación entre procesos para el paso de mensajes entre las partes. En Android existen dos mecanismos de comunicación entre procesos: (1) A través del lenguaje AIDL (*Android Interface Definition Language*); y (2) a través de la clase `Messenger`.

Para SparkleService se utilizó la clase `Messenger` para la comunicación entre procesos, debido a su facilidad de implementación y su uso intuitivo. Este mecanismo permite definir, mediante el uso de constantes, los posibles mensajes o solicitudes que puede recibir SparkleService, al igual que los mensajes de respuesta a las aplicaciones que hagan uso de él.

Independientemente del mecanismo escogido para la comunicación entre procesos, el sistema de Android permite el paso de mensajes entre procesos con un tamaño máximo de 1 MB, razón por la cual SparkleService ofrece la posibilidad de realizar la descarga de una consulta como un archivo y almacenarlo en la memoria del dispositivo (a la que ambas partes pueden acceder) para luego notificar a la aplicación solicitante la ruta completa en donde se almacenó el archivo para que dicha aplicación pueda leerlo y procesarlo.

Para la comunicación con el servicio *Web*, SparkleService utiliza mecanismos convencionales para manejar conexiones HTTP, y de acuerdo a la solicitud que reciba el servicio, se construye la solicitud HTTP adecuada para dar respuesta a la solicitud recibida. Los componentes que deseen hacer uso de SparkleService pueden especificar el URI del *endpoint* que gestionará las consultas, sin embargo tiene una API definida para la comunicación con el *endpoint* que debe ser respetada por el mismo para la correcta comunicación entre las partes. En este caso, el servicio *Web* implementado (especificado en la sección anterior) implementa correctamente la API para una comunicación exitosa.

SparkleService tiene la capacidad para comunicar al cliente cualquier evento que suceda durante su ejecución, no únicamente comunicarle el resultado de una consulta, sino además si ocurre algún error en la comunicación con el servidor o en la consulta, le será notificado al cliente respectivo de la misma manera.

D. Aplicación Android

En esta subsección se describe la aplicación Android desarrollada que hace uso de SparkleService para realizar y recibir resultados de consultas en SPARQL al repositorio RDF almacenado en la nube y expuesto mediante el servicio *Web* desplegado, utilizando una API REST.

La aplicación Android desarrollada lleva por nombre *Sparkle*, y es una aplicación de ejemplo para demostrar cómo una aplicación externa puede hacer uso de SparkleService. Es bueno reiterar que así como la aplicación Sparkle, cualquier otra aplicación puede hacer uso de SparkleService, y Sparkle es la demostración de cómo pueden hacerlo.

El objetivo de Sparkle es demostrar cómo realizar consultas en SPARQL a repositorios RDF en la nube mediante SparkleService, de modo que la interfaz gráfica de la aplicación es bastante sencilla y enfocada a lo que se quiere mostrar. La interfaz gráfica consiste de un campo de texto en donde se introduce el URI del *endpoint* en donde se encuentra el repositorio RDF, un selector para seleccionar el formato deseado del resultado de la consulta, una área de texto que será en donde se ingresará el texto que corresponde a la consulta en lenguaje SPARQL, un *checkbox* para especificar si se quiere descargar la consulta en un archivo o no, y por último un botón para enviar la solicitud de consulta a SparkleService.

Del lado de Sparkle se implementa de igual manera un `Messenger` (llamémoslo A) que le será enviado a SparkleService para que éste sea capaz de saber el cliente al que debe responder, asimismo, el `Messenger` A es quien recibirá y manejará los mensajes enviados desde SparkleService.

Cuando Sparkle se enlaza (*bind*) con SparkleService mediante la llamada al método `bindService()`, el servicio responde enviando un *Binder* a Sparkle, que contrasta con el `Messenger` que existe dentro de SparkleService, de manera que el *Binder* es necesario para crear un segundo `Messenger` (llamémoslo B) en Sparkle, encargado de enviar las solicitudes de consulta a SparkleService.

Al pulsar el botón para ejecutar una consulta, Sparkle verifica que los campos estén correctos y de ser así se toma y se empaqueta toda la información en un mensaje que luego le es enviado a SparkleService pidiendo resolver la consulta y obtener el resultado mediante la descarga de un archivo con el resultado.

En cualquiera de los dos casos, cuando Sparkle es notificado sobre el resultado de la consulta, se abre otra vista que muestra

los resultados en un gran campo de texto, u opcionalmente en una lista de elementos utilizando un *parser* para tomar de forma individual los elementos del resultado (dependiendo del formato en que esté expresado el resultado) y listarlos.

E. Protocolo de Transmisión de Datos

Como se ha expresado múltiples veces, el esquema de comunicación para que un dispositivo móvil pueda tener acceso a la nube, será a través del uso de HTTP. Intuitivamente lo que se puede pensar de HTTP es básicamente un diálogo o lenguaje del cual un dispositivo móvil puede hacer uso para entablar una comunicación con la nube, es un protocolo de aplicación para comunicación entre dispositivos en la red.

HTTP es un protocolo a nivel de aplicación para sistemas de información hipermedia. HTTP ha sido utilizado por la *World Wide Web* desde 1990. Es un protocolo genérico que puede ser utilizado para una gran variedad de tareas más de allá de su uso para hipertexto.

HTTP es un protocolo cliente-servidor, esto quiere decir que las interacciones que son ejecutadas a través de HTTP, son iniciadas siempre por el cliente que quiere acceder o solicitar algún recurso del servidor. De modo que el servidor posee una serie de recursos que puede proveerle al cliente, y el cliente envía solicitudes al servidor solicitando alguno de esos recursos, dicho servidor responde al cliente ya sea con el recurso que solicitó o algún mensaje indicando si pudo o no procesar la solicitud [21].

Pero, ¿por qué utilizar un protocolo de navegador *Web* para comunicarse con la nube? HTTP provee ciertas propiedades, una de ellas es que obtenemos una interfaz uniforme que podemos proveer vía HTTP a servicios o recursos que se encuentran en un servidor. Por ejemplo, si el cliente es un navegador *Web* y envía una solicitud HTTP que el servidor pueda procesar, dicha solicitud es exactamente la misma que si la hubiese enviado un dispositivo móvil.

Además, existe una inversión muy significativa, tanto en los *frameworks* como en librerías, balanceadores de carga (*load balancers*) y cualquier otra infraestructura desarrollada para HTTP, que hacen natural en uso de HTTP como mecanismo de comunicación con la nube.

En el ámbito de SparkleService y el servicio *Web* implementado, se utilizan mensajes de solicitud HTTP para enviar toda la información necesaria al servicio *Web* de modo que éste sea capaz de procesar la consulta en SPARQL solicitada. En esa solicitud se utiliza el método POST de HTTP pues éste permite enviar una cantidad de datos indefinida en el cuerpo (*body*) de la solicitud HTTP, es de hecho en el cuerpo donde se envía el *string* que representa la consulta en lenguaje SPARQL.

De manera análoga, también se utilizan mensajes de respuesta HTTP para recibir la información necesaria para que el cliente sea capaz de descargar el archivo con el resultado de la consulta en el formato especificado en la solicitud. La información recibida en este mensaje de respuesta HTTP es un *string* en el cuerpo de la respuesta en formato JSON representando un objeto con la información necesaria para el análisis y la descarga del archivo que guarda el resultado de la consulta; este objeto JSON incluye hora de respuesta (*timestamp*), nombre generado del archivo (*filename*), URL del archivo y tamaño del archivo (*size*).

IV. Resultados Experimentales

A. Configuración de los Experimentos

1) Plataforma Computacional: Para realizar los experimentos se requirieron de tres equipos: (1) un computador que juega el papel del servidor (la nube) que contiene el repositorio RDF y lo expone a través del servicio *Web*, (2) un dispositivo móvil con Android que juega el papel del cliente donde reside el servicio y la aplicación Android que se comunica con el servidor que contiene el repositorio RDF para realizar consultas y, (3) un enrutador (*router*) al cual están conectados de manera inalámbrica el servidor y el dispositivo móvil para así permitir la comunicación entre ambos mediante una red inalámbrica de área local (WLAN) utilizando el protocolo HTTP. Además de los equipos, fue necesario tener instalado Java para el desarrollo y ejecución de los experimentos. A continuación se describe las características de cada componente:

- Java(TM) SE Runtime Environment (build 1.7.0_71-b14).
- El servidor es un computador con procesador *Intel Core i7-3820* de 4 núcleos a 3.6 GHz y 16 GB de memoria RAM a 2133 MHz bajo el sistema operativo Windows 8.1 de 64 bits.
- El dispositivo móvil es un teléfono LG Nexus 5 con procesador *Qualcomm Snapdragon* 800 *Quad-core* a 2.3 GHz y 2 GB de memoria RAM bajo el sistema operativo Android 5.1 Lollipop.
- El *router* es un TP-LINK TL-WR941ND de 300 Mbps a 2.4 GHz.

2) Casos de Prueba: Los experimentos se realizaron sobre distintas fuentes de datos, tanto sintetizadas como reales. Los datos sintetizados utilizados en este estudio experimental fueron generados por el Lehigh University *Benchmark* (LUBM) [22]. Este *benchmark* está basado en la ontología Univ-Bench, la cual describe universidades, departamentos y las actividades académicas que ocurren en ellos. Cada fuente de datos está representada por la expresión $Univ(N, S)$, la cual denota la fuente de datos que contiene N universidades generadas a partir de la semilla S, comenzando por el índice 0. Para efectos de este estudio experimental, se generaron cinco fuentes de datos de prueba: $Univ(1, 0)$, $Univ(5, 0)$, $Univ(10, 0)$, $Univ(20, 0)$ y $Univ(50, 0)$, las cuales contienen datos de 1, 5, 10, 20 y 50 universidades, respectivamente. Las características de estas fuentes de datos (número de ternas RDF que contienen y tamaño en disco) se muestran en la Tabla I.

Para los datos reales se utilizaron dos fuentes de datos de DBpedia [23] el cual provee datos extraidos del contenido de Wikipedia distribuidos en distintas fuentes de datos. La primera fuente de datos contiene información de geolocalización en latitud y longitud de todos los lugares alrededor

Tabla I: Características de las Fuentes de Datos Sintetizadas

Fuente de datos	Número de ternas RDF	Tamaño (MB)
$Univ(1,0)$	103 074	17,27
$Univ(5,0)$	645 649	108,28
$Univ(10,0)$	1 316 322	220,79
$Univ(20,0)$	2 781 322	468,68
$Univ(50,0)$	6 888 642	1 163,49

Tabla II: Características de las Fuentes de Datos Reales

Fuente de datos	Número de ternas RDF	Tamaño (MB)
$Dataset\ \#1$	2 105 024	297,475
$Dataset\ \#2$	15 849 011	2 115,141

del planeta que están indizados en Wikipedia, esta fuente será referida como *Dataset* #1. La segunda fuente de datos es una combinación entre los datos de geolocalización de la primera fuente, e información muy específica y detallada de propiedades para cada uno de los lugares indizados en Wikipedia, esta fuente será referida como *Dataset* #2. Las características de estas fuentes de datos (número de ternas RDF que contienen y tamaño en disco) se muestran en la Tabla II.

En esta parte del estudio experimental se consideraron dos conjuntos de consultas a evaluar:

- *Benchmark* 1: El primer banco de pruebas pertenece a LUBM. LUBM ofrece 13 consultas de prueba, las cuales toman en consideración los siguientes factores: tamaño de la entrada, selectividad, complejidad, utilización de información jerárquica y capacidad de inferencia lógica. Las consultas de prueba fueron diseñadas en términos de los criterios antes mencionados con énfasis en tamaño grande de la entrada y alta selectividad. En este estudio experimental se modificaron las consultas que involucran la tarea de razonamiento semántico de modo que no sea necesario aplicarlo para darles respuesta.
- *Benchmark* 2: El segundo banco de pruebas pertenece a las fuentes de datos reales de DBpedia, y consiste en dos conjuntos de cinco consultas conjuntivas, el primer conjunto para el *Dataset* #1 y el segundo conjunto para el *Dataset* #2; las consultas fueron diseñadas respetando los criterios de diseño descritos en el *benchmark* 1, así que tampoco involucran la tarea de razonamiento semántico para ser respondida.

3) Métricas de Evaluación:

- *Tiempo de respuesta de las consultas:* tiempo de respuesta (medido en segundos) de las consultas de prueba, cada una de las cuales fue ejecutada cinco veces sobre las distintas fuentes de datos y, posteriormente, promediados los resultados de las ejecuciones. La medición de este valor se realizó a nivel de código, tomando el instante de tiempo previo a la ejecución de la consulta, y el instante de tiempo al finalizar la consulta para luego calcular su diferencia.
- *Tiempo de almacenamiento de los resultados:* tiempo promedio que toma procesar la consulta para darle formato y almacenarla en un archivo local en el servidor para su posterior transmisión al cliente (medido en segundos), se tomarán en cuenta los formatos RDF/XML [19], RDF/JSON [20] y texto plano. Para la medición de este valor se tomaron los tiempos sin el almacenamiento y los tiempos con el almacenamiento para calcular la diferencia entre ellos promediada a cinco veces.
- *Tamaño en disco del archivo con los resultados:* tamaño de archivo con los resultados de cada consulta (medido en MB). Para obtener este valor se consultó directamente el tamaño del archivo resultante en el explorador de archivos del sistema operativo.
- *Tiempo de transmisión de los resultados al dispositivo móvil:* tiempo promedio que toma la transferencia del archivo con los resultados de la consulta del servidor al dispositivo móvil (medido en segundos). Para la medición de este valor se toma el tiempo en que el dispositivo móvil pide al servidor la descarga del archivo y el tiempo en que finaliza la descarga en el dispositivo móvil para luego calcular su diferencia, esto se repite cinco veces para así obtener el promedio.
- *Tiempo total de la operación de consulta desde el dispositivo móvil:* tiempo promedio (medido en segundos) que toma la totalidad de la operación de consulta desde el dispositivo móvil, esto es, desde que el dispositivo solicita al servidor responder una consulta, hasta que los resultados terminan de descargarse. Por consiguiente, este tiempo incluye todas las métricas de tiempo citadas en esta sección. Para la medición de este valor se toma el instante de tiempo en que el dispositivo realiza la solicitud, y el instante de tiempo en que finaliza la descarga de los resultados para luego calcular su diferencia, este proceso se realizó cinco veces por cada consulta para calcular el promedio.

B. Análisis de los Resultados

1) Hipótesis Iniciales:

- Dado un formato solicitado por el dispositivo móvil para la representación del resultado de la consulta, este resultado tomará un tiempo diferente de almacenamiento en el archivo del lado del servidor y, posteriormente, el tiempo de transferencia al dispositivo móvil. Esto se debe a que cada formato posee un algoritmo de procesamiento distinto y cada representación tiene una estructura diferente, lo que resulta en que un mismo archivo resultante pueda tener diferentes tamaños en disco dependiendo del formato.
- La velocidad de transferencia (o velocidad de descarga) del archivo resultante de una consulta al dispositivo móvil tiende a depender del ancho de banda disponible en la red local. Además, esta velocidad no necesariamente se mantiene en el mismo valor durante el tiempo de descarga. Sabiendo esto, es posible que un mismo archivo tome distintos tiempos de descarga.
- Debe existir una diferencia en el tiempo de respuesta de las consultas evaluadas en *cold cache* y en *warm cache*: el tiempo de evaluación de consultas en *warm cache* debe ser menor que en *cold cache* en el sistema de administración de datos RDF considerado (Jena), el cual debe estar en

Tabla III: Tiempo Total (Segundos) de la Operación de Consulta desde el Dispositivo Móvil sobre $Univ(1,0)$ de LUBM

Consultas	RDF/XML		RDF/JSON		Texto Plano	
	cold	*warm*	*cold*	*warm*	*cold*	*warm*
Q01	1,009	0,702	0,990	0,625	0,978	0,590
Q02	1,095	0,651	1,078	0,622	1,008	0,578
Q03	1,066	0,613	1,007	0,582	0,974	0,567
Q04	1,134	0,631	1,071	0,616	1,053	0,570
Q05	1,226	0,965	1,205	0,780	1,148	0,663
Q06	2,088	1,452	2,057	1,417	1,926	1,201
Q07	1,144	0,708	1,098	0,643	1,037	0,624
Q08	3,656	2,480	3,190	2,100	2,613	1,544
Q09	1,351	0,781	1,281	0,687	1,166	0,625
Q10	1,077	0,751	1,072	0,689	1,055	0,593
Q11	1,144	0,737	1,033	0,682	1,127	0,650
Q12	1,034	0,649	1,006	0,631	0,968	0,623
Q13	1,619	0,887	1,502	0,875	1,441	0,787
Promedio	1,434	0,924	1,353	0,842	1,269	0,740

capacidad de mejorar su rendimiento si existen datos precargados en la memoria caché.

- Debe existir una relación entre la cardinalidad del resultado de una consulta y el tamaño del archivo que almacena ese resultado, entendiendo que, a mayor cardinalidad, mayor será el tamaño del archivo dado que se deben almacenar más datos.

2) Resultados Obtenidos: Solamente se mostrarán los resultados correspondientes a la quinta métrica de evaluación (tiempo total de la operación de consulta desde el dispositivo móvil) la cual, a su vez, incluye los resultados parciales de las otras métricas de tiempo ya descritas.

En este apartado se estudia el comportamiento y la eficiencia del esquema implementado sobre la tarea de enviar una solicitud y obtener los resultados mediante su descarga. Para esto, se analizan los resultados experimentales obtenidos con respecto al tiempo promedio (medido en segundos) del total de la operación de consulta, desde que el dispositivo móvil realiza la solicitud al servidor, hasta que se termina de descargar el archivo con los resultados. Cada consulta fue ejecutada cinco veces sobre las distintas fuentes de datos consideradas y almacenadas previamente en el repositorio RDF del lado del *endpoint*.

BENCHMARK 1: En las Tablas III, IV, V, VI y VII se presentan los resultados obtenidos con respecto al tiempo total de la operación de consulta desde el dispositivo por cada una de las 13 consultas para las cinco fuentes de datos sintetizadas de LUBM en *cold cache* y en *warm cache*. En cada tabla se hace una comparación entre los formatos RDF/XML [19], RDF/JSON [20] y texto plano.

BENCHMARK 2: En las Tablas VIII y IX se presentan los resultados obtenidos con respecto al tiempo total de la operación de consulta desde el dispositivo por cada uno de los bancos de cinco consultas para las fuentes de datos reales provistas por DBpedia en *cold cache* y en *warm cache*. En cada tabla se hace una comparación entre los formatos RDF/XML [19], RDF/JSON [20] y texto plano.

Tabla IV: Tiempo Total (Segundos) de la Operación de Consulta desde el Dispositivo Móvil sobre $Univ(5,0)$ de LUBM

Consultas	RDF/XML		RDF/JSON		Texto Plano	
	cold	*warm*	*cold*	*warm*	*cold*	*warm*
Q01	1,084	0,717	1,059	0,667	1,029	0,562
Q02	1,831	0,807	1,768	0,728	1,659	0,708
Q03	1,139	0,660	1,078	0,644	1,039	0,612
Q04	1,082	0,618	1,068	0,747	1,064	0,718
Q05	1,389	0,860	1,261	0,810	1,217	0,664
Q06	5,873	3,881	5,136	3,617	4,402	2,963
Q07	1,240	0,806	1,208	0,722	1,187	0,685
Q08	3,678	2,644	3,472	2,349	2,835	1,757
Q09	2,493	0,886	2,318	0,846	2,239	0,777
Q10	1,115	0,711	1,050	0,663	1,002	0,624
Q11	1,154	0,773	1,124	0,667	1,047	0,647
Q12	1,168	0,711	1,063	0,690	1,031	0,662
Q13	3,669	2,491	3,510	2,575	2,670	1,538
Promedio	2,070	1,274	1,932	1,210	1,725	0,994

Tabla V: Tiempo Total (Segundos) de la Operación de Consulta desde el Dispositivo Móvil sobre $Univ(10,0)$ de LUBM

Consultas	RDF/XML		RDF/JSON		Texto Plano	
	cold	*warm*	*cold*	*warm*	*cold*	*warm*
Q01	1,096	0,691	1,072	0,669	1,059	0,637
Q02	2,664	0,795	2,546	0,694	2,482	0,631
Q03	1,168	0,654	1,092	0,641	1,058	0,589
Q04	1,204	0,681	1,139	0,671	1,055	0,710
Q05	1,378	0,808	1,195	0,788	1,171	0,784
Q06	11,575	8,699	10,346	7,588	7,326	4,535
Q07	1,223	0,771	1,120	0,745	1,105	0,685
Q08	3,986	2,943	3,539	2,527	3,133	2,014
Q09	3,136	0,927	3,109	0,910	3,128	0,879
Q10	1,100	0,730	1,065	0,662	0,972	0,652
Q11	1,222	0,754	1,109	0,732	1,082	0,682
Q12	1,087	0,665	1,066	0,623	1,049	0,607
Q13	4,577	3,190	1,066	2,759	3,283	2,230
Promedio	2,724	1,716	2,266	1,539	2,146	1,203

Tabla VI: Tiempo Total (Segundos) de la Operación de Consulta desde el Dispositivo Móvil sobre $Univ(20,0)$ de LUBM

Consultas	RDF/XML		RDF/JSON		Texto Plano	
	cold	*warm*	*cold*	*warm*	*cold*	*warm*
Q01	1,125	0,689	1,083	0,671	1,067	0,661
Q02	6,137	0,902	5,634	0,873	5,294	0,843
Q03	1,175	0,700	1,124	0,616	1,069	0,611
Q04	1,177	0,756	1,166	0,671	1,104	0,633
Q05	1,389	0,893	1,210	0,729	1,189	0,686
Q06	24,473	21,217	19,837	16,355	11,845	8,866
Q07	1,190	0,773	1,143	0,766	1,122	0,747
Q08	3,882	2,849	3,582	2,527	2,836	1,718
Q09	5,131	1,313	5,085	1,132	4,808	1,109
Q10	1,127	0,705	1,114	0,669	1,051	0,629
Q11	1,159	0,759	1,141	0,707	1,116	0,643
Q12	1,237	0,745	1,174	0,738	1,138	0,715
Q13	8,213	6,901	7,569	5,976	5,281	3,888
Promedio	4,417	3,016	3,912	2,495	2,994	1,673

Los resultados del presente experimento corroboran los valores obtenidos para las otras métricas de evaluación para ambos *benchmarks*, el formato texto plano es ideal si se quisiera sólo visualizar los resultados, más no interpretarlos o manipularlos.

Tabla VII: Tiempo Total (Segundos) de la Operación de Consulta desde el Dispositivo Móvil sobre $Univ(50, 0)$ de LUBM

Consultas	RDF/XML		RDF/JSON		Texto Plano	
	cold	*warm*	*cold*	*warm*	*cold*	*warm*
Q01	1,090	0,706	1,072	0,680	1,043	0,653
Q02	5,578	1,710	5,249	1,462	5,124	1,179
Q03	1,158	0,687	1,085	0,664	0,997	0,586
Q04	1,104	0,748	1,068	0,700	1,037	0,668
Q05	1,467	0,898	1,282	0,806	1,221	0,688
Q06	62,091	57,493	56,994	52,418	41,861	37,096
Q07	1,220	0,797	1,201	0,749	1,079	0,756
Q08	4,557	2,994	3,519	2,455	3,156	1,981
Q09	2,743	1,703	2,578	1,533	2,416	1,399
Q10	1,147	0,721	1,094	0,683	1,074	0,652
Q11	1,184	0,735	1,109	0,698	1,048	0,673
Q12	1,179	0,789	1,171	0,750	1,152	0,735
Q13	23,257	20,254	17,315	14,559	13,998	10,806
Promedio	8,290	6,941	7,287	6,012	5,785	4,452

Tabla VIII: Tiempo Total (Segundos) de la Operación de Consulta desde el Dispositivo Móvil sobre el *Dataset* #1 de DBpedia

Consultas	RDF/XML		RDF/JSON		Texto Plano	
	cold	*warm*	*cold*	*warm*	*cold*	*warm*
Q01	8,675	3,842	8,433	3,814	8,024	3,745
Q02	12,689	7,556	12,124	7,029	12,009	6,628
Q03	11,157	6,638	10,535	6,500	10,316	6,322
Q04	13,753	7,978	13,463	7,623	12,981	7,355
Q05	35,766	33,442	33,412	30,494	27,901	25,527
Promedio	16,408	11,891	15,593	11,092	14,246	9,915

Tabla IX: Tiempo Total (Segundos) de la Operación de Consulta desde el Dispositivo Móvil sobre el *Dataset* #2 de DBpedia

Consultas	RDF/XML		RDF/JSON		Texto Plano	
	cold	*warm*	*cold*	*warm*	*cold*	*warm*
Q01	1,135	0,749	1,049	0,684	1,006	0,642
Q02	10,839	8,621	9,104	6,194	7,773	5,214
Q03	1,423	0,854	1,273	0,833	1,243	0,805
Q04	3,026	1,643	3,062	1,726	2,956	1,604
Q05	1,506	0,878	1,478	0,816	1,350	0,841
Promedio	3,586	2,549	3,193	2,051	2,866	1,821

En efecto, en todas las tablas del experimento, el promedio del tiempo total de la operación de consulta es menor en texto plano, tanto en las fuentes de datos reales como sintetizadas, en *cold cache* y *warm cache*.

En este mismo sentido, el formato RDF/JSON es el más adecuado cuando se pretende manejar y/o interpretar los resultados. Esto se confirma en estos resultados tanto para las fuentes de datos sintetizadas como las reales en *cold cache* y *warm cache* dado que el promedio del tiempo total de la operación de consulta en formato RDF/JSON es menor a la misma medida en formato RDF/XML. A pesar de que el formato RDF/JSON toma más tiempo que RDF/XML para dar formato y almacenar los resultados de la consulta, RDF/JSON lo compensa al generar un archivo más ligero que se traduce en menor tiempo de descarga.

Los experimentos para esta métrica fueron totalmente independientes a los experimentos de las métricas anteriores, por lo que los tiempos de descarga pueden diferir ligeramente respecto al experimento de la métrica previa, dado que la velocidad de descarga, a pesar de que en promedio fue 1.1 Mbps, puede variar durante la descarga, afectando así, el tiempo de la misma, pero al estar promediados todos los valores a cinco veces, esta diferencia se torna menos notoria al generar un valor más preciso.

En general, los tiempos de las consultas en *warm cache* son mejores que los de las consultas en *cold cache* en hasta dos órdenes de magnitud, dando a entender que conforme el servidor reciba más solicitudes de consulta, éste las responderá con mayor rapidez, lo cual resulta muy conveniente desde un ámbito de plataforma móvil como lo es Android. La excepción a la regla es la Tabla VIII en donde la diferencia de tiempos entre *cold cache* y *warm cache* no es tan notoria atribuyendo este hecho a la alta selectividad del banco de consultas para esa fuente de datos, aún así, el tiempo de respuesta en *warm cache* tiende a mejorar en el tiempo.

En conclusión, se puede decir que dados estos resultados de los experimentos, se observa que el esquema de solución planteado genera unos resultados bastante aceptables para dar solución al problema planteado en las métricas consideradas, dado que los resultados muestran que es viable la consulta en lenguaje SPARQL a un repositorio RDF almacenado en la nube, tanto en tiempo como en espacio.

El problema principal, además del tiempo de respuesta, es el espacio en memoria del dispositivo, y en los resultados obtenidos, la consulta Q06 de LUBM sobre la fuente de datos $Univ(50, 0)$ es la que produce un resultado con mayor cardinalidad, generando un archivo de $47,3$ MB en formato RDF/JSON y $57,08$ MB en formato RDF/XML. Sin embargo, esto es un tamaño inferior al de la fuente de datos sobre la que se obtuvieron los resultados, la cual cuenta con un tamaño de $1\ 163,49$ MB. De esta manera, los resultados de la consulta Q06 de LUBM son más fáciles de manipular y almacenar en el dispositivo móvil.

V. Conclusiones y Trabajo Futuro

Con respecto a los resultados del estudio experimental se puede mencionar las siguientes conclusiones:

- En cuanto al tiempo de respuesta de las consultas en el *endpoint*, existe una diferencia importante entre los tiempos de respuesta de las consultas en *cold cache* y *warm cache*, diferencia que mantiene su notoriedad a través de todas las fuentes de datos independientemente del número de ternas de las mismas. Debido a esto, se puede inferir que una vez desplegado el servicio *Web* para procesar las consultas entrantes, éste será capaz de responderlas cada vez más rápido conforme al número de solicitudes que vaya procesando. Adicionalmente, es destacable el hecho de que hay un grupo de consultas en las que el tiempo de respuesta se mantiene en valores muy cercanos tanto en *cold cache* como en *warm cache* sobre las distintas fuentes de datos, en contraste, hay otras consultas de las que puede observarse cómo los tiempos varían en función del número de ternas de la fuente de datos sobre la cual se ejecuten. Esto último puede

atribuirse a que las consultas en las que el tiempo varía en función del número de ternas de la fuente de datos, tienen un poder de selectividad mayor y por consiguiente, requieren un proceso más exhaustivo al momento de seleccionar las ternas resultantes.

- Con respecto al tiempo que toma el *endpoint* en almacenar los resultados de las consultas en un archivo según el formato solicitado, en promedio es razonable, aún cuando la cardinalidad de la solución es alta. El formato RDF/JSON es, en promedio, el que consume más tiempo para dar formato a los resultados y almacenarlo en el archivo. El formato texto plano, en contraste, es el que consume menos tiempo en aplicar formato y almacenarse, esto podría llevar a pensar que este formato es el más conveniente para expresar resultados. Sin embargo, el formato texto plano está diseñado principalmente para visualizar su contenido, los formatos RDF/XML y RDF/JSON son los adecuados si se quisieran procesar e interpretar los resultados.
- En cuanto al tiempo de descarga de los resultados y el tamaño de los mismos en disco, el formato texto plano fue, en promedio, el más ligero en cuanto tamaño y el que consumió menos tiempo durante la descarga en el dispositivo móvil, pues como es lógico pensar, al ser el este formato el que ocupa menos espacio, también es el más rápido en descargar por tener menos datos que transferir. Si se requiere que los resultados se una consulta puedan ser procesados e interpretados, la mejor opción es el formato RDF/JSON, pues a pesar de que este formato en el experimento anterior resultó ser el más lento, en esta ocasión compensa ese tiempo de formato y almacenamiento en buena escala, siendo finalmente más ligero y por consiguiente más rápido al momento de la descarga que RDF/XML. El formato JSON en general, cuenta con *parsers* que presentan un rendimiento superior en la plataforma Android en comparación con el formato XML, haciendo así, más rápido y eficiente el manejo de datos en formato JSON en el dispositivo móvil.
- En referencia al tiempo total de la operación de consulta desde el dispositivo móvil, los resultados corroboran los valores obtenidos en los experimentos anteriores. Ya se había llegado a la conclusión de que el formato texto plano es ideal si se quisiera sólo visualizar los resultados, más no manipularlos, y en efecto, en este experimento el promedio del tiempo total de la operación de consulta es menor en texto plano, tanto en las fuentes de datos reales como sintetizadas en *cold cache* y *warm cache*. También se había mencionado que el formato RDF/JSON es el más adecuado cuando se pretende manipular e interpretar los resultados, esto queda confirmado en este experimento dado que, en promedio, el tiempo total de la operación de consulta en formato RDF/JSON es menor a la misma medida en formato RDF/XML tanto para fuentes de datos sintetizadas como reales en *cold cache* y *warm cache*. Esto a pesar de que el formato RDF/JSON consume más tiempo que RDF/XML para dar formato y almacenar los resultados de una consulta, pero RDF/JSON lo compensa al generar un archivo más ligero que se traduce en menor tiempo de descarga. De manera general, los tiempos de las consultas en *warm cache* son mejores que los de las consultas en *cold cache* en hasta dos órdenes de magnitud, dando a entender que, conforme el servidor reciba más solicitudes de consulta, éste las responderá con mayor rapidez, lo cual resulta muy conveniente desde un ámbito de plataforma móvil como lo es Android

Finalmente, los resultados obtenidos muestran que, en el caso promedio, el esquema planteado presenta un rendimiento razonable con respecto a todas las métricas de evaluación que se definieron para el estudio experimental sobre las fuentes de datos sintetizadas y reales consideradas. En general se obtuvo que el formato RDF/JSON es el más adecuado para consultar desde un dispositivo móvil por ser el más ligero y el más rápido de interpretar en el dispositivo móvil, al ser más ligero en tamaño, se mostró que el tiempo de descarga es menor, aunque hay que notar que RDF/XML también funciona para la tarea de interpretar los resultados. Texto plano por su lado, es el más conveniente cuando se necesite únicamente visualizar los resultados sin requerir que sean procesados dado que este formato es el que resultó ser más ligero y más rápido de almacenar y transferir.

A pesar que los experimentos realizados proporcionan evidencia de las bondades de las soluciones obtenidas por el mecanismo desarrollado en esta investigación, es necesario realizar un estudio más profundo que permita analizar el rendimiento y comportamiento del servicio SparkleService y la aplicación Sparkle al ejecutar otras tareas. Como trabajo futuro se propone realizar las siguientes extensiones al prototipo desarrollado:

- Añadir soporte para los tipos de consultas no considerados en el estudio experimental, actualmente sólo tiene soporte para consultas de tipo `SELECT`, por lo que se recomienda ampliarlo a las consultas de tipo `CONSTRUCT`, `ASK` y `DESCRIBE`.
- Para una gestión más completa de documentos RDF desde el dispositivo móvil, se recomienda añadir soporte a la operación `UPDATE` de SPARQL, de esta manera, podrían insertarse nuevas ternas RDF en un grafo RDF en la nube o, de manera análoga, eliminar ternas RDF.

REFERENCES

[1] *Android*, http://www.android.com
[2] A. Martínez, *BHyper: Procesamiento Eficiente de Documentos RDF con Hipergrafos Dirigidos*, PhD Thesis, Universidad Simón Bolívar, Caracas, Venezuela, April 2012.
[3] *Guía Breve de Web Semántica*, http://www.w3c.es/Divulgacion/GuiasBreves/WebSemantica.
[4] *AndroJena - Jena Android Porting*, https://code.google.com/p/androjena.
[5] *OpenRDF.org, home of Sesame*, http://www.openrdf.org.
[6] *ARQoid - Porting of Jena's ARQ to the Android Platform*, https://code.google.com/p/androjena/wiki/ARQoid.
[7] S. Harris, A. Seaborne, and E. Prud'hommeaux, *SPARQL 1.1 Query Language*, W3C Recommendations, Tech. Rep., 2013.
[8] J. Broekstra, A. Kampman, and F. V. Harmelen, *Sesame: A Generic Architecture for Storing and Querying RDF and RDF Schema*, in proceedings of the 1st International Semantic Web Conference (ISWC 2002), series Lecture Notes in Computer Science, vol. 2342, ISWC. Sardinia, Italy, June 2002, pp. 54–68.

[9] J. Carroll, I. Dickinson, C. Dollin, D. Reynolds, A. Seaborne, and K. Wilkinson, *Jena: Implementing the Semantic Web Recommendations*, in proceedings of the 13th International World Wide Web Conference, series Alternate Track Papers & Posters. New York, USA, May 2004, pp. 74–83.

[10] *Jena – A Free and Open Source Java Framework for Building Semantic Web and Linked Data Applications*, https://jena.apache.org.

[11] *ARQ – A SPARQL Processor for Jena*, http://jena.sourceforge.net/ARQ.

[12] *TDB – A SPARQL Database for Jena*, https://jena.apache.org/documentation/tdb.

[13] P. Webb, D. Syer, J. Long, S. Nicoll, R. Winch, A. Wilkinson, M. Overdijk, C. Dupuis and S. Deleuze, *Spring Framework Reference Documentation*, 4th edition, Pivotal Software, 2014.

[14] *Spring Framework - Core Support for Dependency Injection, Transaction Management, Web Applications, Data Access, Messaging, Testing and More*, http://projects.spring.io/springframework.

[15] *Android Services*, http://developer.android.com/guide/components/services.html.

[16] P. Webb, D. Syer, J. Long, S. Nicoll, R. Winch, A. Wilkinson, M. Overdijk, C. Dupuis and S. Deleuze, *Spring Boot Reference Guide*, 1st edition, Pivotal Software, 2015.

[17] *Spring Boot - An Opinionated View of Building Production-ready Spring Applications*, http://projects.spring.io/spring-boot.

[18] P. Webb, D. Syer, J. Long, S. Nicoll, R. Winch, A. Wilkinson, M. Overdijk, C. Dupuis and S. Deleuze, *Web MVC Framework*, 4th edition, Pivotal Software.

[19] D. Beckett, *RDF 1.1 XML Syntax Specification*, W3C Recommendations, Tech. Rep., February 2014.

[20] A. Seaborne, K. G. Clark, L. Feigenbaum, and E. Torres, *SPARQL 1.1 Query Results JSON Format*, W3C Recommendations, Tech. Rep., March 2013.

[21] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee, *Hypertext Transfer Protocol - HTTP/1.1, RFC 2324*, June 1999.

[22] *Lehigh University Benchmark (LUBM)*, http://swat.cse.lehigh.edu/projects/lubm.

[23] *DBpedia*, http://wiki.dbpedia.org.