A Storage, Retrieval, and Application Platform for Ultra-Large-Scale Linked Data

Yongju Lee
School of Computer Science and Engineering
Kyungpook National University,
Daegu, Korea

Jeonghong Kim
School of Computer Science and Engineering
Kyungpook National University,
Daegu, Korea

Changsu Kim
School of Business
Yeungnam University, Gyeongsan,
Gyeongbuk, Korea

***Abstract*—Recently, a very pragmatic approach towards achieving the Semantic Web has gained some traction with Linked Data. While many standards, methods, and technologies are applicable for Linked Data, there are still a number of open problems in the area of Linked Data. Particularly, there have been tremendous amounts of efforts of Semantic Web community to develop the Linked Data platform but there is still lack of a systematic approach on architectures, technologies, and applications. Therefore we propose a novel storage, retrieval, and application platform and investigate in detail the storage and indexing, query processing, and application approaches for the platform. We also present a semantic Mashup application as an example of the proposed approaches.**

***Keywords*—Linked Data; platform; RDF store; SPARQL; Mashup applications**

I. INTRODUCTION

Linked Data refers to a set of best practices for publishing and interlinking structured data on the Web [1]. These practices were introduced by Tim Berners-Lee in his Web architecture note Linked Data: the *Linked Data principles* [2]. The basic idea of Linked Data is to apply the general architecture of the Web to the task of sharing structured data on global scale. Technically, Linked Data is employing URIs (Uniform Resource Identifications), RDF (Resource Description Framework), and HTTP (Hypertext Transfer Protocol) to publish structured data and to connect related data that is distributed across multiple data resources. Over the past eight years, a large number of data provides have begun to adopt the Linked Data principles. Fig. 1 depicts the growth of Linked Data originating from the W3C LOD(Linked Open Data) project [3].

RDF is the data model for Linked Data, and SPARQL is the standard query language for this data model. All data items in RDF are represented in triples of the form (*subject, predicate, object*). Since RDF triples are modeled as graphs, we cannot directly adopt existing solutions from relational databases and XML technologies [4]. Thus, we need to discuss how Linked Data should be stored, indexed, and retrieved. Spurred by efforts like LOD project, a large amount of semantic data are available in the RDF formant in many fields such as science, business, bioinformatics, social networks, etc. These large volumes of RDF data motivate the need for scalable native RDF data management solutions capable of efficiently storing, indexing, and querying RDF data. In this paper, therefore, we propose a storage, retrieval, and application platform for large scale Linked Data. We present an overview the Linked Data lifecycle workflow architecture, and discuss individual components with regard to storage and indexing, query processing, and application approaches of Linked Data.

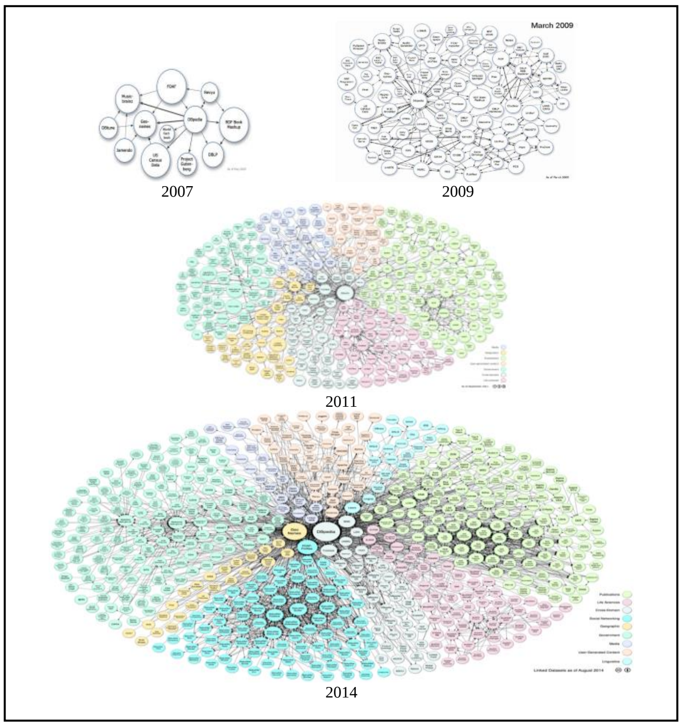

Fig. 1. Growth of the Linking Open Data Cloud

II. LINKED DATA PLATFORM

The platform has a data workflow architecture (see Fig. 2) that has been implemented using a graph-based RDF store and open source semantic components from the Semantic

Web community. Our platform's workflow consists of four steps: *Data acquisition, Building ontology, Storage & Indexing*, and *Retrieval & Analysis*. These steps do not exist in isolation but mutually fertilize themselves.

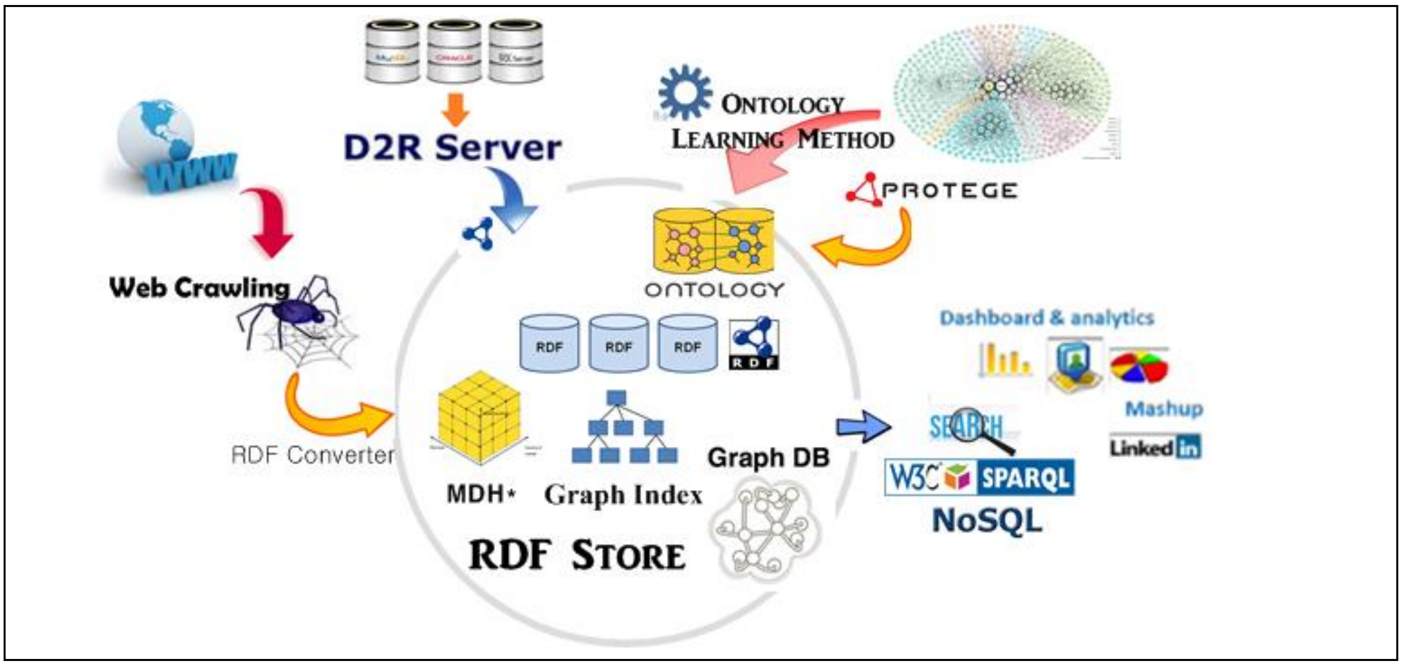

Fig. 2. Data workflow architecture for Linked Data

A. *Data Acquisition*

Information represented in unstructured form or structured representation formalism must be mapped to the RDF data model in order to be used within the workflow-based platform. The proposed system crawls Web sites and extracts unstructured data. We used a semi-automatic scraping procedure for each website to extract key properties. This procedure is based on a crawler such as Scrapy [5], the open source scraping framework. The extracted properties are then transformed into RDF triples. Structure data, e.g. relational databases, are transformed into RDF triples using D2R Server [6]. D2R Server is an open source tool for publishing relational databases on the Linked Data.

B. *Building Ontology*

RDFS(RDF Schema) and OWL(Web Ontology Language) are key Semantic Web technologies that give you a way to write down rich descriptions of your RDF data. Protege [7] is an open source ontology editor and a knowledge acquisition system. It is the leading ontological engineering tool. To ensure the successful employment of Linked Data, it is essential that they rely on the use of high quality ontologies. In spite of using the ontological engineering tool, building such ontologies is difficult and costly, thus hampering Linked Data deployment. We investigate a new ontology learning method to generate ontologies automatically.

C. *Storage and Indexing*

Once there is a critical mass of RDF data, mechanisms have to be in place to store and index this data efficiently. Our platform uses the graph-based RDF store. With this store, the platform provides a SPARQL endpoint that allows any user to access the stored RDF triples via more expressive graph queries. Our system indexes the triples stored in the RDF store. These triples are mapped into multidimensional histograms stored in an MDH* index structure.

D. *Retrieval and Analysis*

Our platform provides search capabilities such as SPARQL and NoSQL over the RDF triples. In addition to search, we can advance to build various applications based on Linked Data, For instance, we can implement an aggregation of dashboards that presents various business analytics computed on the Linked Data.

III. TECHNOLOGIES ENABLING LINKED DATA PLATFORM

This chapter aims to explain the individual components with regard to storage and indexing, query processing, and application approaches for the platform. Due to space limitations, we omit a detail discussion of the data acquisition as well as building ontology of the platform.

A. *Storage and Indexing*

There are two approaches to store Linked Data [8, 9]. First, we can maintain independent data copies in a local storage, benefiting from convenient conditions for efficient query processing, which we call "*local approach*," The second approach is based on accessing distributed data on-the-fly using link traversal, which we call "*distributed approach*."

Local approaches are copying data into a centralized registry in a manner similar to search engines for the Web of documents. By using such a registry, it is possible to provide excellent query response times. However, there are a number of drawbacks. First, query results may not reflect the most recent data. Second, storing all data may be expensive and the performance penalty can be high as the volume of dataset increases. There is a large amount of unnecessary data gathering, processing, and storing. Third, users can only use the portion of the Web of data that has been copied into the registry.

Distributed approaches are not much different from work on relational federation systems. Such approaches offer several advantages: There is no need to synchronize copied data. Queries are more dynamic with up-to-date data. New resources can be added easily without a time lag for indexing and integrating the data, and these systems require less storage. However, the potential drawback is that we cannot assume that all publishers provide reliable SPARQL endpoints for their Linked Data.

We investigate a *compromise method* between the local approach and the distributed approach. The local approach obviously offers better performance, but the queried data might not be up-to-date because Linked Data change a lot. The maintenance of local auxiliary index structures may solve this problem instead of storing data triples entirely locally. Using indexes we can retrieve distributed data resources participating on a query result, rapidly reducing the amount of data that are really needed to be accessed on-demand.

We adapt multidimensional histograms(MDH) techniques [9] for Linked Data storage and indexing. The goal of our index structure called MDH* is to support efficient join query processing without significant storage demand. In order to scale the query processor, we should design a compact storage structure and minimize the number of indexes used in the query evaluation.

The first step of building the MDH* is to transform the RDF triples into the numerical space. We apply a hash function to the RDF triples for numerical numbers. In this case, these numbers are points in the *n*-dimensional data space whose coordinates correspond to 3-dimensional cubes for (*s*, *p*, *o*), where *s* denotes the subject, *p* the predicate, and *o* the object. The coordinates are inserted one after another and aggregated into regions. Each region maintains a list of resources. Each resource in the triple table is extended with two additional *occurrences* in order to speed up join queries rather than single RDF triple. The occurrences specify *s#* and *o#*, where *s#* indicates the number of subjects in which *o* occurs as subjects in the RDF dataset and similarly *o#* indicates the number of objects in which *s* occurs as objects.

Definition 1: A pair (*t*, *v*) is an RDF tuple with count values *v*, where *t* is a triple of points (*x*, *y*, *z*) and *v* is occurrences *s#* and *o#*. Note that the RDF tuple ((*x*, *y*, *z*), (*s#*, *o#*)) is equivalent to the 5-column tuple (*x*, *y*, *z*, *s#*, *o#*)

We decided to use the equi-width histograms for the MDH*, because they can be built efficiently even if the exact distribution is not known in advance. In the histograms, each partition defines the boundaries of a region in the dimension. Since counting and storing the occurrences may be costly, adding the counts can be performed as a batch operation once the MDH has been constructed. When a query is given, the first step is to determine relevant numerical triples that answer the query. By looking up these triples in the MDH*, we obtain a set of resources potentially providing relevant data.

B. *Query Processing*

There are two types SPARQL queries: *single triple pattern* query and *join triple pattern* query. Processing queries with single triple pattern is straightforward. When a query consists of multiple triple patterns that share at least one variable, we call the join triple pattern query. In SPARQL queries, there are eight triple patterns [10]. Among them (*?s, ?p, ?o*) is required a full scan, and the number of triples matching (*s, p, o*) is 0 or 1. Hence, we need to estimate the selectivity of six triple patterns: (*s, p, ?o*), *(?s, p, o*), (*s, ?p, o*), (*s, ?p, ?o*), (*?s, ?p, o*), (*?s, p, ?o*).

Due to the page length limit, we describe here the steps for evaluating (*s, p, ?o*). We process this pattern in three steps: (1) Using the MDH*, we locate all regions that possibly provide the result. (2) We need to find the set of relevant resources in the regions. (3) We examine each relevant resource to find objects matching the given values of *s* and *p*. First of all, (*s, p, ?o*) is converted into a set of coordinates in the data space by applying the same hash function that we used for the index creation. However, in contrast to building hash values for RDF triples, triple patterns for queries might contain variables. Because of these variables, we have to work with regions instead of points. Using a query line in the space, we can determine all regions contained in the MDH* that overlap with the line. After having identified all relevant regions, we can determine the set of relevant resources.

Since the join query is expressed by conjunction of multiple triple patterns, a prerequisite is to identify *relevant resources* that possibly provide the result for a basic triple pattern. With the help of our MDH*, we can choose the data regions that contain all possible triples matching the patterns. Then, we can find sets of relevant resources. A join algorithm can be implemented by using many various techniques (e. g., merge join, hash join, nested-loop join, etc.) known from relational databases. A straightforward implementation of a region join is the nested-loop join. Considering an example of (*s, p, ?o*) $\bowtie$ (*?s, p, ?o*) pattern, if *s#* is not 0, then perform the join operation. This operation checks whether the two tuples satisfy the join condition. If the join condition is satisfied, then the values of these two tuples are added to the result and *s#* is decremented by one. This process repeats until *s#* becomes 0. Thus our algorithm can quickly prune unnecessary scanning that is guaranteed not to match the query. Similar process applies to *o#* if (*?s, p, o*) $\bowtie$ (*?s, p, ?o*) pattern is given.

C. *Application Approaches*

With large scale Linked Data being published on the Web, a number of efforts are under way to develop applications in regard to the Semantic Web. The followings are software components that can be used implement applications on top of the platform.

- *Browser:* Semantic browsers enable users to navigate between data sources along RDF links. They discover new data sources by automatically following *owl:sameAs* links and allow the user to move through a potentially endless Web of data sources connected by RDF links. Examples of such browsers include Tabulator, Marbles, CBS(Context Search Browser), Magpie, and Piggy Bank.

- *Search Engine:* Semantic search engines use robots to crawl RDF data by following data-level links from the Web of Linked Data. They provide expressive query capabilities over the complete data space, similar to those provided by conventional relational databases. Examples of such search engines include FalconS, SWSE(Semantic Web Search Engine), Sindice, Sig.ma, Swoogle, and Watson.

- *Application:* Linked Data applications that target specific topical domains include "Active Hiring" (a search based application providing analytics on on-line job posts [11]), "DBpedia Mobile" (a location-centric DBpedia client application for mobile devices [12]), and "LinkedGeoData" (an efforts to add spatial dimension to the Web of Linked Data [13]).

A key challenge now is to build applications based on the Linked Data. One of main trends in Semantic Web community is publishing big datasets to the Web in the format of Linked Data. The emergence of Linked Data has been making an excellent revolution of Semantic Web applications. For example, Mashup is a Web application that combines content from two or more services to create a new service. Although Mashup has emerged as a very popular method of integrating Web services, it is still suffer the data heterogeneity coming from various sources having different formats. To solve this problem, we adopt the Linked Data based on RDF data as a unified data model. Unlike existing Mashup applications against a fixed set of data sources, Linked Data applications can discover new data sources at runtime and deliver more complete answers as new data sources appear.

Fig. 3 shows our semantic Mashup approach that uses two *Linked Data services*[1]: a hotel search service and a mapping service. Their interfaces could be semantically similar, i.e., the first service returns a location as an output and the second service receives a location. But the vocabulary used by the two services could be very different. For example, what one service interface may encode as *countryID*, *stateName*, and *cityName* may be referred to by another service interface as *countryCode*, *areaCode*, and *nameOfCity*. Today, in IT integration projects, a substantial amount of developer time is spent in identifying these kinds of semantic ambiguities and resolving them. The semantic technology can be used to automatically match and map parameters represented as RDF data interfaces. Here, although *countryID* and *countryCode* are different forms, they have the same semantics since they are properties of the same object (i.e., *country*) (Similarly, *cityName* and *nameOfCity* can be considered as the same semantics). Also *stateName* and *areaCode* have the same semantics since they are referred to the same concept.

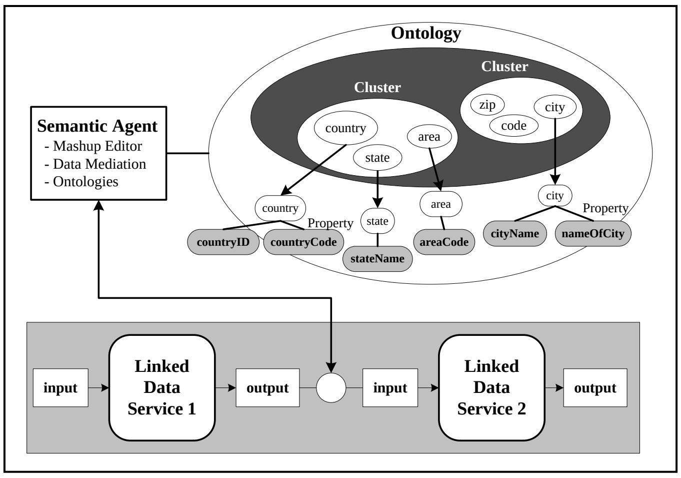

Fig. 3. Semantic Mashup Approach

[1] The W3C Linked Data Platform (LDP) defines a read/write Linked Data architecture based on the RESTful protocol. This concept is similar to the Linked Data service.

IV. CONCLUSION

The evolution of Linked Data on the Web has made a strong wave of research approaches in Semantic Web community. In this paper we proposed a Linked Data platform that is the storage, retrieval, and application platform for ultra-large-scale Linked Data. We also investigate the storage and indexing, query processing, and application approaches for the platform. Our platform is an integrated workflow-based architecture which supports the whole life cycle of Linked Data from data acquisition, building ontology via storage and indexing to retrieval and analysis. The main components of the platform are open source in order to facilitate wide usage and ease the scalability. This proposal is a first step of our research aiming at increasing tool coverage and building real Linked Data applications.

REFERENCES

[1] M. Young, The Technical Writer's Handbook. Mill Valley, CA: University Science, 1989. S. Auer, J. Lehmann, A. N. Ngomo, and A. Zaveri, "Introduction to Linked Data and its lifecycle on the Web," in Proc. 9th Int. Summer School 2013, Mannheim, Germany, pp. 1-90, 2013.
[2] T. Berners-Lee, "Linked Data-Design Issues," 2006, http://www.w3.org/ DesignIssues/LinkedData.html.
[3] SWEO Community Project. (January 2007). Linking open data [Online]. Available: http://www.w3.org/wiki/SweoIG/TaskForces/Community-Projects/LinkingOpenData
[4] M. Svoboda, "Efficient querying of distributed Linked Data," in Proc. 2011 Joint EDBT/ICDT PhD Workshop, 2011, pp. 45-50.
[5] Scrapy: An open source web scraping framework for pthon, http://scrapy.org/
[6] http://d2rq.org/d2r-server.
[7] http://protege.stanford.edu.
[8] O. Hartig, "An overview on execution strategies for Linked Data queries," Datenbank Spektrum, vol. 13, no. 2, pp. 89-99, 2013.

[9] A. Harth, K. Hose, M. Karnstedt, A. Polleres, K. U. Satler, and J. Umbrich, “Data summaries for on-demand queries over Linked Data,” in Proc. 19th Int. Conf. on World Wide Web (WWW), pp. 411-420, 2010.

[10] A. Harth and S. Decker, “Optimized index structures for querying RDF from the Web,” in Proc. 3rd Latin American Web Congress (LA-Web), pp. 71-81, 2005.

[11] A. D. Mezaour, J. Law-To, R. Isele, T. Schandl, and G. Zechmeister, “Revealing Trends and Insights in Online Hiring Market Using Linking Open Data Cloud: Active Hiring a Use Case Study,” Semantic Web Challenge, 2012.

[12] C. Becker and C. Bizer, DBpedia Mobile: A Location-Enabled Linked Data Browser, 1st Workshop about Linked Data on the Web (LDOW 2008), Beijing, China, April 2008.

[13] C. Stadler, J. Lehmann, K. Hoffner, and S. Auer, LinkedGeoData: A Core for a Web of Spatial Open Data, Semantic Web Journal, Vol. 3, No. 4, pp. 333-354, 2012.