engineering proceedings

MDPI

Proceeding Paper

Tool for SPARQL Querying over Compact RDF Representations †

Delfina Ramos-Vidal [1,2,*] and Guillermo de Bernardo [1,*]

[1] Centro de Investigación CITIC, Universidade da Coruña, 15071 A Coruña, Spain
[2] Departmento de Inteligencia Artificial, Universidad Politécnica de Madrid, 28040 Madrid, Spain
* Correspondence: delfina.ramos@udc.es (D.R.-V.); gdebernardo@udc.es (G.d.B.)
† Presented at the 4th XoveTIC Conference, A Coruña, Spain, 7–8 October 2021.

Abstract: We present an architecture for the efficient storing and querying of large RDF datasets. Our approach seeks to store RDF datasets in very little space while offering complete SPARQL functionality. To achieve this, our proposal was built over HDT, an RDF serialization framework, and its interaction with the Jena query engine. We propose a set of modifications to this framework in order to incorporate a range of space-efficient compact data structures for data storage and access, while using high-level capabilities to answer more complicated SPARQL queries. As a result, our approach provides a standard mechanism for using low-level data structures in complicated query situations requiring SPARQL searches, which are typically not supported by current solutions.

Keywords: RDF; SPARQL; compact data structures

Citation: Ramos-Vidal, D.; de Bernardo, G. Tool for SPARQL Querying over Compact RDF Representations. *Eng. Proc.* **2021**, *7*, 33. https://doi.org/10.3390/engproc2021007033

Academic Editors: Joaquim de Moura, Marco A. González, Javier Pereira and Manuel G. Penedo

Published: 15 October 2021

1. Introduction

Massive volumes of data must be efficiently gathered and processed in the era of the Internet in which we are constantly generating more information. In recent years, significant effort has been devoted to develop mechanisms to share data in standardized, machine-readable formats, leading to the so-called Web of Data. The Resource Description Framework (RDF) defines a common framework to describe resources using URIs to name associations between items. An RDF graph can be conceptually represented as a labeled graph or as a set of *triples*. The W3C also promotes SPARQL, a querying standard for RDF, which utilizes an SQL-like language. SPARQL specifies queries via graph pattern matching, imposing limitations on the resultant RDF subgraphs. At the core of SPARQL are triple patterns, which define basic matching with the collection of triples in the RDF graph.

The RDF standard only specifies a conceptual representation of data as a graph, not a specific data storage format. As a result, RDF may be stored in a variety of formats, and a slew of RDF storage solutions, or *RDF* stores, have surfaced in recent years. Several full-featured RDF query engines, such as Virtuoso and Jena, provide full SPARQL support and can handle large RDF datasets [1]. However, in recent years, a number of solutions based on compact data structures have arisen, with the goal of reducing the storage space while still providing efficient querying. Even though database-like solutions function well and fully support SPARQL, compact data structures have been proven to outperform the former in several areas, particularly for simpler query operations such as basic triple-pattern searches.

Compact data structures designed to store RDF usually divide the problem into two parts. On the one hand, there is a Triples component, which stores each RDF triple (s, p, o) assuming that its components are integer identifiers (IDs). On the other hand, a dictionary component holds the mapping from the original strings of the RDF dataset to IDs, allowing the conversion from string to ID and versa. This leads to extremely specialized string dictionaries, such as libCSD [2] and specialized methods for storing RDF triples encoded as integer IDs, such as K2-triples [3] or RDFCSA [4]. These solutions have been shown to be able to store RDF datasets in small spaces and perform very efficiently to answer

basic triple-pattern and other simple queries. The HDT library [5] provides a standard framework for the common representational notion of dictionary encoding, which divides an RDF dataset into three components (header, dictionary, and triples), and provides default implementations for the dictionary and triples. The main issue with most of these solutions based on compact data structures is that they do not completely support SPARQL; instead, they are designed to efficiently serve a limited number of queries using very specialized methods. For example, some solutions may only allow triple-pattern queries or basic join patterns.

In this paper, we present an architecture for the representation of RDF datasets that aim to provide a common framework for the representation of RDF based on compact data structures. Our approach seeks to store RDF data in little space while supporting sophisticated SPARQL searches. Our proposal has immediate applications as a common testing framework for compact data structures that have never been tested in complex SPARQL query scenarios.

2. Our Proposal

We propose an architecture that combines fully fledged RDF engines' high-level capabilities with the compression performance of low-level compact data structures. We developed a prototype tool as an extension of the HDT Java library, a solution that proposes a relatively compact representation of RDF and provides SPARQL support through an integration with Apache Jena and Jena ARQ. We take advantage of this SPARQL support and aim to avoid the main drawback of this library: the data structures used for the dictionary and triples, while reasonably efficient, are much larger than other alternatives such as K2-triples. Our tool extends this framework to allow the utilization of different underlying representations for the storage of the RDF dictionary and triples components. This way, our tool provides a simple mechanism to incorporate and test different data structures using a common framework involving SPARQL queries. Particularly, our solution provides a simple mechanism to integrate existing solutions implemented in C/C++, using Java Native Interface (JNI) to transparently incorporate these structures into our extended HDT Java framework. Currently, we have a functional prototype that integrates the K2Triples and libCSD libraries, state-of-the-art representations able to efficiently compress RDF triples and RDF dictionaries, respectively, the two main components used in the HDT framework to store RDF data. A conceptual overview of the suggested framework can be seen in Figure 1.

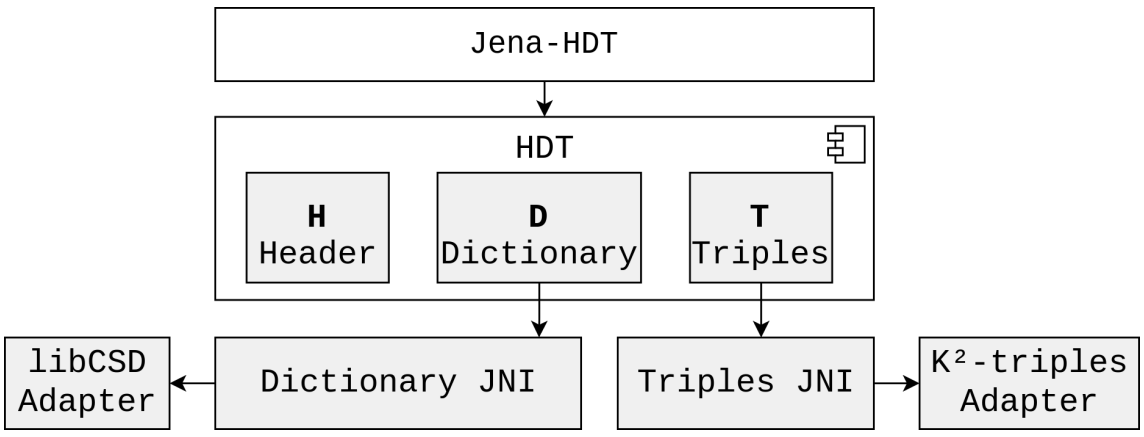

Figure 1. Architecture diagram.

The main advantage of this proposal is the possibility to integrate fully functional alternatives, commonly implemented in C or C++, alternatives that are already known for efficiently solving basic problems. These techniques are not usually applied to solve bigger problems such as SPARQL, due to the additional cost of specifically redesigning all of the SPARQL queries processing mechanisms for each case. The proposed framework can take

care of the high-level query processing, so that future integrations only need to build a specific adapter to query the desired data structure from the JNI extensions defined in our proposal, either for the dictionary or the triples component.

The proposed solution was tested with a dataset generated by the Berlin SPARQL Benchmark, containing 35 million triples. We compressed the data using the different components provided by our framework—first by using the original components given by the HDT library; secondly, by only using the component for the JNI triples and only the component for the JNI dictionary. Lastly, we compressed the dataset combining both JNI triples and dictionary components. A small sample of the results for this experiment can be seen in Table 1. The queries listed below include filters, unbound predicates, unions, result modifiers such as LIMIT, ORDER BY and DISTINCT, and the DESCRIBE operator.

Table 1. Comparison of query execution times for different compression techniques.

	HDT Components	JNITriples	JNIDictionary	JNI Components
Q1 (ms)	331	1573	349	2012
Q2 (ms)	239	202	252	270
Q3 (ms)	363	4738	356	4501
Q4 (ms)	238	214	281	265

Our JNIDictionary implementation is based on plain front coding (PFC), a technique for compressed string dictionary that is also used by the original HDT library. It can be seen that the JNIDictionary component achieves similar results to the original HDT components. Considering that both solutions use similar implementations, but JNIDictionary has the additional cost of translating from Java to C++, the competitive results suggest that the dictionary overhead can be reduced in this scenario. On the other hand, JNITriples is slower, since K^2-triples is much more compact than the solution posed by HDT but is also expected to be slower in many queries. Therefore, JNITriples is competitive in query times for the simpler queries, but significantly slower overall.

3. Conclusions and Future Work

Our proposal is currently a fully functional prototype, but we intend to improve it further in order to provide a competitive and flexible framework in the future. Upgrades in the scalability of the underlying compact data structures at construction time, as well as performance overheads owing to the usage of JNI, are currently possible. We also want to test a variety of state-of-the-art compact representations that should be simple to integrate into the existing framework and might lead to specific enhancements for various SPARQL query operations for specific implementations. As a result, our approach provides a standard framework for evaluating the performance of low-level data structures for RDF representation, as well as potential improvements.

Funding: This research was funded by Xunta de Galicia/FEDER grant ED431G 2019/01, Xunta de Galicia/FEDER-UE grant IN852A 2018/14; Ministerio de Ciencia, Innovación y Universidades grants [TIN2016-78011-C4-1-R; PID2019-105221RB-C41]; Consellería de Cultura, Educación e Universidade/Consellería de Economía, Empresa e Innovación/GAIN/Xunta de Galicia grant ED431C 2021/53; and by MICINN (PGE/ERDF) grant PID2020-114635RB-I00.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Berlin SPARQL Benchmark http://wifo5-03.informatik.uni-mannheim.de/bizer/berlinsparqlbenchmark/.

References

1. Nitta, K.; Savnik, I. Survey of RDF storage managers. In Proceedings of the Advances in Databases, Knowledge, and Data Applications (DBKDA), Chamonix, France, 20 April 2014; pp. 148–153.

2. Martínez-Prieto, M.A.; Brisaboa, N.R.; Cánovas, R.; Claude, F.; Navarro, G. Practical compressed string dictionaries. *Inf. Syst.* **2016**, *56*, 73–108. [CrossRef]
3. Álvarez-García, S.; Brisaboa, N.R.; Fernández, J.D.; Martínez-Prieto, M.A.; Navarro, G. Compressed vertical partitioning for efficient RDF management. *Knowl. Inf. Syst.* **2015**, *44*, 439–474. [CrossRef]
4. Cerdeira-Pena, A.; Fariña, A.; Fernández, J.; Martínez-Prieto, M.A. Self-indexing RDF archives. In Proceedings of the Data Compression Conference (DCC), Snowbird, UT, USA, 30 March–1 April 2016; pp. 526–535.
5. Fernández, J.D.; Martínez-Prieto, M.A.; Gutierrez, C.; Polleres, A.; Arias, M. Binary RDF representation for publication and exchange (HDT). *J. Web Semant.* **2013**, *19*, 22–41. [CrossRef]