Heinrich C. Mayr, Stefanie Rinderle-Ma, Stefan Strecker (Hrsg.): 40 Years EMISA,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2019 135

Knowledge Graph Processing Made (more) Simple

Georg Lausen[1]

Abstract: Knowledge graphs based on RDF and SPARQL are gaining popularity for integrated semantic representation of structured and unstructured data. As knowledge graphs in practical applications tend to become huge, distributed processing using Apache Spark SQL and Hadoop on top of a compute cluster is attractive. For the corresponding relational representation of a knowledge graph a simple relational design using only one single table is proposed. As a consequence no time consuming relational design considerations are required and newly discovered RDF data can be integrated with nearly no extra additional relational design effort.

Keywords: Knowledge Graph, RDF, SPARQL, Apache Spark SQL, Parquet, Hadoop

1 Introduction

Knowledge graphs [DG14, IBM] are gaining popularity for integrated semantic representation of structured and unstructured data. For example, a knowledge graph may result from enriching local structured corporate data by various unstructured data from the Linked Open Data Cloud. The Resource Description Framework (RDF) and its associated processing language SPARQL are widely used in this context. As knowledge graphs in practical applications tend to become huge, processing becomes challenging which has triggered the development of a large number of different approaches [WH18].

In this extended abstract we elaborate on SQL-based SPARQL processing using Apache Spark SQL and Hadoop as distributed processing platform. While our earlier work using Hadoop as platform suffered from the inflexibility of the rigid implementation of the MapReduce paradigm [SP13], Spark SQL allows mapping SPARQL queries into SQL directly and processing them akin to a distributed main-memory database system [SP16,CF18]. Based on the obtained experiences we now give arguments supporting the claim that a mostly simple underlying relational design using only one single wide table is competitive to other designs typically based on several tables. This approach allows efficient query processing and does not require additional relational design considerations. Moreover it is able to account to any kind of newly discovered RDF data with nearly no extra design effort [AK19].

[1] Albert-Ludwigs-Universität Freiburg, Technische Fakultät, Institut für Informatik, 79110 Freiburg i.Br.
lausen@informatik.uni-freiburg.de

2 Single Wide Property Table Approach

Using RDF, a knowledge graph is represented by a set R of (s, p, o)-triples, where s denotes a subject, p a property (predicate) and o an object. In the literature various approaches have been discussed for representation of such sets [WH18]: a single triple table, a set of tables resulting from partitioning the triples by properties, a single wide table covering all properties, and a set of tables akin to tables defined by an emergent schema, are the considered designs. For the wide property table approach WPT, each subject s identifies a single row in the table and each property p is assigned to a column. Objects o are the values inside the table, where each such value may be either a constant or an IRI identifying the subject s' being in a relationship with s as stated by property p. The resulting WPT table in general is very sparse and in addition must be able to handle properties which assign multiple values to the respective subject. These crucial aspects can be solved by using Hive, Spark and Parquet for data representation. These technologies offer an efficient representation of sparse tables, and are also able to store multiple values in columns by their support for complex data types, such as arrays or maps.

For example, consider a knowledge graph which represents a friendOf-relationship between persons and may represent the age for persons. The following SPARQL query determines those persons which are stated to be a friend of a 65 year old person:

```
SELECT DISTINCT ?usr, ?fr WHERE { ?usr friendOf ?fr . ?fr age "65" }
```

Let KG be the wide property table representing the knowledge graph. The schema of KG can be stated by KG(s, …, age, …, friendOf, …) as we are only interested in the columns s for the subjects, friendOf and age. As friendOf in general is multivalued, the respective Spark SQL expression being equivalent to the above SPARQL query has the following structure:

```
SELECT usr, fr  FROM
  ( SELECT s as usr, xfo as fr  FROM KG lateral view explode (friendOf) fo as xfo ) F
  JOIN
  ( SELECT s as usr FROM KG WHERE age IS NOT NULL AND age='65') A
  ON F.fr = A.usr
```

Note that by an explode operation the potential multivalued property friendOF is replaced by a table such that an implicit lateral join can applied.

For a more realistic context, we investigated the Yago (version 2s 2.5.3) knowledge graph, which is derived from Wikipedia, WordNet and GeoNames with approximately 220 Million triples and 104 properties. The resulting WPT is of storage size 3.61GB and extremely sparse: the ratio between number of null values and number of elements in the table is 0.975.

3 Distributed Implementation

We use Apache Hadoop as distributed processing platform. Our cluster runs Cloudera CDH 5.10.0 with Spark 2.2 on Ubuntu 14 and consists of one master and nine workers giving in total 198GB to be used and 108 virtual cores. To validate our expectations about WPT we analyzed the performance of Yago queries executed on top of the different table designs, i.e. single triple table, partitioning by properties, WPT, and a set of tables stemming from an emergent schema. As we detail in [AK19] the single wide property table approach has competitive efficiency. As reasons for this surprising behavior we see: (1) star joins can be solved by selections, as all properties for a certain subject reside in the same row; (2) the number of joins required is minimal with respect to the other approaches; (3) and, because of its size, a single wide property table implies a physical partitioning which provides work for as many nodes in the cluster as possible. Moreover, WPT does not require any additional design overhead and is able to account to any kind of newly discovered RDF data with nearly no extra design effort.

Literature

[AK19] Victor Anthony Arrascue Ayala, Polina Koleva, Anas Alzogbi, Matteo Cossu, Michael Färber, Patrick Philipp, Guilherme Schievelbein, Io Taxidou, Georg Lausen. Relational Schemata for Distributed SPARQL Query Processing. CoRR, 2019

[AL18] Victor Anthony Arrascue Ayala, Georg Lausen. A Flexible N-Triples Loader for Hadoop. ISCW (Posters & Demos), 2018

[CF18] Matteo Cossu, Michael Färber, Georg Lausen. PRoST: Distributed Execution of SPARQL Queries Using Mixed Partitioning Strategies. EDBT 2018

[DG14] Xin Luna Dong, Evgeniy Gabrilovich, Geremy Heitz, Wilko Horn, Ni Lao, Kevin Murphy, Thomas Strohmann, Shaohua Sun, Wei Zhang. Knowledge Vault: A Web-Scale Approach to Probabilistic Knowledge Fusion. KDD 2014

[IBM] https://console.bluemix.net/docs/services/discovery/building-kg.html#watson-discovery-knowledge-graph (March 2019)

[SP13] Alexander Schätzle, Martin Przyjaciel-Zablocki, Thomas Hornung, Georg Lausen. PigSPARQL: A SPARQL Query Processing Baseline for Big Data. ISCW (Posters & Demos) 2013

[SP16] Alexander Schätzle, Martin Przyjaciel-Zablocki, Simon Skilevic, Georg Lausen. S2RDF: RDF Querying with SPARQL on Spark. PVLDB 9(10), 2016

[WH18] Marcin Wylot, Manfred Hauswirth, Philippe Cudré-Mauroux, Sherif Sakr. RDF Data Storage and Query Processing Schemes: A Survey, ACM Comput. Surveys. 2018