Sparklify
A Scalable Software Component for Efficient Evaluation of SPARQL Queries over Distributed RDF Datasets
Claus Stadler, Gezim Sejdiu, Damien Graux, and Jens Lehmann
Presented by Claus Stadler
Overview

● Motivation
● Introduction
  ○ Apache Spark
  ○ Sansa
  ○ Sparqlify
● Approach
  ○ Data Ingestion
  ○ RDF-term aware Vertical Partitioning
  ○ Mapping: RDF views over DataFrames
  ○ Query Evaluation
● Evaluation
  ○ Lehigh University Benchmark (LUBM)
  ○ Waterloo SPARQL Diversity Test Suite (WatDiv)
Motivation

SELECT * { ?s ?p ?o }
Motivation

- Dictionary Encoding
- Type-based semantic optimization
- Materialized Views
- Semantic Query Optimization
- Wide Property Tables
- Column Stores
- Semantic Partitioning
- Source Selection
Motivation

SELECT * { ?s ?p ?o }

* Ontology-Based Data Access
Motivation

- Can we re-use existing OBDA tooling to facilitate running SPARQL queries on RDF kept in Apache Spark?

- Performance?
- Scalability?
Overview

● Motivation

● Introduction
  ○ Apache Spark
  ○ Sansa
  ○ Sparqlify

● Approach
  ○ Data Ingestion
  ○ RDF-term aware Vertical Partitioning
  ○ Mapping: RDF views over DataFrames
  ○ Query Evaluation

● Evaluation
  ○ Lehigh University Benchmark (LUBM)
  ○ Waterloo SPARQL Diversity Test Suite (WatDiv)
Apache Spark

- Fast and generic-purpose cluster computing engine which built over Hadoop
- Allows for massive parallel processing of collections of records
  - RDD - Resilient Distributed Dataset; a collection of serializable objects
  - DataFrame - Conceptually a table
  - Dataset - “Best of both worlds”; Unified access to data as objects and/or tables; extra optimizations based on byte code analysis of which attributes are actually accessed
Sparqlify

● **SPARQL-to-SQL rewriter**
  ○ Supports mappings expressed in W3C R2RML & our SML Syntax*
  ○ Developed for the LinkedGeoData project (LGD), which serves OpenStreetMap as RDF
  ○ SML constraints
  ○ Largest LGD dump created: ~30 billion triples in a single SPARQL query; ran for 3-4 weeks in 2013

● **SparKlify = Sparqlify on Apache Spark**

* Simplified RDB2RDF Mapping, C. Stadler et al., LDOW 2015
Sansa-Stack

- Our Open Source Big Data + Semantic Web Framework
  - Layers for RDF, Querying, OWL, Inferencing and Machine Learning + Examples
  - Spark + Flink support (More features in Spark)
  - 6 releases, 6 months release cycle
  - 13 contributors according to github
  - Several presentations at ISWC2019

- Includes Sparklify as a query engine
Overview

● Motivation
● Introduction
  ○ Apache Spark
  ○ Sansa
  ○ Sparqlify
● Approach
  ○ Data Ingestion
  ○ RDF-term aware Vertical Partitioning
  ○ Mapping: RDF views over DataFrames
  ○ Query Evaluation
● Evaluation
  ○ Lehigh University Benchmark (LUBM)
  ○ Waterloo SPARQL Diversity Test Suite (WatDiv)
Sparklify - System Overview

1. RDF Data
2. Data Ingestion
3. Sparklifying Query Layer
4. SPARQL query
5. SPARQL Algebra Expression Tree (AET)
6. Normalize AET
7. Distributed Data Structures
8. Results

Prefix dbp:<http://dbpedia.org/ontology/>
Prefix ex:<http://ex.org/>

CREATE VIEW view_person AS

CONSTRUCT {
  ?s a dbp:Person .
  ?s ex:workPage ?w .
}

WITH
  ?s = uri('http://mydomain.org/person', ?id)
  ?w = uri('?work_page')

CONSTRAN
  ?w prefix "http://my-organization.org/user/"

FROM
  person;

SELECT id, work_page
FROM view_person;
RDF-Term aware Vertical Partitioning

:x a qb:Observation

:x qb:value 1.000

:x rdfs:label "Measurement"@en

Spark RDD of triples
### RDF-Term aware Vertical Partitioning

**Available RDF Term Types**
- rr:IRI
- rr:BlankNode
- rdf:langString
- xsd:string
- xsd:*

**Partition Metadata**
- subjectType=rr:IRI
- predicate=rdf:type
- objectType=rr:IRI

**Spark DataFrame**

<table>
<thead>
<tr>
<th>s</th>
<th>o</th>
</tr>
</thead>
<tbody>
<tr>
<td>http://...</td>
<td>http://...</td>
</tr>
</tbody>
</table>

**Subject**

- :x a qb:Observation
- :x qb:value 1.000
- :x rdfs:label “Measurement”@en
RDF-Term aware Vertical Partitioning

Available RDF Term Types
rr:IRI, rr:BlankNode, rdf:langString, xsd:string, xsd:*

Partition Metadata
subjectType=rr:IRI
predicate=rdf:type
objectType=rr:IRI

Spark DataFrame
s  o
http://...  http://...

Partition Metadata
subjectType=rr:IRI
predicate=qb:value
objectType=xsd:decimal

Spark DataFrame
s  o
http://...  1.000

:x a qb:Observation
:x qb:value 1.000
:x rdfs:label “Measurement”@en
RDF-Term aware Vertical Partitioning

Available RDF Term Types
- rr:IRI
- rr:BlankNode
- rdf:langString
- xsd:string
- xsd:*
CREATE View rdf_type_iri_iri {
    ?s rdf:type ?o
} WITH
    ?s = IRI(?s)
    ?o = IRI(?o)
FROM
    dataframe_rdf_type_iri_iri
CREATE VIEW rdf_type_iri_iri {
    ?s rdf:type ?o
} WITH
    ?s = IRI(?s)
    ?o = IRI(?o)
FROM
    dataframe_rdf_type_iri_iri

CREATE VIEW qb_value_long {
    ?s rdfs:label ?o
} WITH
    ?s = IRI(?s)
    ?o = STRLANG(?o_v, ?o_l)
FROM
    dataframe_qb_value_long
RDF Views over Tables

CREATE VIEW rdf_type_iri_iri {
    ?s rdf:type ?o
} WITH
    ?s = IRI(?s)
    ?o = IRI(?o)
FROM
dataframe_rdf_type_iri_iri

CREATE VIEW rdfs_label_iri_lang {
    ?s rdfs:label ?o
} WITH
    ?s = IRI(?s)
    ?o = STRLANG(?o_v, ?o_l)
FROM
dataframe_label_type_iri_lang

CREATE VIEW qb_value_long {
    ?s qb:value ?o
} WITH
    ?s = IRI(?s)
    ?o = STRDT(STR(?o), xsd:decimal)
FROM
dataframe_qb_value_long
Querying

SPARQL Query
Select * { ?s ?p ?o }

OBDA System - Sparqlify
- Translates SPARQL to SQL w.r.t. the mappings
- Translates SQL result sets to SPARQL result sets w.r.t. the mappings

SQL-Accessible data
A set of tables / Spark dataframes

Mappings / Views
Create View v1 ...
....
Create View vn ...

SPARQL Results
Overview

● Motivation

● Introduction
  ○ Apache Spark
  ○ Sansa
  ○ Sparqlify

● Approach
  ○ Data Ingestion
  ○ RDF-term aware Vertical Partitioning
  ○ Mapping: RDF views over DataFrames
  ○ Query Evaluation

● Evaluation
  ○ Lehigh University Benchmark (LUBM)
  ○ Waterloo SPARQL Diversity Test Suite (WatDiv)
Evaluation

● Research questions
  ○ How does our approach scale to datasets of different sizes?
  ○ What is the effect on the runtime when more worker nodes are added in the cluster?
  ○ What is the effect on the runtime for different types of queries?

● Lehigh University Benchmark (LUBM), generates synthetic data using the University ontology (Students, Chairs, Departments, Courses, ...)
  ○ 1.000, 5.000, 10.000 universities → 138M - 1.4B triples

● WatDiv - Waterloo SPARQL Diversity Test Suite - different query types for E-Commerce domain (Products, Purchases, Countries, ...)

22
Evaluation

● Cluster Setup
  ○ 7 nodes - 1 master, 6 worker, each with Intel(R) Xeon(R) CPU E5-2620 v4 @ 2.10GHz (32 Cores), 128 GB RAM, 12 TB SATA RAID-5, connected via a Gigabit network
  ○ Each experiment executed 3 times, avg’ed results

● Sparklify does not perform data analysis / summarization / preprocessing (besides vertical partitioning)

● Comparison with SPARQLGX-SDE which translates SPARQL queries to Spark API calls (not directly SQL)

● Therefore comparison between declarative vs imperative approach
## Small dataset size

10,000,000 triples: SPARQLGX-SDE wins

<table>
<thead>
<tr>
<th>Query Types</th>
<th>SPARQLGX-SDE</th>
<th>Sparklify</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Runtime (s) (mean)</td>
<td></td>
</tr>
<tr>
<td>a) total</td>
<td>b) partitioning</td>
<td>c) querying</td>
</tr>
<tr>
<td>QC</td>
<td>103.24</td>
<td>134.81</td>
</tr>
<tr>
<td>QF</td>
<td>157.8</td>
<td>241.24</td>
</tr>
<tr>
<td>QL</td>
<td>102.51</td>
<td>236.06</td>
</tr>
<tr>
<td>QS</td>
<td>131.16</td>
<td>237.12</td>
</tr>
</tbody>
</table>

**Query Types:**
- QS: Star pattern
- QL: Linear pattern
- QF: Snowflake
- QC: Complex pattern
## Large Dataset (WatDiv)

### Query Types:
- QS: Star pattern
- QL: Linear pattern
- QF: Snowflake
- QC: Complex pattern

<table>
<thead>
<tr>
<th>WatDiv-1EB</th>
<th>SPARQLGx-SDE</th>
<th>Sparklify</th>
</tr>
</thead>
<tbody>
<tr>
<td>QC</td>
<td>partial fail</td>
<td>778.62</td>
</tr>
<tr>
<td>QF</td>
<td>6734.68</td>
<td>1295.31</td>
</tr>
<tr>
<td>QL</td>
<td>2575.72</td>
<td>1275.22</td>
</tr>
<tr>
<td>QS</td>
<td>4841.85</td>
<td>1290.72</td>
</tr>
</tbody>
</table>

1.000.000.000 triples on WatDiv: Sparklify wins
## Large Dataset - LUBM

<table>
<thead>
<tr>
<th></th>
<th>SPARQLGX-SDE</th>
<th></th>
<th>Sparklify</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>a) total</td>
<td>b) partitioning</td>
<td>c) querying</td>
<td>d) total</td>
</tr>
<tr>
<td>Q1</td>
<td>1056.83</td>
<td>627.72</td>
<td>718.11</td>
<td>1346.8</td>
</tr>
<tr>
<td>Q2</td>
<td>fail</td>
<td>595.76</td>
<td>fail</td>
<td>n/a</td>
</tr>
<tr>
<td>Q3</td>
<td>1038.62</td>
<td>615.95</td>
<td>648.63</td>
<td>1267.37</td>
</tr>
<tr>
<td>Q4</td>
<td>2761.11</td>
<td>632.93</td>
<td>1670.18</td>
<td>2303.18</td>
</tr>
<tr>
<td>Q5</td>
<td>1026.94</td>
<td>641.53</td>
<td>564.13</td>
<td>1206.67</td>
</tr>
<tr>
<td>Q6</td>
<td>537.65</td>
<td>695.74</td>
<td>267.48</td>
<td>963.62</td>
</tr>
<tr>
<td>Q7</td>
<td>2080.67</td>
<td>630.44</td>
<td>1331.13</td>
<td>1967.25</td>
</tr>
<tr>
<td>Q8</td>
<td>2636.12</td>
<td>639.93</td>
<td>1647.57</td>
<td>2288.48</td>
</tr>
<tr>
<td>Q9</td>
<td>3124.52</td>
<td>583.86</td>
<td>2126.03</td>
<td>2711.24</td>
</tr>
<tr>
<td>Q10</td>
<td>1002.56</td>
<td>593.68</td>
<td>693.73</td>
<td>1287.71</td>
</tr>
<tr>
<td>Q11</td>
<td>1023.32</td>
<td>594.41</td>
<td>522.24</td>
<td>1118.58</td>
</tr>
<tr>
<td>Q12</td>
<td>2027.59</td>
<td>576.31</td>
<td>1088.25</td>
<td>1665.87</td>
</tr>
<tr>
<td>Q13</td>
<td>1007.39</td>
<td>626.57</td>
<td>6.66</td>
<td>633.26</td>
</tr>
<tr>
<td>Q14</td>
<td>526.15</td>
<td>633.39</td>
<td>258.32</td>
<td>891.89</td>
</tr>
</tbody>
</table>

1.380.000.000 triples on LUBM: Mixed results for Sparklify 6 wins, 7 losses, 1 fail
Overall runtimes across all WatDiv datasets
Sparklify vs SPARQLGX-SDE with different number of workers on WatDiv 100M

![Graph showing runtime comparison between Sparklify and SAPRQLGX-SDE with varying number of worker nodes. The y-axis represents runtime in seconds, and the x-axis represents the number of worker nodes. The graph shows that Sparklify has lower runtime compared to SAPRQLGX-SDE across all tested worker node counts.]
Sparklify vs SPARQLGX-SDE
Per-query type performance on WatDiv 100M

Query Types:
QS: Star pattern
QL: Linear pattern
QF: Snowflake
QC: Complex pattern
Use Cases

- **Etherium Blockchain analysis** – In collaboration with Alethio analyzing the Hubs & Authorities in an RDF version of the Ethereum Transaction Network (18B triples)
- **SPECIAL** – A Semantic Transparency and Compliance Use Case: Analyze log information concerning personal data processing for presentation in a dashboard – [https://www.specialprivacy.eu/](https://www.specialprivacy.eu/)
- **SLIPO** – Categorizing Areas of Interests (AOI) Use Case: Sparklify used to refine, filter and select relevant POIs for data processing pipelines – [http://slipo.eu/](http://slipo.eu/)
Conclusions & Future Work

● Application of OBDA tooling on Big Data frameworks
  ○ Significantly reduced amount of work needed to implement a SPARQL endpoint, by means of leveraging both Spark SQL as well as existing OBDA components
    ■ E.g. leverage “SQL datatype to RDF term type” mapping and source selection
  ○ Yields promising results w.r.t. scalability
    ■ Spark SQL queries enables leveraging Spark optimizer for JOIN ordering
    ■ Works better on larger datasets
  ○ Greatest limitation in practice: lack of aggregation functions

● Future Work
  ○ Comparison with Ontop
  ○ Investigate dictionary encoding: Instead of storing IRIs, blank node labels and literals directly in the partitions, store integer references to them
Thank you!

Q & A

https://github.com/SANSA-Stack

https://github.com/SmartDataAnalytics/Sparqlify

Contact: cstadler@informatik.uni-leipzig.de