Towards In-Memory RDFS Entailment

Javier D. Fernández, Claudio Gutiérrez, Miguel A. Martínez-Prieto, Jorge Pérez

Department of Computer Science, Univ. de Chile (Chile)
Department of Computer Science, Univ. de Valladolid (Spain)
Ontology Engineering Group (OEG), Univ. Politécnica de Madrid (Spain)
Index

- **Motivation**
  - Big Semantic Data
  - RDFS Entailment
  - Current Solutions
  - Our Objective
- **RDF/HDT**
- Current Proposal: RDFS-HDT
- Conclusions
Motivation. Big Semantic Data

- Massive publication efforts have enriched the Web with huge amounts of semantic data represented in RDF
- Considering the whole Web of Data as a network of networks
- Also within each dataset

Linked Data cloud: > 62 billion triples

431 M. triples ~ 63 GB
Motivation. Big Semantic Data

Moreover, “BIG” is relative

Big Semantic Data: “Semantic data whose volume, velocity and variety exceed the computational resources available for its efficient management in a given system.”

Premise: Traditional RDF stores and SPARQL engines already suffer from scalability problems managing Big Semantic Data.

What about inference? In particular RDFS Entailment
Motivation. RDFS Entailment

- **RDF Schema (RDFS):** Vocabulary with predefined semantics to describe relationships such as typing of entities and hierarchy relations in classes and properties.

![Diagram showing relationships between classes and properties in RDF Schema](image)
Motivation. RDFS Entailment

- **RDFS Entailment**: Infer new facts, not originally explicit in the RDF graph.
Motivation. RDFS Entailment

- **RDFS Entailment (formal):** given an RDF graph $G$, and a new fact $t$ (not originally explicit in $G$)
  - If $t$ can be inferred from $G$ and the rules of the RDF Schema (RDFS) standard, then we say that $G$ entails $t$, or that $t$ is in the entailment of $G$.

- **RDFS inference rules**

  1. **Sub-property:**
     \[
     \frac{(A, \text{sp}, B) \cdot (B, \text{sp}, C)}{(A, \text{sp}, C)} \\
     (a) \\
     \frac{(A, \text{sp}, B) \cdot (\mathcal{X}, A, \mathcal{Y})}{(\mathcal{X}, B, \mathcal{Y})} \\
     (b)
     \]

  2. **Subclass:**
     \[
     \frac{(A, \text{sc}, B) \cdot (B, \text{sc}, C)}{(A, \text{sc}, C)} \\
     (a) \\
     \frac{(A, \text{sc}, B) \cdot (\mathcal{X}, \text{type}, A)}{(\mathcal{X}, \text{type}, B)} \\
     (b)
     \]

  3. **Typing:**
     \[
     \frac{(A, \text{dom}, B) \cdot (\mathcal{X}, A, \mathcal{Y})}{(\mathcal{X}, \text{type}, B)} \\
     (a) \\
     \frac{(A, \text{range}, B) \cdot (\mathcal{X}, A, \mathcal{Y})}{(\mathcal{Y}, \text{type}, B)} \\
     (b)
     \]

- **The question:** Given an RDF Graph $G$, and a new triple $t$,
  \[G \vDash t?\]
Motivation. Current Solutions ($G \models t$ ?)

(a) Precalculate the closure
- The union of $G$ and all triples inferred from $G$ and RDFS rules
- Quadratic size, $O(|G|^2)$, in the worst case
  - A real problem in Big Semantic Data

(b) Compute $G \models t$ on demand
- Theoretically efficient in time, $O(n \log n)$ [Muñoz et al, 2009]
- Big Semantic Data are generally too large to fit in main memory → use of external memory and/or distributed environments (e.g. Map/Reduce)
- Unaffordable I/O costs make this theoretically bound far from the truth.

Motivation. Our objective

- Compute $G \models t$ on demand
  - Based on [Muñoz et al, 2009]
  - Using **compressed structures** for RDF management:
    - Optimize the memory footprint
    - Perform queries without prior decompression
    - Perform in memory minimizing I/O costs

- Our current work makes use of RDF/HDT
  - Binary format and indexes known to assure a reduced memory footprint while providing fast triple pattern resolution in main memory.
Index

- Motivation
  - Big Semantic Data
  - RDFS Entailment
  - Current Solutions
  - Our Objective
- RDF/HDT
- Current Proposal: RDFS-HDT
- Conclusions
- A lightweight Binary RDF Serialization.
- + **Compact** Indexes for querying Big Semantic Data in memory.
- W3C Submission. http://www.w3.org/Submission/2011/03/
RDF/HDT. Overview

- **Header**: Metadata describing the RDF dataset
- **Dictionary**: Mapping between IDs ↔ elements in the dataset
- **Triples**: Structure of the data after the ID replacement
RDF/HDT. Dictionary+Triples partition

Dictionary

1. <http://example.org/Valladolid>
2. <http://example.org/javier>
3. dbpedia:birthPlace
4. foaf:mbox
5. foaf:name
6. “Valladolid”
7. “jfergar@infor.uva.es”

Triples

1

2

3 4

5 6

7
So far...

- HDT << any other RDF format || RDF engine
- Competitive query performance.
  - Very fast on triple patterns, x 1.5 faster (Virtuoso, RDF3x).
  - Joins on the same scale of existing solutions (Virtuoso, RDF3x).
  - Still room for optimizations

- Could HDT be used for RDFS Entailment???
Index

- Motivation
  - Big Semantic Data
  - RDFS Entailment
  - Current Solutions
  - Our Objective
- RDF/HDT
- **Current Proposal: RDFS-HDT**
- Conclusions
RDFS-HDT

- Use the core idea from HDT
  - Minimal changes to be consistent with current tools and emerging community.
  - Equivalent size of the indexes
- Compute $G \models t$ on demand
  - Based on [Muñoz et al, 2009]:
    - Relevant RDFS keywords rdfs:subPropertyOf (sp), rdfs:subClassOf (sc), rdf:type (type), rdfs:domain (dom), and rdfs:range (range).
RDFS-HDT. *Compressed in-memory indexes*

- A traditional HDT Dictionary component:
  - Based on compressed string dictionaries.

- But four different HDT Triples components:
  1. The *subproperty* subgraph, $G(sp)$, comprises all triples (subject, *sp*, object).
  2. The *subclass* subgraph, $G(sc)$, comprises all triples (subject, *sc*, object).
  3. The subgraph containing dom and range, $G(dr)$, comprises all triples (subject, *dom*, object) or (subject, *range*, object).

  - $G(sp)$, $G(sc)$ and $G(dr)$ are based on (a) Traditional HDT Triples or (b) Direct Acyclic Graphs (DAGs)

4. All the remaining triples in the *general* subgraph $G_o$.
   - Based on Traditional HDT Triples
Based on simple SPARQL Triple Pattern resolution on HDT

\[ t = (a, \texttt{type}, b) \]?

```java
properties.push(resolveTP(a, ?p, ?x));
while (properties.hasElements()) {
    while (subproperties.hasElements()) {
        domains.push(resolveTP(subproperties.head, dom, ?d));
        while (domains.hasElements()) {
            if (domains.head == b) return true;
            else {
                domains.push(obtainParent(Gsc, domains.head));
                domains.pop();
            }
        }
        subproperties.push(obtainParent(Gsp, properties.head));
        subproperties.pop();
    }
    properties.pop();
}
```
RDFS-HDT. Some preliminary results of RDF/HDT

- Compression

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Nt Size (MB)</th>
<th>Virtuoso</th>
<th>RDF3X</th>
<th>HDT</th>
<th>HDT - FoQ</th>
</tr>
</thead>
<tbody>
<tr>
<td>2011 AU Census</td>
<td>52</td>
<td>38.46%</td>
<td>27.91%</td>
<td>3.02%</td>
<td>4.96%</td>
</tr>
<tr>
<td>Jamendo</td>
<td>144</td>
<td>109.72%</td>
<td>66.15%</td>
<td>10.22%</td>
<td>12.31%</td>
</tr>
<tr>
<td>AEMET</td>
<td>726</td>
<td>65.33%</td>
<td>33.67%</td>
<td>6.55%</td>
<td>8.07%</td>
</tr>
<tr>
<td>Dbtune</td>
<td>9,566</td>
<td>41.48%</td>
<td>43.13%</td>
<td>6.93%</td>
<td>9.16%</td>
</tr>
<tr>
<td>2000 US Census</td>
<td>21,796</td>
<td>25.40%</td>
<td>30.32%</td>
<td>3.73%</td>
<td>6.39%</td>
</tr>
<tr>
<td>Dbpedia 3-8</td>
<td>63,053</td>
<td>73.91%</td>
<td>48.46%</td>
<td>10.77%</td>
<td>13.36%</td>
</tr>
<tr>
<td><strong>MEAN</strong></td>
<td><strong>-</strong></td>
<td><strong>59.05%</strong></td>
<td><strong>41.61%</strong></td>
<td><strong>6.87%</strong></td>
<td><strong>9.04%</strong></td>
</tr>
</tbody>
</table>
Evaluation. Querying (TP)

- Querying TPs

![Graph showing time in ms for different query types]

- **DBTUNE**
  - i7 3820 @ 3.6Ghz
  - 16GB RAM
Index

- Motivation
  - Big Semantic Data
  - RDFS Entailment
  - Current Solutions
  - Our Objective
- RDF/HDT
- Current Proposal: RDFS-HDT
- Conclusions
Conclusions

- We position the idea of RDF Entailment of Big Semantic Data on demand, in-memory, on compressed RDF.
- Using the core idea from HDT (RDFS-HDT), performing SPARQL TP resolution.
- Promissory results in size and performance.

Ongoing Work:
- Test the initial prototype over a large corpus with different domains and schema characteristics.
- *Use other data structures providing ancestor-queries for* $G(sp)$ and $G(sc)$ subgraphs.