GraphScale:
Adding Expressive Reasoning to Semantic Data Stores

Thorsten Liebig, Vincent Vialard, Michael Opitz, and Sandra Metzl

derivo GmbH, Ulm, Germany

Abstract. We present GraphScale, a technology that empowers semantic data stores with OWL reasoning. It connects a given data store with any state of the art OWL reasoner. The underlying abstraction approach allows to efficiently perform a full materialization of the store based on sound and complete OWL 2 RL reasoning for high-performance querying.

1 Motivation

Triple stores and graph databases are popular systems for storing and retrieving semantic data. Like other types of databases their key features are capacity of data volume and query performance, but most of them also provide some reasoning functionality. Reasoning allows to enrich the original data with facts that follow from background knowledge typically expressed by an ontology. Most of the popular systems only support low level reasoning (RDF/RDFS). Some support selected OWL language constructs only by enabling additional inference rules. However, reasoning and querying data at an expressivity of OWL 2 RL is not efficient enough in practise. In contrast, efficient reasoners for expressive ontologies are typically in-memory systems that can not deal with huge amounts of data without expensive hardware. There is no satisfiable solution so far that combines efficient and expressive reasoning with a scalable storage back end.

GraphScale is a bridging technology for adding efficient OWL 2 RL reasoning to potentially any data store. The approach sketched in Figure 1 is based on an abstraction refinement technique that builds a compact representation of the data suitable for a state of the art in-memory OWL 2 reasoning system. The facts derived for this abstract representation are propagated back to the data store, and the abstraction is updated accordingly. This process is repeated until no new facts can be derived. The result is a fully materialized data store ready for querying via its built-in query interface or via the GraphScale API that uses the abstraction as an index.

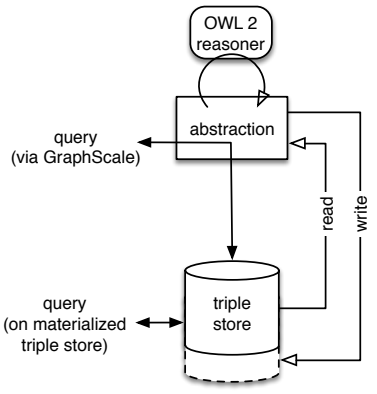

Fig. 1. The GraphScale approach

The whole process is achievable using standard interfaces of the data store and any OWL 2 reasoner. As a consequence the GraphScale technology allows the addition of expressive and efficient reasoning to existing data repositories at minimal cost.

2 GraphScale Approach

The idea behind GraphScale is based on the observation that entities within a data repository often have similar structure. For a resource in a triple store this refers to identical rdf:type information and same sets of properties. When considering the LUBM data set for example, there are many entities which are similar in terms of being a student and male, attending some courses and liking some sport. From a reasoning perspective they can be treated the same way for many inferences.

The GraphScale approach takes advantage of these similarities and computes a condensed version of the original data called abstraction, based on equivalence classes of entities that share a similar structure. The abstraction is an OWL ontology consisting of a set of star-shaped structures (see right panel in Figure 2) each of which is a representative for an equivalence class. Since the abstraction is typically a lot smaller (and less connected) than the original ontology, it can be fed to any existing in-memory OWL 2 reasoning system.

The facts derived by the OWL 2 reasoning system for the abstraction are propagated back to the data store. This might lead to entities being no longer similar and by using an iterative refinement step the abstraction is revised until a fixpoint is reached.

The process for computing and maintaining this abstraction follows a strategy that is provable sound and complete for the description logic Horn $\mathcal{ALCHOI}$. The corresponding proof as well as a detailed description of the abstraction refinement can be found in our previous work [2].

The $\mathcal{ALCHOI}$ logic covers almost all of the OWL 2 RL language profile. This profile is interesting for industry-scale semantic data processing because of its fair expressivity. In order to cover all of OWL 2 RL we extended the refinement procedure with property features such as functionality and property chains. The local derivations caused by the latter language features have to be carried out on the original data as a preprocessing step before building and updating the abstraction.

Figure 1 depicts the overall procedure of the GraphScale materialization. GraphScale builds the abstraction from the original data (read arrow) and collectively writes back all consequences (write arrow) derived with the help of an external, standard OWL 2 reasoning system. Any preprocessing also is a read-write cycle on the data back-end.

3 GraphScale System

The GraphScale system[1] is a Java-based implementation of the abstraction refinement briefly explained in the previous section. As a bridging technology the system provides

[1] `http://graphscale.de`

interfaces to the external OWL 2 reasoning engine as well as to the data back-end in order to support different options on either side.

By supporting the OWL API and OWLlink [3] the GraphScale system is open to virtually any available OWL 2 reasoning system. As of now we have mostly used Konclude [4] as well as HermiT [1] as OWL 2 engines. The data store interface requires just basic read/write access to entities, their type information and their properties, which is achievable through SPARQL Query + Update provided by most triple stores. A native API allowing combined operations will however allow for better performance. As of now there are implementations of native bindings to the graph database Neo4J[2] as well as to Oracle 12c.

The GraphScale approach offers parallelization options on both sides, the reasoner and the data store. Due to its unconnected structure, the abstraction can be split easily and fed to multiple prover instances running on different computers in order to save time. A more effective way to further increase throughput for reasoning or querying would be to replicate or shard the data store for better read/write performance. Since the abstraction is the key element for reasoning, existing replication or sharding technologies can be used for data storage and update without jeopardising soundness and completeness of reasoning. The GraphScale system can also run with its own in-memory data store for trading memory consumption for processing speed.

When using the GraphScale query interface to query the data store, the abstraction is used as a dedicated index. It contains precise structural information useful for query planing and even allows to answer type queries without accessing the data store.

4 Showcase

First, we will provide a live demonstration of a visualization tool that displays the original data and its abstract representation side by side (for data sets of moderate size). As an example, Figure 2 depicts the data graph of the NTN ontology[3] on the left and the collection of star shaped structures of the corresponding abstraction on the right. Among other features, the tool interactively highlights entities in the original ontology and their representatives in the abstraction. In the snapshot of Figure 2 the pointer is placed over a representative of the abstraction in the right panel and all corresponding entities of the original data set are highlighted in the left panel.

We will also provide recent benchmark results exhibiting the advantages of the abstraction for materialization and querying for a number of data sets with different characteristics. The scalability of the approach will be demonstrated using the LUBM and UOBM data sets at various sizes. These results will be compared with the results of other data stores with reasoning abilities such as RDFox[4], GraphDB[5] or Blazegraph[6].

[2] `http://neo4j.com`
[3] New Testament Names Ontology: `http://semanticbible.com/ntn/`
[4] `http://www.cs.ox.ac.uk/isg/tools/RDFox/`
[5] `http://graphdb.ontotext.com/`
[6] `http://www.blazegraph.com`

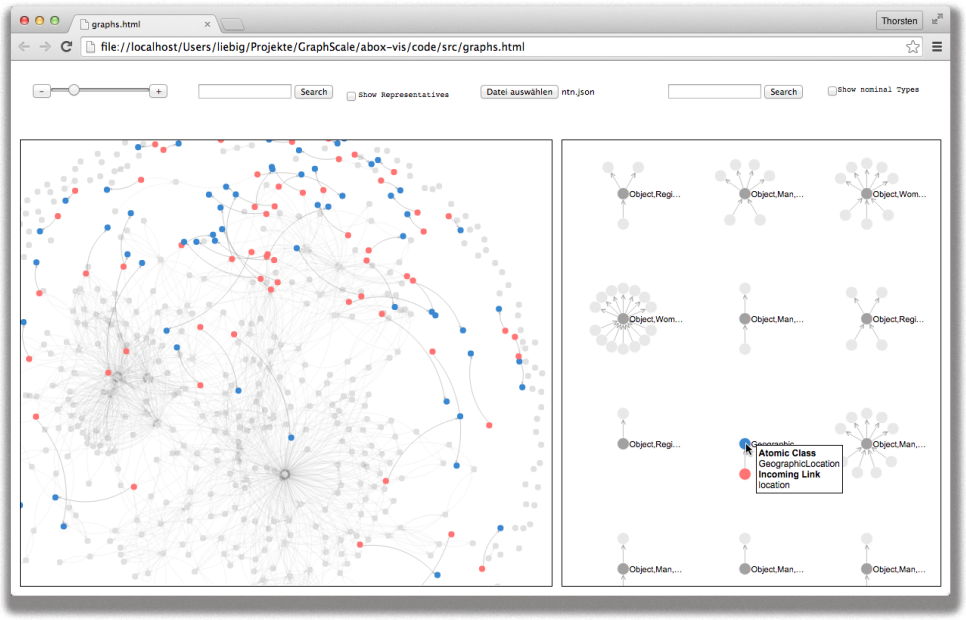

Fig. 2. Visualization of original data and its abstraction (both only partially visible)

We will show that GraphScale can even materialize ontologies faster than the expressive reasoning system it uses for the reasoning about the abstraction, thanks to the very small size of the abstraction in comparison to the original data.

5 Status

The GraphScale implementation shows extremely promising results for dealing with large and complex semantic data sets. We are currently optimizing the query planing and the next task will be the extension of the parallelization. We also plan to extend reasoning for streaming data. The system will be released under a dual license distinguishing academic and evaluation usage from commercial deployment.

References

1. B. Glimm, I. Horrocks, B. Motik, G. Stoilos, and Z. Wang. Hermit: An owl 2 reasoner. *Journal of Automated Reasoning (JAR)*, 53(3):245–269, 2014.
2. B. Glimm, Y. Kazakov, T. Liebig, T.-K. Tran, and V. Vialard. Abstraction refinement for ontology materialization. In *Proceedings of the 13th International Semantic Web Conference (ISWC 2014)*, volume 8796. Springer-Verlag, 2014.
3. T. Liebig, M. Luther, O. Noppens, and M. Wessel. Owllink. *Semantic Web – Interoperability, Usability, Applicability*, 2(1):23–32, 2011.
4. A. Steigmiller, T. Liebig, and B. Glimm. Konclude: System description. *Journal of Web Semantics*, 27(1):78–85, 2014.