Reasoning over Dynamic Data in Expressive Knowledge Bases with Rscale

Thorsten Liebig[1] and Michael Opitz[2]

[1] derivo GmbH, Ulm, Germany, liebig@derivo.de
[2] Ulm University, Ulm, Germany, michael.opitz@uni-ulm.de

Abstract. We introduce Rscale, a secondary storage-aware OWL 2 RL reasoning system capable of dealing with incremental additions and deletions of facts. Our initial evaluation indicates that Rscale is suitable for stream reasoning scenarios characterized by expressive reasoning tasks triggered by a moderate change frequency.

Keywords: OWL 2 RL, stream reasoning, incremental updates

1 Motivation

Streaming data is supposed to occur at varying volume and granularity. For instance, sensor data or stock quotations come at high frequency and as numerical data as opposed to streams for qualitative facility monitoring that also consists of states at a moderate frequency. Our work on Rscale is targeting at the latter scenario where numeric and symbolic data has to be processed while considering expressive background knowledge at the same time. Consider a power plant scenario where certain key parameters need to be constantly monitored. Within such a setting it is important to take sophisticated domain knowledge into account when interpreting the data and system states. As an example, a high tank pressure may be normal in case of certain open valves and a low temperature history. Additionally, in case of an abnormal state detection engineers might be interested in the particular fraction of the streaming data that caused this effect.

This kind of scenario needs a streaming-aware reasoning back-end that is capable of dealing with expressive knowledge as well as non-standard inference services such as providing justifications. In this paper we present Rscale, a rule-based OWL 2 RL reasoning engine that suits this requirements.

2 Fast OWL 2 RL Reasoning with Updates

Rscale[3] is an industrially approved system for scalable reasoning over dynamic ontologies within the expressive OWL 2 RL language profile. The system utilizes a relational database as secondary storage component and adopts the *incremental maintenance of materializations* idea [2, 4]. The Java-based Rscale reasoner implements the OWL API[4] and also comes as an OWLlink[5] server.

[3] http://www.rscale.de/ [4] http://owlapi.sourceforge.net/
[5] http://www.w3.org/Submission/2010/04/

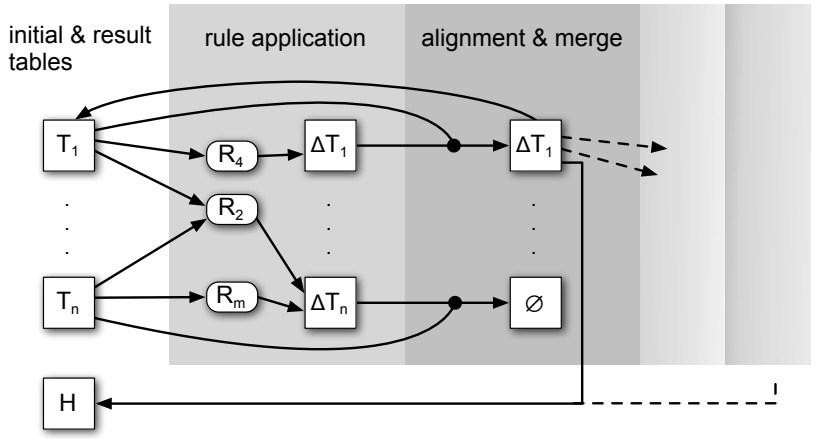

Fig. 1. Reasoning procedure of Rscale

The system design is different from a triplestore in that it abstracts from the triple storage model and applies a meta modeling approach inspired from our previous work [5] and maintains dedicated storage schemas for the various RL statements such as SubClassOf, ObjectPropertyDomain, ClassAssertion, etc.

Rscale implements the complete set of OWL 2 RL rules [3] which are known to be sound and complete under Direct Semantics for OWL 2 RL ABox inference tasks. The system incorporates a set of optimizations such as delta iteration and rule triggers.

Figure 1 illustrates the basic reasoning procedure. After initial storing of the ontology in its corresponding base tables $\mathsf{T}_1, ..., \mathsf{T}_n$ the rule engine triggers the applicable rules $\mathsf{R}_1, ..., \mathsf{R}_m$ as SQL queries for execution at the DB back-end. The execution results are written in delta tables $\Delta\mathsf{T}_i$. The subsequent alignment and merge phase will prune all already asserted base table facts from each delta table. The resulting deltas are the rule trigger for the next round and are finally added to the base tables as the inferred fact of the first round. These two-phase rounds are repeated until all delta tables are empty after alignment with the base tables (i.e. there are no new inferences).

In case of incremental additions Rscale writes new statements into new delta tables and re-activates its rule execution mechanism to materialize all successive inferences. A history table (H) optionally keeps book of all consequence dependencies and is updated after each processing round. This allows to prune all direct and indirect consequences of a deleted fact for efficient retraction. When deleting a statement the history table is used to identify the derived facts from the deleted sources. All derived facts will then be removed from the corresponding base tables. A triggering mechanism will then activate all those rules which populate these tables to re-establish those deleted consequences that are also inferable from the remaining facts. The history also provides justification for inferences in terms of streaming facts and background axioms.

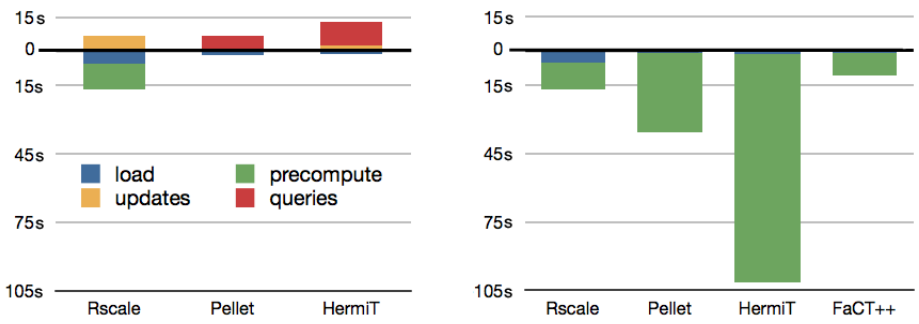

Fig. 2. Timings of streaming scenario (left) and precomputation of inferences (right)

3 Evaluation

Our evaluation aims at providing some performance data from available OWL 2 RL reasoning systems within a streaming setting of sequentially KB updates and queries. According to our focussed scenario we assume an OWL 2 RL TBox, a medium sized ABox and mostly ABox changes

Potential benchmark candidates are dedicated OWL 2 RL reasoning systems, such as OREL[6], DLEJena[7], Elly[8], or BaseVisor[9]. Surprisingly, non of these systems could be used for our evaluation. OREL, DLEJena, and Elly were not able to load and process our initial KB's within 10 minutes and 1 GB of RAM. BaseVisor could, but we could not manage to run incremental updates with this system. Therefore, we had to switch to more expressive, main memory reasoning systems such as FaCT++ 1.5.2[10], HermiT 1.3.4[11], and Pellet 2.2.2[12].

The KB for our spot test was a $\mathcal{ALCHOIF}$ RL ontology with 117 classes, 93 object properties, and 1259 individuals. The test procedure was made of a set of simple queries, successive ABox assertions (object property assertion), a TBox change (subClassOf removal and addition), and successive ABox removals (all in all 22 change-query cycles). The test was carried out on a standard Core2Duo Linux computer. The left hand side of Figure 2 shows the timing results divided into fixed processing time (below the dark zero line) and variable processing time caused by the consecutive queries and updates (above zero line). Since Rscale is an incremental materialization approach it does most of the work just once, namely when precomputing the ontology after loading. Consequently, updates require some incremental maintenance whereas querying time is negligible and just consist of DB lookups. The corresponding times for Pellet and HermiT utilize a non-buffering, non-precomputing mode, where reasoners can use their own minimal effort strategy. Figure 2 shows that they require more time on queries because they have to compute results in an on-demand fashion (FaCT++

[6] http://code.google.com/p/orel/ [7]http://lpis.csd.auth.gr/systems/DLEJena/
[8] http://elly.sourceforge.net/ [9]http://www.vistology.com/basevisor/basevisor.html
[10] http://code.google.com/p/factplusplus/ [11]http://www.hermit-reasoner.com/
[12] http://clarkparsia.com/pellet

failed to handle incremental updates). Concerning stream frequency this reveals a frequency of 3.2 updates for Rscale and Pellet as well as 1.7 updates per second for HermiT with respect to this test case.

For comparison the complete preprocessing time for all systems is depicted in the right hand side of Figure 2. This shows that preprocessing all inferences is costly for Pellet, HermiT, and FaCT++ and therefore not a feasible option within a dynamic scenario.

Results from tests with other KB's where often roughly comparable with the one above but also revealed some exceptions. For instance, larger KB's often caused an out of memory exception (here 1GB) or practical non-termination (here 10 min) for either FaCT++, Pellet, or HermiT. Rscale seems to scale fairly well with large KB's and never hit the main memory limit. On the other hand we also discovered test cases where Rscale ran eight times longer than the fastest main memory reasoner. To sum up, Rscale shows its main advantage when KB's are large, queries require expressive reasoning, and updates come at moderate frequency.

4 Future Work

The evaluation indicates that Rscale is a suitable approach for dealing with a streaming scenario that requires rich reasoning techniques, something that is not possible in pure RDFS-based systems and are not supported by Stream Reasoning techniques such as C-SPARQL [1]. The latter, however, provides a highly efficient approach for dealing with chronologically expiring data from streams – which, on the other hand, is not possible with Rscale yet. Therefore, potential future work includes the integration of C-SPARQL into Rscale – something for which we see no conceptual barrier.

We currently work on parallelizing the reasoning process of Rscale for more performance in multi-core environments. Other work aims at further optimizing updates and removals in Rscale as well as support for more complex queries.

References

1. Barbierie, D.F., Braga, D., Ceri, S., Della Valle, E., Grossniklaus, M.: Incremental Reasoning on Streams and Rich Background Knowledge. In: Proc. of the Extended Semantic Web Conference (ESWC 2010). Springer-Verlag (2010)
2. Gupta, A., Mumick, I.S., Subrahmanian, V.S.: Maintaining Views Incrementally. In: Proc. of SIGMOD Conference. pp. 157 – 166. ACM Press, New York (1993)
3. Motik, B., Grau, B.C., Horrocks, I., Wu, Z., Fokoue, A., Lutz, C.: OWL 2 Web Ontology Language: Profiles. Recommendation, World Wide Web Consortium (2009)
4. Volz, R., Staab, S., Motik, B.: Incrementally maintaining materializations of ontologies stored in logic databases. Journal of Data Semantics II 3360, 1 – 34 (2005)
5. Weithöner, T., Liebig, T., Specht, G.: Efficient Processing of Huge Ontologies in Logic and Relational Databases. In: Poster Proc. of ODBASE 2004. pp. 28–29. Springer Verlag (2004)