TriAL-QL: Distributed Processing of Navigational Queries*

Martin Przyjaciel-Zablocki[1], Alexander Schätzle[1], and Adrian Lange[2]

[1] Department of Computer Science, University of Freiburg, Germany
zablocki|schaetzle@informatik.uni-freiburg.de
[2] IIG Telematics, University of Freiburg, Germany
lange@iig.uni-freiburg.de

Abstract. Navigational queries are natural query patterns for RDF data, but yet most existing RDF query languages fail to cover all the varieties inherent to its triple-based model, including SPARQL 1.1 and its derivatives. TriAL* is one of the most expressive approaches but not supported by any existing RDF triplestore. In this paper, we propose TriAL-QL, an easy to write and grasp language for TriAL*, preserving its procedural structure. To demonstrate its feasibility, we provide a proof of concept implementation for Hadoop using Hive as execution layer and give some preliminary experimental results.

1 Introduction

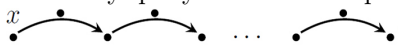
Graph databases and their respective graph query languages are commonly used for RDF data querying. However, in contrast to the standard graph model, an edge label in RDF (*predicate*) does not come from a finite alphabet and may also appear as a source or destination (*subject* and *object*, respectively) of another edge. Consequently, RDF query languages based on typical graph query languages like *nested regular expressions* (NRE) are not capable of such constructs and lose important features, e.g. reasoning over predicates within a query [2]. To the best of our knowledge, there are only two RDF query languages that enable expressive navigational capabilities with reasoning and can be evaluated in polynomial time, namely *Triple Query Language Lite* (TriQ-Lite) [3] and *Triple Algebra with Recursion* (TriAL*) [4]. TriQ-Lite is defined as a Datalog extension that captures SPARQL queries enriched with the OWL 2 QL profile, whereas TriAL* is a closed language that works directly with triples including recursion over *triple joins*. Although, the steady growth of Semantic Web data with its high degree of diversity in both, structure and vocabulary, justifies such expressive RDF query languages, it also raises the need for solutions that scale with the data size. In recent years, Hadoop has become the de-facto standard for processing Big Data, with a recent trend on scalable, interactive SQL-on-Hadoop solutions. The descent of TriAL* from relational algebra and its inherent compositionality led us to the decision to build an implementation of

* A substantially extended version will appear in [5]

TriAL* based on Hadoop. While TriAL* is a neat approach for querying RDF, its algebraic notation is inappropriate for practical usage. Thus, we propose the TriAL* Query Language (TriAL-Ql) that keeps the procedural structure of TriAL* by representing each algebra operation with a SQL-like statement. This way, even complex navigational queries are easy to grasp and write.

Our major contributions can be summarized as follows: (1) We introduce TriAL-QL, a query language for TriAL* with an intuitive mapping between both. (2) Next, we provide an Hadoop-based implementation of TriAL-QL using Hive. Optimizations include a precomputed 1-hop neighborhood, different evaluation strategies and a carefully chosen storage layout. (3) Finally, we show some preliminary experiments that demonstrate the scalability and feasibility of our approach.

2 TriAL-QL: A Procedural Query Language for RDF

The *Triple Algebra with Recursion* (TriAL*) [4] is one of the most expressive RDF query languages with polynomial complexity. In contrast to many other approaches, TriAL* is a closed and hence compositional language, i.e. the output is a set of triples rather than graphs or bindings. It works directly with triples, which allows us to write queries that are not expressible using query languages based on the standard graph model (e.g. *regular path queries* and *nested regular expressions*). TriAL* takes the relational algebra as its basis with some restrictions to guarantee closure. The most important operator is a *triple join* between two ternary relations E_1 and E_2 representing sets of triples, defined as: $E_1 \bowtie^{i,j,k}_{\theta,\eta} E_2$, where $i, j, k \in \{s_1, p_1, o_1, s_2, p_2, o_2\}$ indicate the implicit projection on three fields to keep the operation closed with s_1 referring to the subject of E_1, etc. θ represents the join conditions whereas η is a set of conditions between objects and data values. To express paths of arbitrary length, recursion is added with the *right* $(e \bowtie^{i,j,k}_{\theta,\eta})^*$ and $left$ $(\bowtie^{i,j,k}_{\theta,\eta} e)^*$ *Kleene closure*, where e is a TriAL* expression. Thus, a reachability query that asks for pairs (x, z) which follow the connection pattern 

corresponds to the TriAL* expression $(E \bowtie^{s_1,\ p_1,\ o_2}_{o_1=s_2})^*$ with E being a relation name in a triplestore (cf. [4] for more details).

TriAL-QL. Next, we introduce the notation of TriAL-QL. The basic idea is to flatten the algebra expressions of TriAL* to a sequence of interrelated statements. A complete grammar can be found on our project website [1]. Table 1 shows the algebra of TriAL* with the corresponding syntax in TriAL-QL. Each algebra operation is represented by exactly one SQL-like statement. Accordingly, we can express the previously discussed reachability query by the following TriAL-QL statement:

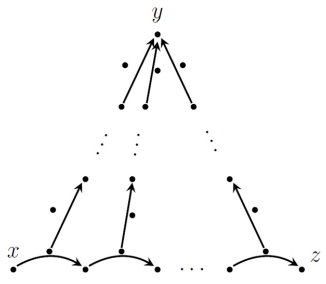
SELECT $s_1,\ p_1,\ o_2$ FROM E ON $o_1 = s_2$ USING *right*

[1] http://dbis.informatik.uni-freiburg.de/forschung/projekte/DiPoS/

Table 1. TriAL-QL Algebra & Syntax, where e, e_1 and e_2 correspond to a TriAL* expression. (i, j, k, θ, η as previously defined, cf. [4].)

Algebra (TriAL*)	**Syntax (TriAL-QL)**
$\sigma_{\theta,\eta}(e)$	SELECT i, j, k FROM e FILTER θ, η
$e_1 \bowtie^{i,j,k}_{\theta,\eta} e_2$	SELECT i, j, k FROM e_1 JOIN e_2 ON θ FILTER η
$e_1 \cup e_2$	e_1 UNION e_2
$e_1 - e_2$	e_1 MINUS e_2
$(e \bowtie^{i,j,k}_{\theta,\eta})^*$	SELECT i, j, k FROM e ON θ FILTER η USING $right$
$(\bowtie^{i,j,k}_{\theta,\eta} e)^*$	SELECT i, j, k FROM e ON θ FILTER η USING $left$

Next, we consider a more complex reachability problem introduced in [4] asking for pairs (x, z) which follow a connection pattern that requires reasoning capabilities:



TriAL*:

$e_1 = (E \bowtie^{s_1,\ o_2,\ o_1}_{p_1=s_2})^*$

$e_2 = (e_1 \bowtie^{s_1,\ p_1,\ o_2}_{o_1=s_2,\ p_1=p_2})^*$

$\Downarrow$

TriAL-QL:

$e_1 =$ SELECT $s_1,\ o_2,\ o_1$ FROM E
ON $p_1 = s_2$ USING $left$

$e_2 =$ SELECT $s_1,\ p_1,\ o_2$ FROM e_1
ON $o_1 = s_2,\ p_1 = p_2$ USING $left$

The inner expression e_1 computes the transitive closure of the predicates from E, while e_2 computes the transitive closure of this resulting relation. Again, we can see that TriAL-QL stays close to its TriAL* expression illustrating the strength of a compositional language where the result of the first statement can be used as input for the second. This makes TriAL-QL queries easy to write, understand and modify.

Further, new operators can be added smoothly, meeting our requirements. First, we extend the syntax of TriAL-QL with a STORE operation that enables us to materialize the result of a TriAL* expression e as a new relation in a triplestore. This way, not only the output but also intermediate results can be stored for later processing, if desired. Next, as we focus on processing web-scale RDF data, we have seen the need to introduce more control over the recursion depth of the $right$ and $left$ *Kleene* operator. Therefore, we introduce a scalar that replaces the *Kleene Star* and limits the number of join compositions. In TriAL-QL this can be formulated within the USING clause by writing, e.g., $left(4)$ for applying $left$ *Kleene* four times.

3 A Distributed Execution Engine for TriAL-QL

We implemented our execution engine on top of Hadoop using Hive as intermediate layer rather than working directly with MapReduce. This makes us independent of any Hadoop changes (e.g. Yarn) while taking advantage of continuously

optimized Hive versions or newer execution engines like Tez that come along with the recent SQL-on-Hadoop trend. Due to very limited space restrictions, we give only a brief introduction to our implementation. In short, a TriAL-QL query is first parsed to generate an abstract syntax tree and next mapped to TriAL* which is in turn translated into HiveQL queries. We investigated different evaluation strategies based on (1) *linear* and (2) *nonlinear* recursion as introduced in [1] and performed exhaustive experiments with different storage formats (e.g. RCFile, ORC, Parquet) and strategies (e.g. indices, partitions) to identify best practices for storing RDF data in Hive. Further, a precomputed 1-hop neighborhood reduces the amount of required joins.

Experiments. Some preliminary results are illustrated in Figure 1. We used the *Social Intelligence Benchmark* (SIB) data generator[2] to create social networks of different sizes. The left hand side (a) shows execution times and number of resulting triples for an exemplary query with three joins and one set operation using linear evaluation. Both series exhibit an almost linear scaling behavior. The right hand side (b) compares the linear to the nonlinear evaluation on a more complex query involving the *Kleene Star*. In this example, the nonlinear evaluation is superior to the linear one with increasing data size as it reduces the amount of required join iterations from 12 (linear) to 8 (nonlinear). Exhaustive experiments that include more advanced evaluation strategies are needed for more comprehensive conclusions and are part of ongoing work [5]. However, the first preliminary results already demonstrate the scalability and feasibility of our approach.

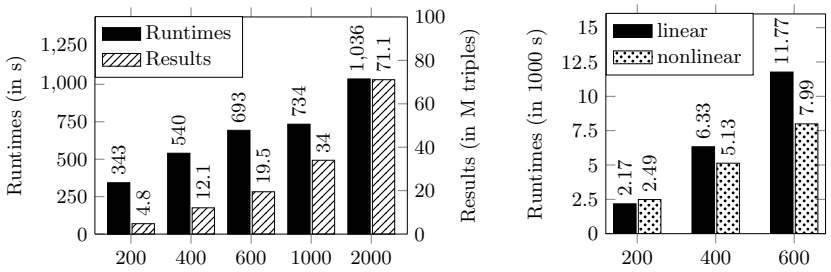


Fig. 1. (a) execution times vs. results (linear) (b) linear vs. nonlinear execution

References

1. Afrati, F.N., Borkar, V.R., Carey, M.J., Polyzotis, N., Ullman, J.D.: Map-reduce extensions and recursive queries. In: EDBT 2011, Sweden, March 21-24 (2011)
2. Angles, R.: A comparison of current graph database models. In: 28th ICDE Workshops, 2012, Arlington, VA, USA, April 1-5 (2012)
3. Arenas, M., Gottlob, G., Pieris, A.: Expressive languages for querying the semantic web. In: PODS'14, Snowbird, UT, USA, June 22-27, 2014 (2014)
4. Libkin, L., Reutter, J.L., Vrgoc, D.: Trial for RDF: adapting graph query languages for RDF data. In: PODS 2013, New York, NY, USA - June 22 - 27, 2013 (2013)
5. Przyjaciel-Zablocki, M., Schätzle, A., Lausen, G.: TriAL-QL: Distributed Processing of Navigational Queries. In: 18th WebDB 2015, Melbourne, Australia (2015)

[2] `http://www.w3.org/wiki/Social_Network_Intelligence_BenchMark`