Reasoning in RDFS is Inherently Serial, at least in the worst case

Peter F. Patel-Schneider

pfpschneider@gmail.com

There have recently been several papers presenting scalable distributed inference systems for the W3C Resource Description Framework Schema language (RDFS). These papers have made claims to the effect that they can produce the RDFS closure for billions of RDF triples using parallel hardware.

For example, Urbani *et al* claim "a distributed technique to perform materialization under the RDFS [...] semantics using the MapReduce programming model" [4]. Their initial algorithm used a small number of simple MapReduce passes in sequence to perform distributed materialization. In each pass all all triples are distributed according to the processing performed during that pass, but as well all *schema triples* (triples with predicate rdfs:domain, rdfs:range, rdfs:subPropertyOf, or rdfs:subClassOf) are sent to all reduce nodes. To handle some of the issues described here, later versions of their algorithm looped through the passes until no new inferences were made.

Similarly, Weaver and Hendler claim to be "materializing the *complete* [emphasis added] finite RDFS closure in a scalable manner" [6]. Their algorithm uses a single MapReduce pass and sends *ontological triples* (schema triples plus triples with predicate rdf:type and object rdfs:Datatype, rdfs:Class, or rdfs:ContainerMembershipProperty) to all reduce nodes. However, the algorithm distributes the other triples evenly, without regard to contents. Each reduce node then produces the finite RDFS closure of its input and these results are finally combined. They call their algorithm "embarrassingly parallel" because the inference part can be split into many independent pieces.

These claims seem believable, as on first glance there appears to be nothing to prevent the effective parallelization of RDFS inference. However, RDFS inference does have some aspects that make it less easy than one might expect.

First, RDFS has a formal model-theoretic semantics [1]. The model-theoretic semantics provides an unequivocal definition for inference and related notions in RDFS. The definition is not, however, in terms of a simple set of rules of inference, but is instead in terms of entailment, a model-theoretic notion. Claims of computing RDFS entailment, inference, closure, or materialization must thus be compared with this model-theoretic definition.

Second, RDFS has a infinite number of axioms. This trivially makes the RDFS closure of any RDF graph infinite. Necessarily these axioms are very simple and ter Horst [3] has discovered how to produce a finite subset of the RDFS closure that contains all the useful triples, and from which the remaining triples can be very easily computed.

Third, RDFS inherits the peculiarities of RDF, particularly that its internal vocabulary for defining ontologies can be manipulated in the same way as domain vocabularies can. This provides some power, but also provides considerable opportunities for mischief. One particularly mischievous way of manipulating the RDF and RDFS internal vocabulary is to use it to make extra connections within the internal vocabulary itself, as in

rdf:type rdfs:subPropertyOf rdfs:subPropertyOf .

Although this kind of manipulation is allowable, the problems it produces wildly outweigh its utility. Another mischievous way of manipulating the RDFS internal vocabulary is to subordinate it to non-RDFS vocabulary, as in

rdfs:subClassOf rdfs:subPropertyOf ex:interclassRelationship .

Although it may be interesting to create such "higher-level" vocabulary, such pursuits are generally best left to philosophers.

However, some manipulation of the RDF and RDFS vocabulary can be useful under certain circumstances. For example, one might want to make some differentiation between different kinds of properties and use a specific property for the sub-property relationship for them, as in

ex:partProperty rdfs:subClassOf rdf:Property .
ex:partSubPropertyOf rdfs:subPropertyOf rdfs:subPropertyOf .
ex:partSubPropertyOf rdfs:domain ex:partProperty .
ex:partSubPropertyOf rdfs:range ex:partProperty .

Although this ability to extend the RDF and RDFS ontology vocabulary does not appear to be much used in practice, with only two sub-properties of rdfs:subPropertyOf and three sub-properties of rdfs:subClassOf in the 2011 Billion Triple Challenge Dataset, it nonetheless can be useful.

When one is claiming to perform RDFS inference one needs to be very clear how one is handling these tricky issues. If one does not explicitly disclaim completeness then the RDF model-theoretic semantics provides the standard that must be adhered to. If one is claiming to produce RDF closures or perform RDFS materialization then one has to provide some way of handing the infinite axiomatization of RDFS. If one does not explicitly state which RDF graphs are not permitted as input then all RDF graphs must be allowed.

As noted above Urbani *et al* prominently claim to be performing RDFS materialization, and do so without any caveats in several places. It is only on close examination of their 2009 paper [5] that one finds that they ignore several inferences where domain classes or properties are made superclasses or superproperties of RDF or RDFS classes or properties and some inferences on container membership properties. These are the only qualifications related to RDFS materialization to be found in this paper. In their 2011 paper [4], there are some comments in the body of the paper about ignoring hijacking cases. There is also a comment about adding one extra step to handle special cases related to container membership properties and datatypes. Finally, there is a comment indicating that they also do not handle RDF and RDFS ontology vocabulary extension, but that it could be somehow handled by adding an extra loop. In

essence in this latter paper Urbani *et al* eliminate the possibility of modifying or extending the RDF and RDFS vocabulary and thus excluding the situations that make RDF and RDFS different from a very cut down version of OWL 2 DL [2]. What they do end up computing, provided that one ignores several bugs in the algorithms and fills in a lot of missing information, is fairly close to complete when their exclusions are taken into account. Their work does point the way to using a fixed number of simple MapReduce passes to materialize the *finite* RDFS closure when the RDF and RDFS vocabulary is neither modified nor extended.

Weaver and Hendler do state very clearly that they are only producing finite RDFS closures, but then very clearly state that they are complete. It is only in the middle of the paper that they state that they assume that there are no modifications or extension of the RDF and RDFS vocabulary. So, although Weaver and Hendler are up-front about the finite closure, counter to their claims of completeness they only consider the RDFS fragment of OWL 2 DL. The result of this algorithm appears to be nearly complete when their exclusions are taken into account, although this is somewhat hard to tell, as there are some bugs in the arguments in the paper. This work points the way to a simple distributed, single-pass method to materialize the *finite* RDFS closure when the RDF and RDFS vocabulary is neither modified nor extended.

So if the RDF and RDFS ontology vocabulary is neither modified nor extended it is possible to easily compute RDFS closures. However, just extending the RDF and RDFS ontology vocabulary produces problems. Consider

ex:physicalSubPropertyOf rdfs:subPropertyOf rdfs:subPropertyOf .
ex:physicalPartOf ex:physicalSubPropertyOf ex:partOf .
ex:wheelOf ex:physicalSubPropertyOf ex:physicalPartOf .

The closure of this RDF graph includes

ex:wheelOf rdfs:subPropertyOf ex:partOf .

which is not necessarily produced by Urbani *et al* or Weaver and Hendler. The basic problem here is that it is possible to infer new schema or ontological triples, and these triples can then produce further triples, even further ontological triples.

It might be the case that these failures are simply due to a bug or limitation in the algorithms, and that the general approach in these papers, or a slight generalization of it, can be used to perform finite RDFS materialization. However, this is not the case. Instead, it turns out that there are proofs in RDFS that are arbitrarily large but only depend on a fixed number of triples that contain any of the RDF or RDFS vocabularies.

An example of this comes from RDF graphs of the following form:

sp_1 rdfs:subPropertyOf rdfs:subPropertyOf .
sp_2 sp_1 sp_1 .
sp_3 sp_2 sp_2 .
...
sp_n sp_{n-1} sp_{n-1} .

This graph RDFS-entails

sp_n rdfs:subPropertyOf rdfs:subPropertyOf .

Any proof process that produces these results will need to perform an arbitrarily large amount of work involving triples that do not mention the RDF or RDFS vocabularies. Even worse, there is no commonality between the triples beyond the chaining from one triple to the next. This means that any distribution process will not be able to combine any significant fraction of the triples together, except by using this chain. The end result is that it is not possible to produce these proofs in a fixed number of finite parallel steps, even if determining the closure of all triples that include any RDF or RDFS vocabulary can be done in a fixed amount of time.

In fact, a simple reduction from the Monotone Circuit Problem shows that RDFS entailment is inherently serial, so it is unlikely that any significant speedup can be achieved using a polynomial number of processors, at least in the worst case. In cases like the one above a parallel implementation will have to produce inferences one by one down the chain. Of course, this example is rather contrived, but a system that performs RDFS materialization is not allowed to pick and choose its inputs; it has to handle all valid RDF graphs that have not been explicitly excluded.

So what then does this say about the empirical difficulty of computing finite RDFS closures using distributed computing, as opposed to the difficulty of describing just which are the problematic cases? Actually very little, as the example above demonstrates the worst case, and the general pattern is very unlikely. (The situation is, however, very different for OWL, as chaining of information along domain properties is quite common in OWL.) The investigations by Urbani *et al* and Weaver and Hendler show that computing the RDFS closure could be usually quite easy, particularly when excluding the mischievous cases that no sane person should argue for. Further, it would not be hard to detect cases where the RDF and RDFS vocabulary is being extended and to handle these cases specially. Although this would be slow in comparison to the normal processing, this is the price that users of the extension facility must pay.

References

1. Patrick Hayes. RDF semantics. W3C Recommendation, http://www.w3.org/TR/rdf-mt/, 2004.
2. Boris Motik, Peter F. Patel-Schneider, and Bijan Parsia. OWL 2 web ontology language: Structural specification and functional-style syntax. W3C Recommendation, http://www.w3.org/TR/owl2-syntax/, 2009.
3. Herman J. ter Horst. Completeness, decidability and complexity of entailment for RDF Schema and a semantic extension involving the OWL vocabulary. *Journal of Web Semantics*, 3(2-3):79–115, 2005.
4. Jacopo Urbani, Spyros Kotoulas, Jason Maassen, Frank van Harmelen, and Henri Bal. WebPIE: A web-scale parallel inference engine using MapReduce. *Journal of Web Semantics*, 10:59–75, 2012.
5. Jacopo Urbani, Spyros Kotoulas, Eyal Oren, and Frank van Harmelen. Scalable distributed reasoning using MapReduce. *ISWC-2009*, pages 634–649.
6. Jesse Weaver and James A. Hendler. Parallel materialization of the finite RDFS closure for hundreds of millions of triples. *ISWC-2009*, pages 682–697.