Editors-in-Chief
Journal of Web Semantics

Dear Sirs,

I am writing to you in regards to the paper “WebPIE: A Web-scale parallel inference engine using MapReduce”, by Jacopo Urbani, Spyros Kotoulas, Jason Maassen, Frank van Harmelen, and Henri Bal, that appears in the special issue of the *Journal of Web Semantics* on Scalability. I am very interested in the problem of scalable reasoning over large amounts of Semantic Web data and so have been perusing the articles in this special issue.

In their paper, Urbani *et al* claim to have “a distributed technique to perform materialization under the RDFS [...] semantics”. They use MapReduce, running three or four MapReduce steps in order. They also modify the usual MapReduce setup by distributing all so-called schema triples to all processes.

Their algorithm for RDFS reasoning is described in Appendix A.1, with much of the description in the form of the pseudo-code in Algorithms 3, 4, 5, and 6. This pseudo-code does not define any of the datatypes it uses. It also has many uninitialized variables (e.g., `derived` in Algorithm 3), `subproperties` and `superproperties` in Algorithm 4, and `domains` and `ranges` in Algorithm 5) and methods (e.g., `contains` and `recursive_get` in Algorithm 4 and `get_recursively` in Algorithm 6). It is thus impossible to determine just what the algorithm is actually doing and how well it would perform.

Even if there was an effective description of the algorithm in the paper, the claims of the paper are unsupported for a number of reasons, ranging from trivial to fatal.

First, the deductive closure of any set of triples in the RDFS semantics (which is what is being computed by materialization under the RDFS semantics) is infinite. Fortunately, there are several ways around this problem, the first
probably due to Herman ter Horst. However, the paper does not mention this problem at all.

Second, it is not sufficient to just run rules on the input triples. There are also the axiomatic RDFS triples to consider. Again, fortunately, this is a simple problem to overcome—just add the finite relevant set of axiomatic triples to the input. However, the paper does not mention this important step.

Third, the scalability of the approach in the paper depends on being able to scalably compute the transitive closure of the input \texttt{rdfs:subPropertyOf} and \texttt{rdfs:subClassOf} properties in a serial process and then store the result in main memory. The paper correctly points out that in practice there are usually not too many triples for these properties (roughly 2 million such triples in the billion triples of the Billion Triple Challenge). However, this is not an adequate analysis to show scalability. With 2 million triples in the input, the transitive closure may include 4 trillion triples, which can neither be scalably computed serially nor stored in main memory. Of course, this is not likely to be the case in practice, but the paper does not do this analysis.

Fourth, there is a significant bug in Algorithm 6, which uses only two reducers ("\texttt{0 rdf:type}" and "1 \texttt{rdfs:subClassOf}"). It appears that the mappers should use \texttt{value.subject} instead of \texttt{value.predicate}. As well, Algorithm 6 does not implement Rule 12 or Rule 13, contrary to the claim in the text.

The above problems can be overcome with some effort without changing the thrust of the paper, but there are problems that are not so easy to dispose of.

One basic problem is that it is unclear as to just what the authors think they are doing. On one hand, the authors say that they are performing RDFS materialization. On the other hand, the authors say that “they do not consider” certain aspects of the problem (in the caption of Figure 3) and that they “ignore the first case of rule 5” (in Section 3.5). So in the body of the paper, the authors appear to be doing something different from what they claimed in the abstract and elsewhere in the paper.

The non-considered aspects are not even very esoteric. It is quite reasonable to extend the RDFS vocabulary. One example would be to create a sub-property of \texttt{rdfs:subPropertyOf}, to be used for special kinds of properties
(perhaps to distinguish physical properties from other properties). However, in this case there is an interaction from Rule 7 to Rule 5.

So the materialization of

\[
\text{physicalSubPropertyOf rdfs:subPropertyOf rdfs:subPropertyOf}
\]
\[
\text{physicalPartOf rdfs:subPropertyOf partOf}
\]
\[
\text{wheelOf physicalSubPropertyOf physicalPartOf}
\]

includes

\[
\text{wheelOf rdfs:subPropertyOf partOf}
\]

But determining this requires first using Rule 7 to produce

\[
\text{wheelOf rdfs:subPropertyOf physicalPartOf}
\]

and only then using Rule 5 to produce the result above.

A second basic problem is that it is possible to infer schema triples from combinations of schema triples and non-schema triples, which then participate in other inferences. This is what makes RDFS materialization difficult to analyze. The above example illustrates one situation where this occurs, but there there is only a single non-schema triple so one could imagine easily fixing up the approach in the paper to cover it.

There, however, are cases that are even more difficult to handle. Consider, for example

\[
\text{physicalSubPropertyOf rdfs:subPropertyOf rdfs:subPropertyOf}
\]
\[
\text{wheelOf physicalSubPropertyOf physicalPartOf}
\]
\[
\text{wheel1 wheelOf car2}
\]

Here Rule 7 is first needed to produce

\[
\text{wheelOf rdfs:subPropertyOf partOf}
\]

but then another application of Rule 7 produces

\[
\text{wheel1 physicalPartOf car2}
\]
A new problem here is both the second and third triples are not schema triples. To perform the inference in one MapReduce step would require processing both these triples in the same reduce process, but the paper does not contain any attempt to do so.

Fixing this particular example might be done by doing an initial MapReduce pass for Rule 7 and then re-performing the schema triples processing and distribution.

The above changes to the approach in the paper are required for RDFS materialization but certainly may not be sufficient. A deep question, then, is whether the general approach in the paper can actually perform scalable RDFS materialization. Unfortunately, the answer to this question is no.

The general approach of the paper is to segregate a small, easily-identifiable subset of the triples (the schema triples). These triples are processed and the result is globally accessible. The remaining triples are then subject to a MapReduce step where each reducer sees only a small fraction of the remaining triples. The outputs of the reducers are triples, which are added back into the triple set. This whole process is then repeated a small, fixed number of times, based on a static analysis of the interactions between the RDFS inference rules. (The actual approach in the paper may not have the ability to augment the schema triples, but this is hard to determine, and adding this ability is only being generous to the approach.)

What causes this approach to fail in general is the presence of deep minimum inference trees in RDFS that involve non-schema triples at each step. As each reducer only sees a small number of non-schema triples at each step. As each reducer only sees a small number of non-schema triples at each step. The inference steps have to be done one after another on each path down the tree, so there needs to be a large (i.e., non-fixed) number of MapReduce iterations to produce the entire derivation.

An example of such an input is:

```
sp1 rdfs:subPropertyOf rdfs:subPropertyOf
sp2 sp1 sp1
sp3 sp2 sp2
sp4 sp3 sp3
...```

4
This input RDFS-entails

\[ \text{spn} \text{ rdfs:subPropertyOf} \text{ rdfs:subPropertyOf} \]

However, the shallowest proof of this result is \(O(n)\) deep and involves non-schema triples at each stage. To see that the result follows, suppose that

\[ \text{spi} \text{ rdfs:subPropertyOf} \text{ rdfs:subPropertyOf} \]

has been proved. Then Rule 7 can be used with this and

\[ \text{spi+1} \text{ spi} \text{ spi} \]

to derive

\[ \text{spi+1} \text{ rdfs:subPropertyOf} \text{ spi} \]

and then Rule 5 can be used to derive

\[ \text{spi+1} \text{ rdfs:subPropertyOf} \text{ rdfs:subPropertyOf} \]

An inductive process shows that the end result follows.

Because each reducer can only do a small fraction of this proof chain a fixed number of MapReduce iterations is unable to produce the the entire depth of the entire derivation, showing that the general approach in the paper is unable to perform RDFS materialization.

Of course, the example here is rather contrived, but this makes no matter. A system that performs RDFS materialization is not allowed to pick and
choose its inputs; it has to handle all valid RDF graphs.

Yours Sincerely,

Peter F. Patel-Schneider
Email: pfpschneider@gmail.com