Proc. 23rd Int. Workshop on Description Logics (DL2010), CEUR-WS 573, Waterloo, Canada, 2010.

A MapReduce Algorithm for $\mathcal{EL}^+$

Raghava Mutharaju, Frederick Maier, and Pascal Hitzler

Kno.e.sis Center, Wright State University, Dayton, Ohio

Abstract. Recently, the use of the MapReduce framework for distributed RDF Schema reasoning has shown that it is possible to compute the deductive closure of sets of over a billion RDF triples within a reasonable time span [22], and that it is also possible to carry the approach over to OWL Horst [21]. Following this lead, in this paper we provide a MapReduce algorithm for the description logic $\mathcal{EL}^+$, more precisely for the classification of $\mathcal{EL}^+$ ontologies. To do this, we first modify the algorithm usually used for $\mathcal{EL}^+$ classification. The modified algorithm can then be converted into a MapReduce algorithm along the same key ideas as used for RDF schema.

1 Introduction

The realization of Semantic Web reasoning is central to substantiating the Semantic Web vision [8]. By its very nature, automated reasoning requires a formal representation of knowledge, and in the Semantic Web at least RDF [14] and OWL [9] are two languages commonly used for this purpose. OWL, which is essentially the description logic $\mathcal{SROIQ}$, is considerably more expressive than RDF, and reasoning with it is therefore computationally more expensive. However, restricted *profiles* of OWL 2 (including OWL 2 EL, which is essentially the description logic $\mathcal{EL}^{++}$ [2]) have been developed [15], and for each of these polynomial time algorithms exist for standard inferencing tasks.

There is a large amount of data that is exposed on the Web in RDF and OWL formats. E.g., in a recent discussion of Linked Open Data [5], it is estimated that there are approximately 4.7 billion RDF triples on the Web interlinked by 142 million RDF links.[1] Reasoning with such large amounts of data is inherently difficult due to the high computational complexity of RDF and OWL reasoning. At the same time, however, it has been argued that the Linked Open Data cloud is in need of more expressive schema knowledge, knowledge of a sort expressible in OWL [10]. In order to reason with such data, scalable reasoning algorithms are essential, and parallelization of reasoning is one of the obvious routes to investigate in achieving the required scalability.

The present paper describes the first steps in an effort toward achieving that end. Specifically, we present a parallel algorithm for classifying $\mathcal{EL}^+$ontologies using MapReduce, which is a programming model and software framework for

[1] Some OWL is used as well, usually for indicating that two resources should be considered equal, using `owl:sameAs`.

distributed processing of data on clusters of machines. In doing so, we follow the lead of [21, 22], where the MapReduce framework was successfully applied for computing RDF Schema closure and for reasoning with OWL Horst.

These publications are part of a recent trend in Semantic Web reasoning to explore parallelization of reasoning tasks. Some of the most notable recent developments are the use of the MapReduce framework for RDF [19, 21, 22], using distributed hash tables for RDF Schema [11], the MaRVIN peer-to-peer platform for RDF [16], and the approach in [23] for parallel computation of RDF Schema closures. However, there is relatively little work on attempting to carry these successes over to OWL reasoning, apart from some investigations into OWL Horst [18, 21], OWL RL [13], distributed resolution for $\mathcal{SHIQ}$ ontology networks [17], and some preliminary investigations [1, 6].

The remainder of the paper is structured as follows. Background information on $\mathcal{EL}^+$ and the MapReduce framework is provided in Section 2, and the new algorithm is described in Section 3. An example illustrating how the algorithm works is also given. Section 4 concludes with directions for future research.

Acknowledgements. We thank Keke Chen, Frank van Harmelen, Spyros Kotoulas and Jacopo Urbani for helpful discussions.

2 Preliminaries

2.1 The Description Logic $\mathcal{EL}^+$

Concepts in $\mathcal{EL}^+$ [3, 4, 2] are formed according to the grammar

$$C ::= A \mid \top \mid C \sqcap D \mid \exists r.C,$$

where A ranges over concept names, r over role names, and C, D over (possibly complex) concepts. An $\mathcal{EL}^+$ *ontology* is a finite set of *general concept inclusions* (*GCIs*) $C \sqsubseteq D$ and *role inclusions* (*RIs*) $r_1 \circ \cdots \circ r_n \sqsubseteq r$, where C, D are concepts, n is a positive integer and $r, r_1, \ldots, r_n$ are role names.

The CEL algorithm [4] performs *classification* of an $\mathcal{EL}^+$ ontology, i.e., it computes the complete subsumption hierarchy between all concept names occurring in the ontology. Classification is one of the standard reasoning tasks. The algorithm first transforms the ontology into *normal form*, which requires that all concept and role inclusions are of one of the forms shown in the left part of Figure 1. This can be done in linear time [2]. For the remainder of the paper, we assume that input ontologies are in normal form.

The algorithm is formulated in terms of two mappings S and R, where $S(X)$ maps a class name X to a set of class names, and $R(r)$ maps each role name r to a set of class name pairs. Intuitively, $B \in S(A)$ implies $A \sqsubseteq B$, and $(A, B) \in R(r)$ implies $A \sqsubseteq \exists r.B$. For purposes of the algorithm, $\top$ is taken as a concept name. The mappings are initialized by setting $S(A) = \{A, \top\}$ for each class name A in the input ontology $\mathcal{O}$, and $R(r) = \emptyset$ for each role name in $\mathcal{O}$. The sets $S(A)$ and $R(r)$ are then extended by applying the completion rules shown in the right part of Figure 1 until no rule is applicable.

Normal Form	Completion Rule
$A_1 \sqcap \cdots \sqcap A_n \sqsubseteq B$	**R1** If $A_1, \ldots, A_n \in S(X)$, $A_1 \sqcap \cdots \sqcap A_n \sqsubseteq B \in \mathcal{O}$, and $B \notin S(X)$ then $S(X) := S(X) \cup \{B\}$
$A \sqsubseteq \exists r.B$	**R2** If $A \in S(X)$, $A \sqsubseteq \exists r.B \in \mathcal{O}$, and $(X, B) \notin R(r)$ then $R(r) := R(r) \cup \{(X, B)\}$
$\exists r.A \sqsubseteq B$	**R3** If $(X, Y) \in R(r)$, $A \in S(Y)$, $\exists r.A \sqsubseteq B \in \mathcal{O}$, and $B \notin S(x)$ then $S(X) := S(X) \cup \{B\}$
$r \sqsubseteq s$	**R4** If $(X, Y) \in R(r)$, $r \sqsubseteq s \in \mathcal{O}$, and $(X, Y) \notin R(s)$ then $R(s) := R(s) \cup \{(X, Y)\}$
$r \circ s \sqsubseteq t$	**R5** If $(X, Y) \in R(r)$, $(Y, Z) \in R(s)$, $r \circ s \sqsubseteq t \in \mathcal{O}$, $(x, Z) \notin R(t)$ then $R(t) := R(t) \cup \{(X, Z)\}$

Fig. 1. The CEL algorithm for $\mathcal{EL}^+$

The algorithm is guaranteed to terminate in polynomial time relative to the size of the input ontology, and it is also sound and complete: after termination, $B \in S(A)$ if and only if $A \sqsubseteq B$ holds, for all class names A and B.

2.2 MapReduce

MapReduce is a programming model for distributed processing of data on clusters of machines (each machine being called a *node*) [7]. The data set to be processed is divided into multiple chunks, and each chunk is assigned to an idle node. There are three different types of node, and each type has its own function.

Master: The Master node assigns chunks to Map nodes and passes the intermediate output locations to Reduce nodes. It also takes care of node failures.

Map: Map nodes accept data chunks from the Master and generate intermediate output according to a user-defined function. In its general form, the function accepts a key-value pair and returns a set of key-value pairs. The output pairs are typically written to a local disk, and the location of these is returned to the Master. The functionality of Map nodes can be represented as

$$\text{Map} : (k_1, v_1) \mapsto \text{list}(k_2, v_2).$$

Reduce: Reduce nodes are notified of the locations of intermediate output. They group values by key, and then process the values according to a user-defined Reduce function. One or more output values is produced. The general process can be represented as

$$\text{Reduce} : (k_2, \text{list}(v_2)) \mapsto \text{list}(v_3).$$

There are several prominent implementations of the MapReduce model.[2] Using them, developers need only define the Map and Reduce functions. Lower level and administrative tasks, such as allocating data to nodes and recovering from failures, are handled by general purpose components of the system.

[2] E.g., Hadoop (`http://hadoop.apache.org/`) is a popular Java implementation.

3 MapReduce for $\mathcal{EL}^+$

We follow the lead of [22], which describes a MapReduce algorithm for computing RDF Schema closures. However, since the completion rules from Figure 1 are structurally more complicated than the RDF Schema completion rules, we cannot straightforwardly adopt their approach. In the submitted paper [21], the authors extend their approach to OWL Horst [20], facing structurally similar problems. However, due to the specific knowledge bases they are looking at, they choose a solution which is not applicable in our case. We will return to this discussion in Section 3.2, after presenting our algorithm.

3.1 Revising the CEL algorithm

Considering the completion rules in Figure 1, it is rather straightforward to cast rules R2 and R4 into a MapReduce format, and we will see in Section 3.2 how this is done. Rules R1, R3, and R5, however, cannot be transformed directly, and so we first give an alternative formulation of the CEL algorithm. The rules of the reformulated algorithm are cast into a MapReduce format in Section 3.2.

The reformulation requires an additional function P (which stands for *Partial*) and an extension of the function R. These serve to split some of the completion rules from Figure 1 into two rules, explained shortly.

The function P maps each class name X (including $\top$) to a set of pairs (A, B), where A and B are class names (again including $\top$). Intuitively, $(A, B) \in P(X)$ implies $A \sqcap X \sqsubseteq B$. Initially, $P(X)$ is set to $\emptyset$ for each X.

R is extended to map expressions of the form $r \circ s$, for role names r and s, to pairs of class names (possibly including $\top$). The intuition remains the same, however: $(A, B) \in R(r \circ s)$ implies $A \sqsubseteq \exists(r \circ s).B$. The latter expression is not a valid $\mathcal{EL}^+$ expression, but it is semantically unproblematic. Furthermore, it causes no problems in the algorithm, since we do not formally deal with such expressions, and in particular they are not allowed in the input ontology.

We also require another normalization step: Each axiom of the form $A_1 \sqcap \cdots \sqcap A_n \sqsubseteq A$, for $n > 2$, is replaced by $n-1$ axioms $A_1 \sqcap A_2 \sqsubseteq N_1$, $N_1 \sqcap A_3 \sqsubseteq N_2$, $\ldots$, $N_{n-2} \sqcap A_n \sqsubseteq A$, where all N_i are class names not occurring anywhere else in the final knowledge base. This transformation obviously retains the original subsumption hierarchy between named classes of the original ontology.

Our revised algorithm for $\mathcal{EL}^+$ is now identical to the algorithm presented in Section 2.1, except that the completion rules from Figure 1 are replaced with those in Figure 2, and the input is now required to be in the modified normal form. The algorithm terminates if no application of any of the rules extends any of the sets $S(X)$, $R(r)$, $P(X)$, or $\mathcal{O}$.

Let us explain the rationales behind the new completion rules. The original rule R1 can be simulated by subsequent applications of R1-1 and R1-2 (note that an inclusion axiom of the form A $\sqsubseteq$ B—that is, where $n = 1$—is covered by R1-2 alone). At the same time, output produced by applying R1-1 is only used in the precondition of R1-2, and so it does not have any other effect on the outcome of the overall algorithm. Rule R2 is left untouched apart from

Normal Form	Completion Rule	Key
$A_1 \sqcap A_2 \sqsubseteq B$	**R1-1** If $A_1 \in S(X)$ and $A_1 \sqcap A_2 \sqsubseteq B \in \mathcal{O}$ then $P(X) := P(X) \cup \{(A_2, B)\}$	A_1
$(A, B) \in P(X)$	**R1-2** If $A \in S(X)$ and $((A, B) \in P(X)$ or $A \sqsubseteq B \in \mathcal{O})$ then $S(X) := S(X) \cup \{B\}$	A
$A \sqsubseteq \exists r.B$	**R2** If $A \in S(X)$ and $A \sqsubseteq \exists r.B \in \mathcal{O}$ then $R(r) := R(r) \cup \{(X, B)\}$	A
$\exists r.A \sqsubseteq B$ for A	**R3-1** If $A \in S(X)$ and $\exists r.A \sqsubseteq B \in \mathcal{O}$ then $\mathcal{O} := \mathcal{O} \cup \{\exists r.X \sqsubseteq B\}$	A
$\exists r.A \sqsubseteq B$ for r	**R3-2** If $(X, Y) \in R(r)$ and $\exists r.Y \sqsubseteq B \in \mathcal{O}$ then $S(X) := S(X) \cup \{B\}$	r (or Y)
$r \sqsubseteq s$	**R4** If $(X, Y) \in R(r)$ and $r \sqsubseteq s \in \mathcal{O}$ then $R(s) := R(s) \cup \{(X, Y)\}$	r
$r \circ s \sqsubseteq t$	**R5-1** If $(X, Z) \in R(r)$ and $(Z, Y) \in R(s)$ then $R(r \circ s) := R(r \circ s) \cup \{(X, Y)\}$	Z

Fig. 2. Revised CEL algorithm for $\mathcal{EL}^+$. The keys are used in the MapReduce algorithm. Note that in R4, r is allowed to be compound, i.e., of the form $s \circ t$.

removing the precondition $(X, B) \notin R(X)$, which is used only for termination purposes. Since we have reworded the termination condition, there is in effect no difference between the two versions of the rule. The reason for rewording is that the new termination condition is more easily cast into MapReduce format. The original rule R3 can be simulated by subsequent applications of R3-1 and R3-2. Rule R3-1 introduces new axioms into $\mathcal{O}$, but since the new axioms are logical consequences of the knowledge base, they do not affect soundness or completeness of the algorithm. Note that this also does not cause any problems with respect to termination, since there is a finite upper bound[3] on the number of possible axioms of the form $\exists r.X \sqsubseteq B$. Rule R4 is again left unchanged, apart from the fact that it now also applies to compound expressions of the form $s \circ t$. Note, however, that this extension of R4 is semantically sound, and so it does not affect the correctness of the algorithm. The original rule R5 can be simulated by subsequent application of R5-1 and R4—the latter in this case using the extended form with composed roles. The newly introduced output produced by applying R5-1 is sound, and so the correctness of the algorithm is again left unaffected.

It is straightforward to show formally that our revised algorithm is indeed sound, complete, and terminating. It is also of polynomial worst-case complexity in the size of the input ontology; this can be shown easily along the lines of argument for the algorithm presented in [3] for $\mathcal{EL}^{++}$.

[3] The upper bound is $k \cdot l^2$, where k is the number of role names, and l is the number of class names (including $\top$) in $\mathcal{O}$.

3.2 Parallelization using MapReduce

We now convert the completion rules of Figure 2 into MapReduce algorithms computing the closure of $\mathcal{O}$, S, R and P. Initialization is done by setting $S(A) = \{A, \top\}$ and $P(A) = \emptyset$, for each class name A (including $\top$), and $R(r) = \emptyset$ for each role name r. In the discussion below, we freely switch between viewing S, R, and P as maps and viewing them as sets, and we slightly abuse terminology by referring to all expressions of the form $A \in S(X)$, $(A, B) \in P(X)$ or $(X, Y) \in R(r)$, in addition to all elements of $\mathcal{O}$, as *axioms*.

The completion rules are applied in an iterative manner, picking one rule to apply in each iteration. The rules are interdependent and the results of previous iterations are reused in subsequent iterations. This is realized by adding the outputs of each iteration to the database where $\mathcal{O}$, S, R and P are stored.

The general strategy that is followed by all the algorithms is given in Figure 3. The set of axioms (taken from $\mathcal{O}$, S, P, and R) forms the input. This set is divided into multiple chunks, and each chunk is distributed to different computing nodes. These first act as map nodes and then as reduce nodes, thereby completing the parallel application of one of the completion rules.

To give a concrete example, consider rule R2. Each map node first identifies, in its input chunk, all axioms of the form $A \in S(X)$ and $A \sqsubseteq \exists r.B$ and then outputs them as key-value pairs $\langle A, A \in S(X)\rangle$ and $\langle A, A \sqsubseteq \exists r.B\rangle$, respectively. In the reduce phase, all pairs with key A end up in the same reduce node, which can then complete the application of R2 by adding to R.

This idea of casting completion rules into MapReduce closely follows [22]. Note, however, that the rules R1, R3, and R5 from the original CEL algorithm cannot directly be dealt with using this approach, since each of them has three preconditions which do not share a common element which could be used as a key for the reduce phase. In [21], which deals with OWL Horst, a similar problem occurs, and the authors deal with it by performing part of the operation in-memory using a central store. This is made possible because of the specific form of the problematic completion rules for OWL Horst, where one of the preconditions is always a schema axiom, and because of the specific applications the authors have in mind, where there is much less schema knowledge than facts. Note that we cannot adopt this approach for $\mathcal{EL}^+$, since a separation along similar lines would hardly be reasonable for studying classification in $\mathcal{EL}^+$. We hence choose to provide a generic algorithm, based on the revised CEL algorithm.

We refrain from giving detailed descriptions of the MapReduce algorithms corresponding to all of the completion rules in Figure 2. We do give details for rules R1-1 and R1-2 in Figures 4 and 5, however. The remaining rules are dealt with in a completely analogous manner, using the keys shown in Figure 2.

The behavior of R1-1 and R1-2 can be illustrated using the below axioms.

$$\begin{aligned} \mathrm{A} \sqcap \mathrm{B} &\sqsubseteq \mathrm{C} \\ \mathrm{A} &\sqsubseteq \mathrm{B} \\ \mathrm{A} &\sqsubseteq \mathrm{D} \end{aligned}$$

ComputeClassificationSet()
{

1. Each node takes a subset of all axioms as input and computes additional elements for $\mathcal{O}$, S, R or P, depending on the completion rule which is applied.
   (a) In the Map phase, based on the rule that is applied in the current iteration, the set of axioms which satisfy any one of the preconditions of the rule are found and given as output. In the output $\langle \text{key}, \text{value} \rangle$ pairs, *key* is the concept or relationship which is common to both the preconditions of a rule (as indicated in Figure 2). The value is the corresponding axiom (which satisfies the precondition).
   (b) In the Reduce phase, all axioms belonging to the same key are collected from different nodes and conclusions of the completion rule are computed according to the completion rule, taking all valid combinations of axioms into account.
2. All outputs are stored in the database, unless they are already contained in it.
3. Call ComputeClassificationSet() again until no selection of a rule results in any additions to the database.

}

Fig. 3. General strategy followed by MapReduce algorithms for each completion rule.

It readily follows that A is a subclass of both C and D, and one can obtain this result using R1-1 and R1-2 alone. When the algorithm is initialized, $S(X) = \{X, \top\}$ and $P(X) = \emptyset$ for each class name X. When R1-1 is applied (we may suppose that both axioms are given to a single node), the map function generates the key-value pair $\langle \text{A}, \text{A} \sqcap \text{B} \sqsubseteq \text{C} \rangle$. Other pairs are produced as well (specifically, $\langle X, S(X) \rangle$ and $\langle \top, S(X) \rangle$, for each name X). This intermediate output is used in the reduce phase, which in this example produces the following result: For key A, $v_1 = \text{A} \in S(\text{A})$ and $v_2 = \text{A} \sqcap \text{B} \sqsubseteq \text{C}$ together cause (B, C) to be added to $P(\text{A})$. This new axiom is added to the set of axioms already present, and all axioms—old and new—are used in the next map-reduce step.

The map phase of R1-2 then yields the following key-value pairs:

$$\{\langle \text{A}, \text{A} \in S(\text{A}) \rangle, \langle \text{B}, (\text{B}, \text{C}) \in P(\text{A}) \rangle, \langle \text{A}, \text{A} \sqsubseteq \text{B} \rangle, \langle \text{A}, \text{A} \sqsubseteq \text{D} \rangle\}$$

In the reduce phase, since $\text{A} \in S(\text{A})$ and $\text{A} \sqsubseteq \text{B}$ are both associated with key A, B is added to $S(\text{A})$. Analogously, D is added to $S(A)$. These are the only significant changes made during the iteration. Applying R1-2 again, however, causes C to be added to $S(\text{A})$ as well. Specifically, when the map function is invoked, since B is now an element of $S(\text{A})$, the pair $\langle \text{B}, \text{B} \in S(\text{A}) \rangle$ will be generated, as will $\langle \text{B}, (\text{B}, \text{C}) \in P(\text{A}) \rangle$. In the reduce phase, since both tuples are now indexed by the same key (namely, B), they can be used in conjunction. It is this that allows C to be added to $S(\text{A})$.

```
map(key, value)
    // key: line number (not relevant)
    // value: an axiom
{
    if(value == A ∈ S(X))
        emit(⟨A, A ∈ S(X)⟩);
    else if(value == A1 ⊓ A2 ⊑ B)
        emit(⟨A1, A1 ⊓ A2 ⊑ B⟩);
}
reduce(key, iterator values)
    // key: A concept name (e.g. A)
    // values: axioms corresponding to a rule precondition
{
    for each v1 in values
        for each v2 in values
        {
            if(v1 == A1 ∈ S(X) and v2 == A1 ⊓ A2 ⊑ B)
                emit((A2, B) ∈ P(X));
        }
}
```

Fig. 4. MapReduce algorithm for R1-1. The input of the map function is an axiom, taken from either the ontology $\mathcal{O}$, or else one generated from the sets S, P, or R. Key-value pairs are generated, which are used in the reduce phase. The reduce function accepts a key and a list of values. Every possible combination of values is examined to determine whether R1-1 is applicable. A list of axioms is produced.

4 Conclusion and Future Work

Due to the ever increasing amount of data on the Web, there is a need for parallelizable approaches to reasoning algorithms. Following the lead of existing work on scalable implementations of RDF Schema closure, we have in this paper provided a MapReduce algorithm for the classification of $\mathcal{EL}^+$ ontologies. This approach, we believe, is scalable and will reduce the time needed to compute classification over large ontologies.

Our next step is to implement this algorithm using the Hadoop framework and the cloud computing infrastructure available at Wright State University. The experiences reported in [21, 22] on using MapReduce for RDF Schema indicate that optimizations, in particular concerning the choice of which completion rule is applied next, will be crucial for the performance. We intend to use the master node to monitor the outputs of previous MapReduce steps, and to use this output to decide which completion rule to apply next.

The results in [21, 22] also indicate that the MapReduce approach requires rather large datasets to show a pay-off in terms of performance, which in our case may require the generation of artificial datasets for initial experiments.

```
map(key, value)
    // key: line number (not relevant)
    // value: an axiom
{
    if(value == A ∈ S(X))
        emit(⟨A, A ∈ S(X)⟩);
    else if(value == (A, B) ∈ P(X))
        emit(⟨A, (A, B) ∈ P(X)⟩);
    else if(value == A ⊑ B)
        emit(⟨A, A ⊑ B⟩);
}
reduce(key, iterator values)
    // key: A concept (e.g., A)
    // values: axioms corresponding to a rule precondition
{
    for each v1 in values
        for each v2 in values
        {
            if(v1 == A ∈ S(X))
            {
                if(v2 == (A, B) ∈ P(X) or v2 == A ⊑ B)
                    emit(B ∈ S(X));
            }
}
```

Fig. 5. MapReduce algorithm for R1-2

We furthermore consider this line of work on $\mathcal{EL}^+$ to be only the starting point for investigations into more expressive languages, such as $\mathcal{EL}^{++}$, ELP [12], or even OWL 2 DL.

References

1. Mina Aslani and Volker Haarslev. Towards Parallel Classification of TBoxes. In Franz Baader, Carsten Lutz, and Boris Motik, editors, *Proceedings of the 21st International Workshop on Description Logics (DL2008), Dresden, Germany, May 13-16, 2008*, volume 353 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2008. Available from http://ceur-ws.org/Vol-353/AslaniHaarslev.pdf.
2. Franz Baader, Sebastian Brandt, and Carsten Lutz. Pushing the EL envelope. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI-05)*, Edinburgh, UK, 2005. Morgan-Kaufmann Publishers.
3. Franz Baader, Carsten Lutz, and Boontawee Suntisrivaraporn. CEL—A Polynomial-time Reasoner for Life Science Ontologies. In U. Furbach and N. Shankar, editors, *Proceedings of the 3rd International Joint Conference on Automated Reasoning (IJCAR'06), Seattle, WA, USA, August 17-20, 2006*, volume 4130 of *Lecture Notes in Artificial Intelligence*, pages 287–291. Springer-Verlag, 2006.

4. Franz Baader, Carsten Lutz, and Boontawee Suntisrivaraporn. Efficient reasoning in $\mathcal{EL}^+$. In *Proceedings of the 2006 International Workshop on Description Logics (DL2006)*, volume 189 of *CEUR Workshop Proceedings*, 2006.
5. Christian Bizer, Tom Heath, and Tim Berners-Lee. Linked data – the story so far. *International Journal on Semantic Web and Information Systems*, 5(3):1–22, 2009.
6. Jürgen Bock. Parallel Computation Techniques for Ontology Reasoning. In Amit P. Sheth et al., editors, *Proceedings of the 7th International Semantic Web Conference, ISWC 2008, Karlsruhe, Germany, October 26-30, 2008*, volume 5318 of *Lecture Notes in Computer Science*, pages 901–906. Springer, 2008.
7. Jeffrey Dean and Sanjay Ghemawat. MapReduce: Simplified data processing on large clusters. In *Proceedings of the 6th Symposium on Operating System Design and Implementation (OSDI 2004), December 6-8, 2004, San Francisco, California, USA*, pages 137–150. USENIX Association, 2004.
8. Pascal Hitzler. Towards reasoning pragmatics. In Krzysztof Janowicz, Martin Raubal, and Sergei Levashkin, editors, *GeoSpatial Semantics, Third International Conference, GeoS 2009, Mexico City, Mexico, December 3–4, 2009. Proceedings*, Lecture Notes in Computer Science, pages 9–25. Springer, 2009.
9. Pascal Hitzler, Markus Krötzsch, Bijan Parsia, Peter F. Patel-Schneider, and Sebastian Rudolph, editors. *OWL 2 Web Ontology Language: Primer*. W3C Recommendation 27 October 2009, 2009. Available from `http://www.w3.org/TR/owl2-primer/`.
10. Prateek Jain, Pascal Hitzler, Peter Z. Yeh, Kunal Verma, and Amit P. Sheth. Linked Data is Merely More Data. In Dan Brickley, Vinay K. Chaudhri, Harry Halpin, and Deborah McGuinness, editors, *Linked Data Meets Artificial Intelligence*, pages 82–86. AAAI Press, Menlo Park, CA, 2010.
11. Zoi Kaoudi, Iris Miliaraki, and Manolis Koubarakis. RDFS Reasoning and Query Answering on Top of DHTs. In Amit P. Sheth et al., editors, *Proceedings of the 7th International Semantic Web Conference, ISWC 2008, Karlsruhe, Germany, October 26-30, 2008*, volume 5318 of *Lecture Notes in Computer Science*, pages 499–516. Springer, 2008.
12. Markus Krötzsch, Sebastian Rudolph, and Pascal Hitzler. ELP: Tractable Rules for OWL 2. In Amit P. Sheth et al., editors, *Proceedings of the 7th International Semantic Web Conference, ISWC 2008, Karlsruhe, Germany, October 26-30, 2008*, volume 5318 of *Lecture Notes in Computer Science*, pages 649–664. Springer, 2008.
13. Gergely Lukácsy and Péter Szeredi. Scalable Web Reasoning Using Logic Programming Techniques. In Axel Polleres and Terrance Swift, editors, *Proceedings of the Third International Conference on Web Reasoning and Rule Systems, RR 2009, Chantilly, VA, USA, October 25-26, 2009*, volume 5837 of *Lecture Notes in Computer Science*, pages 102–117. Springer, 2009.
14. Frank Manola and Eric Miller, editors. *Resource Description Framework (RDF). Primer*. W3C Recommendation, 10 February 2004. Available at `http://www.w3.org/TR/rdf-primer/`.
15. Boris Motik, Bernardo Cuenca Grau, Ian Horrocks, Zhe Wu, Achille Fokoue, and Carsten Lutz, editors. *OWL 2 Web Ontology Language: Profiles*. W3C Recommendation, 27 October 2009. Available at `http://www.w3.org/TR/owl2-profiles/`.
16. Eyal Oren, Spyros Kotoulas, George Anadiotis, Ronny Siebes, Annette ten Teije, and Frank van Harmelen. Marvin: Distributed reasoning over large-scale Semantic Web data. *Web Semantics: Science, Services and Agents on the World Wide Web*, 7(4):305–316, 2009.

17. Anne Schlicht and Heiner Stuckenschmidt. Distributed Resolution for Expressive Ontology Networks. In Axel Polleres and Terrance Swift, editors, *Proceedings of the Third International Conference on Web Reasoning and Rule Systems, RR 2009, Chantilly, VA, USA, October 25-26, 2009*, volume 5837 of *Lecture Notes in Computer Science*, pages 87–101. Springer, 2009.
18. Ramakrishna Soma and Viktor K. Prasanna. Parallel inferencing for OWL knowledge bases. In *2008 International Conference on Parallel Processing, ICPP 2008, September 8-12, 2008, Portland, Oregon, USA*, pages 75–82. IEEE Computer Society, 2008.
19. Radhika Sridhar, Padmashree Ravindra, and Kemafor Anyanwu. RAPID: Enabling Scalable Ad-Hoc Analytics on the Semantic Web. In Abraham Bernstein et al., editors, *Proceedings of the 8th International Semantic Web Conference, ISWC 2009, Chantilly, VA, USA, October 25-29, 2009*, volume 5823 of *Lecture Notes in Computer Science*, pages 715–730. Springer, 2009.
20. 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.
21. Jacopo Urbani, Spyros Kotoulas, Jason Maassen, Frank van Harmelen, and Henri Bal. OWL reasoning with WebPIE: calculating the closure of 100 billion triples. In *Proceedings of the 8th Extended Semantic Web Conference (ESWC2010), Heraklion, Greece, May 30–June 3, 2010*. Springer, 2010.
22. Jacopo Urbani, Spyros Kotoulas, Eyal Oren, and Frank van Harmelen. Scalable Distributed Reasoning Using MapReduce. In Abraham Bernstein et al., editors, *Proceedings of the 8th International Semantic Web Conference, ISWC 2009, Chantilly, VA, USA, October 25-29, 2009*, volume 5823 of *Lecture Notes in Computer Science*, pages 634–649. Springer, 2009.
23. Jesse Weaver and James A. Hendler. Parallel materialization of the finite RDFS closure for hundreds of millions of triples. In Abraham Bernstein et al., editors, *The Semantic Web – ISWC 2009, 8th International Semantic Web Conference, ISWC 2009, Chantilly, VA, USA, October 25-29, 2009*, volume 5823 of *Lecture Notes in Computer Science*, pages 682–697. Springer, 2009.