2010

Distributed Reasoning with *EL*++ using MapReduce

Frederick Maier

Raghava Mutharaju

Pascal Hitzler
pascal.hitzler@wright.edu

Distributed Reasoning with $\mathcal{EL}^{++}$ using MapReduce

Frederick Maier and **Raghava Mutharaju** and **Pascal Hitzler** [1]

Technical Report, Kno.e.sis Center, Wright State University, Dayton, Ohio, 2010.

Abstract. It has recently been shown that the MapReduce framework for distributed computation can be used effectively for large-scale RDF Schema reasoning, computing the deductive closure of over a billion RDF triples within a reasonable time [23]. Later work has carried this approach over to OWL Horst [22]. In this paper, we provide a MapReduce algorithm for classifying knowledge bases in the description logic $\mathcal{EL}^{++}$, which is essentially the OWL 2 profile OWL 2 EL. The traditional $\mathcal{EL}^{++}$ classification algorithm is recast into a form compatible with MapReduce, and it is shown how the revised algorithm can be realized within the MapReduce framework. An analysis of the circumstances under which the algorithm can be effectively used is also provided.

1 Introduction

The successful incorporation of large-scale automated reasoning is central to realizing the Semantic Web vision [7]. The assertions comprising Semantic Web ontologies are expressed in a formal knowledge representation language—RDF [13] and OWL [8] being the languages most commonly used—and one of the ultimate goals of the Semantic Web is the development of efficient tools for drawing logical conclusions from these assertions. The primary obstacle to the creation of such tools has been that traditional reasoning algorithms do not scale particularly well, and this makes them incapable of handling the number of assertions contained in large ontologies. This is due essentially to the inherent complexity of the underlying reasoning tasks. *E.g.*, the worst-case complexity of classifying $\mathcal{SROIQ}$ ontologies ($\mathcal{SROIQ}$ being the description logic upon which OWL 2 is based) lies in the class `N2ExpTime`. Restricted *profiles* [15] of OWL 2 (including OWL 2 EL, which is essentially the description logic $\mathcal{EL}^{++}$ [2]) *do* exist, and the standard reasoning tasks for these can be solved in polynomial time. Nevertheless, it is unlikely that the current algorithms for even these will scale to the number of assertions predicted to be stored on the Web. *E.g.*, a recent discussion of Linked Open Data [4] estimates that approximately 4.7 billion RDF triples are on the Web, interlinked by 142 million RDF links [4].[2] 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 continued steps toward creating parallel reasoning algorithms for the Semantic Web. Specifically, we present a parallel algorithm for classifying $\mathcal{EL}^{++}$ ontologies using MapReduce, which is a programming model and software framework for distributed processing of data on clusters of machines. In doing so, we follow the lead of [22, 23], where MapReduce was successfully used for computing RDF Schema closure and for reasoning with OWL Horst.

These publications are part of a recent trend in Semantic Web research to explore parallelization of reasoning tasks. Some of the most notable recent developments are the use of the MapReduce framework for RDF [20, 22, 23], using distributed hash tables for RDF Schema [9], the MaRVIN peer-to-peer platform for RDF [16], and the approach in [24] 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 [19, 22], OWL RL [12], distributed resolution for $\mathcal{SHIQ}$ ontology networks [17], and some other preliminary investigations [1, 5].

The remainder of the paper is structured as follows. Section 2 provides an overview of the description logic $\mathcal{EL}^{++}$ and the traditional polynomial-time algorithm for classifying $\mathcal{EL}^{++}$ ontologies. We then briefly recall the MapReduce framework in Section 3. Modifications to the $\mathcal{EL}^{++}$ classification algorithm, needed in order to make the algorithm parallelizable using MapReduce, are then presented in Section 4. In Section 5 we show how to cast the revised algorithm into MapReduce, and a theoretical discussion is given in Section 6. Section 7 concludes with directions for future research.

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

2 Classifying $\mathcal{EL}^{++}$ Ontologies

The description logic $\mathcal{EL}^{++}$ [2] (which could also be called $\mathcal{ELRO}$ following more standard nomenclature [3]) is the description logic underlying the OWL 2 profile OWL 2 EL. It appeared on the scene a few years ago [2], and received widespread attention due to the fact that it was the first known polynomial time description logic which found application in a commercial setting: The commercial SNOMED

[1] Kno.e.sis Center, Wright State University, Dayton, Ohio

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

ontology,[3] with about 300,000 axioms, falls within $\mathcal{EL}^{++}$, and with the advent of the algorithm it was for the first time possible to formally classify this ontology [18]. This success spawned considerable and still expanding interest in polynomial-time description logics.

Concepts in $\mathcal{EL}^{++}$ are formed according to the grammar

$$C ::= A \mid \top \mid \bot \mid \{a\} \mid C \sqcap D \mid \exists r.C$$

where A ranges over concept names, a over individual names, r over role names, and C, D over (possibly complex) concepts. An $\mathcal{EL}^{++}$ *ontology* (or *constraint box*) 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.

$\mathcal{EL}^{++}$ also allows what are called concrete domains in concept descriptions. However, since we are interested here in the purely logical fragment of $\mathcal{EL}^{++}$, we omit a discussion of them. Given this, $\mathcal{EL}^{++}$ is essentially $\mathcal{EL}^{+}$ extended with a bottom concept, $\bot$, and *nominals* $\{a\}$, where $\{a\}$ is the class containing only individual a.

The formal semantics of $\mathcal{EL}^{++}$ is given in model-theoretic terms, following the standard approach employed for description logics, which can, *e.g.*, be found in [3]. We refrain from repeating this here. The OWL 2 profile OWL 2 EL [15] is a syntactic variant of $\mathcal{EL}^{++}$ with a few minor additions, which we ignore for the sake of clarity.

The primary reasoning task for $\mathcal{EL}^{++}$ ontologies is *classification*—the computation of the complete subsumption hierarchy between all concept names and nominals occurring in the ontology. Other tasks, such as concept satisfiability and the consistency of so-called ABox assertions, are reducible to classification. The classification algorithm for $\mathcal{EL}^{++}$ is shown in Figure 1. It requires the input ontology $\mathcal{O}$ to be in *normal form*, where all concept inclusions have one of the forms

$$C_1 \sqsubseteq D \mid C_1 \sqsubseteq \exists r.C_2 \mid C_1 \sqcap C_2 \sqsubseteq D \mid \exists r.C_1 \sqsubseteq D$$

and all role inclusions have the form $r \sqsubseteq s$ or else $r \circ s \sqsubseteq t$. Each C_1 and C_2 must be in the set $BC_{\mathcal{O}}$, where $BC_{\mathcal{O}}$ is the set consisting of $\top$, each concept name, and each nominal. Each D must be in the set $BC_{\mathcal{O}}^{\bot}$, where $BC_{\mathcal{O}}^{\bot}$ is $BC_{\mathcal{O}} \cup \{\bot\}$.

The transformation into normal form can be done in linear time [2], and the process potentially introduces concept names not found in the original ontology. The normalized ontology is a *conservative extension* of the original, in the sense that every model of the original can be extended into one for the normalized ontology. In the following, we assume that all of the ontologies we deal with are already in normal form.

The classification algorithm makes use of two mappings S and R, where $S(X)$ maps each element $X \in BC_{\mathcal{O}}$ to a subset of $BC_{\mathcal{O}}^{\bot}$, and $R(r)$ maps each role name r to a binary relation over $BC_{\mathcal{O}}^{\bot}$. Intuitively, $B \in S(A)$ implies $A \sqsubseteq B$, while $(A, B) \in R(r)$ implies $A \sqsubseteq \exists r.B$. In the algorithm, for each element $X \in BC_{\mathcal{O}}$, $S(X)$ is initialized to contain just $\{X, \top\}$, and $R(r)$, for each role name r, is initialized to $\varnothing$. The sets $S(X)$ and $R(r)$ are then extended by applying the completion rules shown in Figure 1 until no rule is applicable.

The condition $C \leadsto_R D$ in rule **C6** is used to indicate that there are concepts $C_1, \ldots, C_k \in BC_{\mathcal{O}}$ such that

1. $C_1 \in \{C, \top\} \cup \{\{b\} \mid b \text{ an individual}\}$,
2. $(C_j, C_{j+1}) \in R(r_j)$ for some role name r_j $(1 \leq j < k)$, and
3. $C_k = D$.

This condition differs slightly from the one presented in [2], which is incorrect.[4] We follow the corrected version from [10].

The classification algorithm is guaranteed to terminate in polynomial time relative to the size of the input ontology, and it is also sound and complete: For all class names A and B, $A \sqsubseteq B$ if and only if either $S(A) \cap \{B, \bot\} \neq \varnothing$, or else there is an $\{a\} \in BC_{\mathcal{O}}$ such that $\bot \in S(\{a\})$.

C1	If $C' \in S(C)$, $C' \sqsubseteq D \in \mathcal{O}$, and $D \notin S(C)$, then $S(C) := S(C) \cup \{D\}$
C2	If $C_1, C_2 \in S(C)$, $C_1 \sqcap C_2 \sqsubseteq D \in \mathcal{O}$, and $D \notin S(C)$, then $S(C) := S(C) \cup \{D\}$
C3	If $C' \in S(C)$, $C' \sqsubseteq \exists r.D \in \mathcal{O}$, and $(C, D) \notin R(r)$, then $R(r) := R(r) \cup \{(C, D)\}$
C4	If $(C, D) \in R(r)$, $D' \in S(D)$, $\exists r.D' \sqsubseteq E \in \mathcal{O}$, and $E \notin S(C)$, then $S(C) := S(C) \cup \{E\}$
C5	If $(C, D) \in R(r)$, $\bot \in S(D)$, and $\bot \notin S(C)$, then $S(C) := S(C) \cup \{\bot\}$
C6	If $\{a\} \in S(C) \cap S(D)$, $C \leadsto_R D$, and $S(D) \not\subseteq S(C)$, then $S(C) := S(C) \cup S(D)$
C10	If $(C, D) \in R(r)$, $r \sqsubseteq s \in \mathcal{O}$, and $(C, D) \notin R(s)$, then $R(s) := R(s) \cup \{(C, D)\}$
C11	If $(C, D) \in R(r_1)$, $(D, E) \in R(r_2)$, $r_1 \circ r_2 \sqsubseteq r_3 \in \mathcal{O}$, and $(C, E) \notin R(r_3)$, then $R(r_3) := R(r_3) \cup \{(C, E)\}$

Figure 1. The classification completion rules for $\mathcal{EL}^{++}$. Rules **C7**–**C9** in [2] deal with concrete domains and so are not included.

3 MapReduce

MapReduce is a programming model for distributed processing of large amounts of data on clusters of machines (called *nodes*) [6]. The input is manipulated in two stages (called the *map phase* and *reduce phase*, respectively), and both input and output are in the form of *key-value* pairs. A central Master node divides the data into chunks, and each chunk is assigned to an idle Map node. A domain specific map function processes the data, producing intermediate key-value pairs. These are typically written to a local disk, and their locations are forwarded to Reduce nodes via the Master. Each Reduce node then processes the values indexed by a given key, producing zero or more output pairs.

As an example of how MapReduce could be used in the context of $\mathcal{EL}^{++}$ reasoning, consider rule **C1** in Figure 1. Ignoring the constraint ensuring that no duplicates are added to $S(C)$, the rule has only two preconditions: $C' \in S(C)$ and $C' \sqsubseteq D \in \mathcal{O}$. Both of these make use of a common element C', and this can be used as a key. When implemented in the MapReduce framework, the map function is used to identify the preconditions relevant to **C1** (indexing them using the key), and the reduce function processes the identified elements, completing the application of the rule. In the case of

[3] http://www.ihtsdo.org/snomed-ct/

[4] The algorithm in [2] omits the case where $C_1 = \top$. It is clear, however, that this case must also be considered.

C1, this means that each relevant $S(C)$ is updated appropriately.

There are several prominent implementations of the MapReduce model.[5] Using them, developers need only define the map and reduce functions, leaving lower level and administrative tasks to general purpose components of the system.

4 Making the $\mathcal{EL}^{++}$ Algorithm Parallelizable

To use MapReduce with $\mathcal{EL}^{++}$, we follow the lead of [23], 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 [22], the authors extend their approach to OWL Horst [21], 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.

Certain rules of the original $\mathcal{EL}^{++}$ classification algorithm are already amenable to implementation in a MapReduce framework, in the sense that the preconditions of the rules possess a common element that can serve as key—see the example for rule **C1** in Section 3. Rules **C2**, **C4**, **C6**, and **C11**, however, cannot be used directly, and each must first be split into multiple rules of a more suitable form. This is explained below.

In the revised algorithm, new structures are used in addition to R and S. Specifically, the functions P and L map each element of $BC_{\mathcal{O}}$ to a subset of $BC_{\mathcal{O}}^{\bot} \times BC_{\mathcal{O}}^{\bot}$, while H maps each element of $BC_{\mathcal{O}}$ to a subset of $BC_{\mathcal{O}}^{\bot}$. N is a subset of $BC_{\mathcal{O}}^{\bot}$, while J is a subset of $BC_{\mathcal{O}}^{\bot} \times BC_{\mathcal{O}}^{\bot}$. Intuitively, $(C_1, D) \in P(C_2)$ means $C_1 \sqcap C_2 \sqsubseteq D$. L is used as a counterpart to R, the difference being that L represents axioms with the existential restriction on the left-hand side: $(C, D) \in L(r)$ implies $\exists r.C \sqsubseteq D$. The set J is used to indicate that some nominal is a superclass of two concepts: $(C, D) \in J$ implies $\{o\} \in S(C)$ and $\{o\} \in S(D)$ for some nominal $\{o\}$. $D \in N$ indicates that a sequence of the form

$$C_1 \sqsubseteq \exists r_1.C_2, C_2 \sqsubseteq \exists r_2.C_3, \ldots, C_{k-1} \sqsubseteq \exists r_{k-1}.D$$

holds, where C_1 is either $\top$ or else some nominal $\{o\}$. $D \in H(C)$ implies that a similar sequence holds, with $C_1 = C$.

The reformulation of the algorithm also requires that R be altered to map binary role chains $r \circ s$ (in addition to simple roles) to a subset of $BC_{\mathcal{O}}^{\bot} \times BC_{\mathcal{O}}^{\bot}$. The intuition remains the same, however: $(C, D) \in R(r \circ s)$ implies $C \sqsubseteq \exists(r \circ s).D$. The latter expression is not grammatically correct in $\mathcal{EL}^{++}$, but it is semantically unproblematic, and furthermore it causes no problems in the algorithm.

During the initialization of the algorithm, for each $C \in BC_{\mathcal{O}}$, $S(C)$ is set to $\{C, \top\}$, while $P(C)$ and $H(C)$ are set to $\varnothing$. Similarly, J, N are initialized to $\varnothing$, as is each $R(r)$, $R(r \circ s)$ and $L(r)$, where r and s are simple role names. Applying the completion rules adds to these sets.

The completion rules of the revised algorithm are shown in Figure 2. With the exception of the removal of the con-

	Completion Rule	Key
C1	If $C' \in S(C)$ and $C' \sqsubseteq D \in \mathcal{O}$ then $S(C) := S(C) \cup \{D\}$	C'
C2-1	If $C_1 \in S(C)$ and $C_1 \sqcap C_2 \sqsubseteq D \in \mathcal{O}$ then $P(C) := P(C) \cup \{(C_2, D)\}$	C_1
C2-2	If $C_2 \in S(C)$ and $(C_2, D) \in P(C)$ then $S(C) := S(C) \cup \{D\}$	C_2
C3	If $C' \in S(C)$, $C' \sqsubseteq \exists r.D \in \mathcal{O}$ then $R(r) := R(r) \cup \{(C, D)\}$	C'
C4-1	If $D' \in S(D)$ and $\exists r.D' \sqsubseteq E \in \mathcal{O}$ then $L(r) := L(r) \cup \{(D, E)\}$	D'
C4-2	If $(C, D) \in R(r)$ and $(D, E) \in L(r)$ then $S(C) := S(C) \cup \{E\}$	r
C5	If $(C, D) \in R(r)$, $\bot \in S(D)$ then $S(C) := S(C) \cup \{\bot\}$	D
C6-1	If $\{a\} \in S(C)$ and $\{a\} \in S(D)$ then $J := J \cup \{(C, D)\}$	a
C6-2-1	If $(C, D) \in R(r)$ then $H(C) := H(C) \cup \{D\}$	C
C6-2-2	If $(C, D) \in R(r)$ and $C \in H(E)$ then $H(E) := H(E) \cup \{D\}$	C
C6-2-3	If $(C, D) \in J$ and $D \in H(C)$, then $S(C) := S(C) \cup S(D)$	C
C6-3-1	If $(\{b\}, D) \in R(r)$ or $(\top, D) \in R(r)$ then $N := N \cup \{D\}$	D
C6-3-2	If $(C, D) \in R(r)$ and $C \in N$ then $N := N \cup \{D\}$	C
C6-3-3	If $(C, D) \in J$ and $D \in N$, then $S(C) := S(C) \cup S(D)$	D
C10	If $(C, D) \in R(r)$, $r \sqsubseteq s \in \mathcal{O}$ then $R(s) := R(s) \cup \{(C, D)\}$	r
C11-1	If $(C, D) \in R(r_1)$, and $(D, E) \in R(r_2)$ then $R(r_1 \circ r_2) := R(r_1 \circ r_2) \cup \{(C, E)\}$	D

Figure 2. Revised $\mathcal{EL}^{++}$ algorithm. In C10, r is allowed to be of the form $r_1 \circ r_2$. The keys are used in the MapReduce algorithm presented in Section 5.

straint preventing duplicate additions (see the below discussion), rules **C1**, **C3**, and **C5** are left unchanged. Rule **C10** is altered to apply to both simple roles and binary role chains. This modification is semantically sound, and it does not affect the correctness of the algorithm.

The changes to the other rules are more substantial. The action of **C2** is simulated by applying **C2-1** and then **C2-2**. Rule **C4** is simulated by subsequent applications of **C4-1** and **C4-2**, and rule **C11** is simulated by **C11-1** and **C10**. *E.g.*, if $(C, D) \in R(r)$ and $(D, E) \in R(s)$, then **C11-1** yields $(C, E) \in R(r \circ s)$. If $r \circ s \sqsubseteq t$ is an axiom of $\mathcal{O}$, then since it matches the template for **C10**, it follows that $(C, E) \in R(t)$.

The rules replacing **C6** are more difficult to understand. Rules **C6-2-1** and **C6-2-2**, and also **C6-3-1** and **C6-3-2**, are used to explicitly generate the sets $C_1, C_2, \ldots, C_k$ establishing $C \leadsto_R D$. Each pair of rules covers a distinct way in which $C \leadsto_R D$ may be established.

Specifically, **C6-2-1** and **C6-2-2** cover the case where $C_1 = C$. The sequence C_1, C_2, …, C_k is begun using rule **C6-2-1** and extended with repeated applications of **C6-2-2**. The presence of C_i in $H(C)$ implies that $(C, C_2) \in R(r)$,

[5] Hadoop (`http://hadoop.apache.org/`) is a popular Java implementation.

$(C_2, C_3) \in R(r_2)$, ..., $(C_{i-1}, C_i) \in R(r_{i-1})$ all hold, and this is sufficient to establish $C \leadsto_R C_i$.

As a concrete example, consider the below set of GCIs:

$$\mathrm{A} \sqsubseteq \exists r_1.\mathrm{B} \qquad \mathrm{B} \sqsubseteq \exists r_2.\mathrm{C}$$
$$\mathrm{C} \sqsubseteq \exists r_3.\mathrm{D} \qquad \mathrm{D} \sqsubseteq \exists r_4.\mathrm{E}$$

Given the way $S(X)$ is initialized, it must be that $X \in S(X)$ for each $X \in BC_{\mathcal{O}}$. As this is so, Rule **C3** yields $(A, B) \in R(r_1)$, $(B, C) \in R(r_2)$, $(C, D) \in R(r_3)$, and $(D, E) \in R(r_4)$. Given this, it is clear that $A \leadsto_R E$ holds (according to the original definition). Using the revised rules, **C6-2-1** yields $B \in H(A)$. Iterative applications of **C6-2-2** yield $\{B, C, D, E\} \subseteq H(A)$, and so for any element $X \in H(A)$ (including E), $A \leadsto_R X$ holds.

In a similar way, rules **C6-3-1** and **C6-3-2** are used to cover the case where $C \leadsto_R D$ holds and the initial concept C_1 in the sequence C_1, C_2, ..., C_k is a nominal $\{b\}$ or else $\top$. As with $H(C)$, for each element $X \in N$, $C \leadsto_R X$ holds. Observe that the particular nature of C_1 in this case is unimportant (and it need not be related to C). This explains why N is a set and not a function from $BC_{\mathcal{O}}$ to $BC_{\mathcal{O}}^{\perp}$.

In rule **C6-1**, $(C, D) \in J$ is used solely to encode that $\{a\} \in S(C) \cap S(D)$ holds for some individual a. $(C, D) \in J$ is used together with the results of **C6-2-1** and **C6-2-2** (alternatively, **C6-3-1** and **C6-3-2**) as input into **C6-2-3** (alternatively, **C6-3-3**). Applying **C6-2-3** or **C6-3-3** completes the simulation of **C6** in the original algorithm.

Each addition to $S(C)$, $P(C)$, $R(r)$, $L(r)$, $H(C)$, J and N is entailed by the knowledge base $\mathcal{O}$, and so the rules are sound. The additions also do not cause any problems with respect to termination, since there is a finite upper bound on the number of additions to any of the sets. The revised algorithm terminates if no application of any of the rules causes further additions. This alteration from the original scheme was made because it makes the algorithm more compatible with the MapReduce format. It is easy to see that the new termination condition is equivalent to the additional constraints used in the original rules. Indeed, using a straightforward inductive argument, it is easy to show that the revised algorithm yields the same results as the original, and so it is sound, complete, and terminating. It is also of polynomial worst-case complexity in the size of the input ontology; this can be shown along the lines of argument presented in [2].

5 Parallelization using MapReduce

The rules are now in a form amenable to implementation using the MapReduce framework. In the discussion below, we will slightly abuse terminology by referring to all expressions of the form $D \in S(C)$ (and $(C, D) \in P(E)$, $(C, D) \in R(r)$, $C \in N$, *etc.*), in addition to the original elements of $\mathcal{O}$, as *axioms*.

The general strategy of the algorithm is as follows: The completion rules are applied in an iterative manner, with one rule being applied in a given iteration and the results of previous iterations being reused. Newly generated output is added to a database storing the contents of $\mathcal{O}$, S, P, R, H, N, and J. Within a given iteration, the axioms obtained thus far are divided into multiple chunks. Each chunk is distributed to different computing nodes, which first act as map nodes and then as reduce nodes. Each map-reduce cycle results in the parallel application of one of the completion rules. In the map phase, based on the rule chosen to be applied, the axioms satisfying any of the preconditions of the rule are found, and intermediate $\langle key, value\rangle$ pairs are generated. In each pair, *key* is a concept or relation common to each precondition of the rule (as indicated in Figure 2), while *value* is the axiom itself (which matches the precondition). In the reduce phase, all axioms belonging to the same key are collected from different nodes and the conclusions of the completion rule are computed, taking all valid combinations of axioms into account. All outputs are stored in a database without duplication. Iterations continue until a fixpoint is reached.

We refrain from giving descriptions of the map and reduce functions for all of the revised completion rules. The functions for **C6-3-2** and **C6-3-3** are given in Figures 3 and 4, however, and we will walk through an example using **C6-3-3**. The other rules are handled in an analogous manner, using the keys listed in Figure 2.

$$\{\mathrm{b}\} \sqsubseteq \exists r_1.\mathrm{C}_2 \qquad \mathrm{C} \sqsubseteq \{\mathrm{a}\}$$
$$\mathrm{C}_2 \sqsubseteq \exists r_2.\mathrm{C}_3 \qquad \mathrm{D} \sqsubseteq \{\mathrm{a}\}$$
$$\mathrm{C}_3 \sqsubseteq \exists r_3.\mathrm{D} \qquad \mathrm{F} \sqsubseteq \{\mathrm{a}\}$$
$$\mathrm{D} \sqsubseteq \mathrm{E} \qquad \mathrm{D} \sqsubseteq \mathrm{G}$$

Given the above inclusion axioms, **C1** can be used to yield the following: $\{\mathrm{a}\} \in S(\mathrm{C})$, $\{\mathrm{a}\} \in S(\mathrm{D})$, $\{\mathrm{a}\} \in S(\mathrm{F})$, $\mathrm{E} \in S(\mathrm{D})$, and $\mathrm{G} \in S(\mathrm{D})$. **C3** yields $(\{\mathrm{b}\}, \mathrm{C}_2) \in R(r_1)$, $(\mathrm{C}_2, \mathrm{C}_3) \in R(r_2)$, and $(\mathrm{C}_3, \mathrm{D}) \in R(r_3)$. **C6-1** yields the following pairs in J: (C, D), (C, F), (D, F), (D, C), (F, C), (F, D).[6] Applying **C6-3-1** and **C6-3-2** repeatedly yields $\mathrm{C}_2 \in N$, $\mathrm{C}_3 \in N$, and $\mathrm{D} \in N$. Given all of these as input, **C6-3-3** can now be applied. In the map phase, the following intermediate key-value pairs are produced.

$$\langle \mathrm{C}_2, \mathrm{C}_2 \in N\rangle \qquad \langle \mathrm{C}_3, \mathrm{C}_3 \in N\rangle \qquad \langle \mathrm{D}, \mathrm{D} \in N\rangle$$
$$\langle \mathrm{D}, (\mathrm{C}, \mathrm{D}) \in J\rangle \qquad \langle \mathrm{C}, (\mathrm{D}, \mathrm{C}) \in J\rangle \qquad \langle \mathrm{F}, (\mathrm{C}, \mathrm{F}) \in J\rangle$$
$$\langle \mathrm{C}, (\mathrm{F}, \mathrm{C}) \in J\rangle \qquad \langle \mathrm{F}, (\mathrm{D}, \mathrm{F}) \in J\rangle \qquad \langle \mathrm{D}, (\mathrm{F}, \mathrm{D}) \in J\rangle$$

Observe that $\mathrm{D} \in N$, $(\mathrm{C}, \mathrm{D}) \in J$, and $(\mathrm{F}, \mathrm{D}) \in J$ are all indexed by the same key (D), and so will be processed by the same reduce node. Every combination of values passed to a reduce node will be tried. When this occurs, $(\mathrm{C}, \mathrm{D}) \in J$ and $\mathrm{D} \in N$ cause $D_i \in S(\mathrm{C})$ to be generated for each $D_i \in S(\mathrm{D})$. Similarly, $(\mathrm{F}, \mathrm{D}) \in J$ and $\mathrm{D} \in N$ cause $D_i \in S(\mathrm{F})$ to be generated for each $D_i \in S(\mathrm{D})$. In particular, since $\mathrm{E} \in S(\mathrm{D})$ and $\mathrm{G} \in S(\mathrm{D})$, it follows that $\mathrm{E} \in S(\mathrm{C})$, $\mathrm{G} \in S(\mathrm{C})$, $\mathrm{E} \in S(\mathrm{F})$, and $\mathrm{G} \in S(\mathrm{F})$ are all generated. Note that the values taken from $S(\mathrm{D})$ and added to $S(\mathrm{C})$ and $S(\mathrm{F})$ are stored in a database and not included in the values passed to the reduce node.

6 Theoretical Analysis

Reasoning algorithms are usually not naturally parallelizable. Furthermore, it is known that the worst-case complexity of P-complete problems (such as $\mathcal{EL}^{++}$ ontology classification) cannot gain from parallelization [14]. Nevertheless, the ability to be able to run several computations in parallel is bound to

[6] In **C6-1**, we assume no pair (C, D) is added to J when $\mathrm{C} = \mathrm{D}$.

```
map(key, value)
//key: a line number (ignored); value: an axiom
{
    if(value == $(C,D) \in R(r)$)
        emit($\langle C$, $(C,D) \in R(r)\rangle$);
    else if(value == $C \in N$)
        emit($\langle C$, $C \in N\rangle$);
}
reduce(key, iterator values)
//key: a concept name or nominal; values: (axioms)
{
    for each $v_1$ in values
        for each $v_2$ in values
        {
            if($v_1$ == $(C,D) \in R(r)$ and $v_2$ == $C \in N$
                emit($D \in N$);
        }
}
```

Figure 3. MapReduce algorithm for **C6-3-2**. The input of the map function is an axiom, taken from $\mathcal{O}$, S, P, R, H, J, or N. Key-value pairs are generated in the map phase, and these serve as input in the reduce phase. For a given key, every possible combination of values is examined to determine whether **C6-3-2** is applicable. Ultimately, a list of axioms is produced as output.

```
map(key, value)
//key: a line number (ignored); value: an axiom
{
    if(value == $(C,D) \in J$)
        emit($\langle D$, $(C,D) \in J\rangle$);
    else if(value == $C \in N$)
        emit($\langle C$, $C \in N\rangle$);
}
reduce(key, iterator values)
//key: a concept name or nominal; values: (axioms)
{
    for each $v_1$ in values
        for each $v_2$ in values
        {
            if($v_1$ == $(C,D) \in J$ and $v_2$ == $D \in N$
                for each $E \in S(D)$ emit($E \in S(C)$);
        }
}
```

Figure 4. MapReduce algorithm for **C6-3-3**.

have a significant impact on performance, at least for suitable knowledge bases. At the same time, however, this impact cannot be more than a decrease in the runtime by k, where k is the number of nodes employed in parallel.

In this section, we show that in the worst case we indeed gain nothing at all. In the best case, however, we gain optimal speed-up. This indicates that the gain from parallelization depends substantially on the structure of the knowledge base. This insight is encouraging since it suggests that application scenarios for parallelization of $\mathcal{EL}^{++}$ knowledge base classification exist.

In the following, we use n to indicate the size of the input knowledge base, and k to indicate the number of nodes for parallelization. For our qualitative discussion, it is useful to assume that k is of the order of magnitude of n, or to simply assume $n \leq k$, as it yields an order of magnitude for the impact of the parallelization.

For the best-case scenario, consider a knowledge base consisting of n axioms $A_i \sqsubseteq A_{i+1}$, where $1 \leq i < n$. For the complete classification, there are $n^2 - n - 1$ new inclusions to be computed, which requires $n^2 - n - 1 \in O(n^2)$ executions of rule **C1**. At the same time, however the same computation can be achieved by making only $\frac{n}{2} \in O(n)$ calls to the parallel MapReduce algorithm for the **C1** rule. Note that this holds even if $k = 1$, which is misleading, since each MapReduce call would in this case be much more expensive (order of magnitude: n-times as expensive) than a call to the sequential **C1** rule. However, assuming $k \geq n$ as discussed above, the speed-up is indeed in the order of magnitude of decreasing a quadratic time to a linear time (assuming k grows with n). Note that this is an extreme case which potentially allows for massive parallelization through MapReduce—and in this case having n nodes really pays off, although the assumption is unrealistic for large n. Realistic cases will usually not be as extreme.

On the other end of the spectrum is the worst-case scenario, which does not allow any significant speed-up. Consider the knowledge base consisting of the axioms $A_i \sqsubseteq \exists r.A_{i+1}$, for $1 \leq i < n$, and $A_n \sqsubseteq \bot$. If we assume that the initialization of the R function according to rule **C3** has already been performed before the actual running of the algorithm (which is not an unreasonable assumption), the sequential algorithm requires $n - 1$ calls of rule **C5** to arrive at the classification (namely, $A_i \sqsubseteq \bot$ for all i). The parallel algorithm, however, also requires $n - 1$ calls to the MapReduce version of **C5**, since each call results in only one new axiom.

We have ignored the general overhead which MapReduce implementations generally have. In Hadoop, for example, there is a significant overhead at start-up, which indicates that an overall gain will only be made if the input knowledge base is very large. We still must perform experiments to determine what size and what type of knowledge base is most suitable for our approach, but indications are that we will require knowledge bases which are of a size not currently available, unless artificially created. However, realistic knowledge bases will be required for a final verdict on the usability of any reasoning approach.

7 Conclusion and Future Work

The amount of information stored on the Web using formal knowledge representation languages is already of considerable size, and it will only increase as time progresses [4]. As this is so, there is an ever increasing need for parallelizable reasoning algorithms. In this paper, following the lead of existing work on scalable implementations of RDF Schema closure, we have provided a MapReduce algorithm for classifying $\mathcal{EL}^{++}$ ontologies. While a formal guarantee of a performance increase over sequential algorithms cannot be provided, we believe that the parallel approach is scalable and for some large ontologies will significantly reduce the time needed to compute the subsumption hierarchy.

We currently have a crude prototype implementation of our approach making use of Hadoop and the infrastructure at our institute. Toy examples work out fine. However, we have not been able to evaluate it—and measure runtimes—on large data sets yet. The experiences reported in [22, 23] on using MapReduce for RDF Schema indicate that optimizations, in particular concerning the choice of which completion rule is applied next, will have a profound effect on performance. This is, of course, to be expected. Since the output of some completion rules is used exclusively by certain other rules, the order in which rules are applied is significant. It is not beneficial to attempt **C2-2**, for instance, unless **C2-1** has successfully produced new output. We envision that in the completed implementation, the master node monitors the outputs obtained during each iteration and dynamically schedules the completion rules to apply next. Our current research is dedicated in part to studying which scheduling algorithms and heuristics may be applied to yield good results.

The results presented in [22, 23] also indicate that the MapReduce approach requires rather large data sets to show a pay-off in terms of performance. In our case, this will likely require the generation of artificial data sets for initial experiments. While there are currently few if any natural data sets that can be used in order to fully evaluate the framework, we do not believe that this will always be the case—at least in part, the sizes of knowledge bases used in practice are restricted by the capabilities of the reasoning tools processing them.

We furthermore consider this line of work to be only the starting point for investigations into more expressive languages, including ELP [11] and OWL 2 DL.

Acknowledgements. This project was supported by the Wright State University Research Council.

REFERENCES

[1] M. Aslani and V. Haarslev. Towards Parallel Classification of TBoxes. In F. Baader, C. Lutz, and B. 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] F. Baader, S. Brandt, and C. Lutz. Pushing the EL envelope. In *Proc. 19th Int. Joint Conf. on Artificial Intelligence (IJCAI-05)*, Edinburgh, UK, 2005. Morgan-Kaufmann Publishers.

[3] F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, 2007.

[4] C. Bizer, T. Heath, and T. Berners-Lee. Linked data – the story so far. *International Journal on Semantic Web and Information Systems*, 5(3):1–22, 2009.

[5] J. Bock. Parallel Computation Techniques for Ontology Reasoning. In A.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.

[6] J. Dean and S. Ghemawat. MapReduce: Simplified data processing on large clusters. In *Proceedings of the USENIX Symposium on Operating Systems Design & Implementation (OSDI)*, pages 137–150, 2004.

[7] P. Hitzler. Towards reasoning pragmatics. In K. Janowicz, M. Raubal, and S. 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.

[8] P. Hitzler, M. Krötzsch, B. Parsia, P.F. Patel-Schneider, and S. Rudolph, editors. *OWL 2 Web Ontology Language: Primer.* W3C Recommendation 27 October 2009, 2009. Available from `http://www.w3.org/TR/owl2-primer/`.

[9] Z. Kaoudi, I. Miliaraki, and M.s Koubarakis. RDFS Reasoning and Query Answering on Top of DHTs. In A.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.

[10] M. Krötzsch, S. Rudolph, and P. Hitzler. Conjunctive Queries for a Tractable Fragment of OWL 1.1. In K. Aberer et al., editors, *Proceedings of the 6th International Semantic Web Conference (ISWC 2007), Busan, Korea, November 11-15, 2007*, volume 4825 of *LNCS*, pages 310–323. Springer, 2007.

[11] M. Krötzsch, S. Rudolph, and P. Hitzler. ELP: Tractable Rules for OWL 2. In A.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.

[12] G. Lukácsy and P. Szeredi. Scalable Web Reasoning Using Logic Programming Techniques. In A. Polleres and T. 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.

[13] F. Manola and E. Miller, editors. *Resource Description Framework (RDF). Primer.* W3C Recommendation, 10 February 2004. Available at `http://www.w3.org/TR/rdf-primer/`.

[14] S. Miyano. Parallel complexity and P-complete problems. In ICOT, editor, *Proceedings of the International Conference On Fifth Generation Computer Systems*, pages 532–541, 1988.

[15] B. Motik, B. Cuenca Grau, I. Horrocks, Z. Wu, A. Fokoue, and C. Lutz, editors. *OWL 2 Web Ontology Language: Profiles.* W3C Recommendation, 27 October 2009. Available at `http://www.w3.org/TR/owl2-profiles/`.

[16] E. Oren, S. Kotoulas, G. Anadiotis, R. Siebes, A. ten Teije, and F. 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] A. Schlicht and H. Stuckenschmidt. Distributed Resolution for Expressive Ontology Networks. In A. Polleres and T. 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] S. Schulz, B. Suntisrivaraporn, F. Baader, and M. Boeker. SNOMED reaching its adolescence: Ontologists' and logicians' health check. *International Journal of Medical Informatics*, 78(Supplement 1):S86–S94, 2009.

[19] R. Soma and V.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.

[20] R. Sridhar, P. Ravindra, and K. Anyanwu. RAPID: Enabling Scalable Ad-Hoc Analytics on the Semantic Web. In A. 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.

[21] H. 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.

[22] J. Urbani, S. Kotoulas, J. Maassen, F. van Harmelen, and H. Bal. OWL reasoning with MapReduce: calculating the

closure of 100 billion triples. Submitted.

[23] J. Urbani, S. Kotoulas, E. Oren, and F. van Harmelen. Scalable Distributed Reasoning Using MapReduce. In A. 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.

[24] J. Weaver and J. A. Hendler. Parallel materialization of the finite RDFS closure for hundreds of millions of triples. In A. Bernstein et al., editors, *The Semantic Web – ISWC 2009, 8th International Semantic Web Conference, ISWC 2009, Chantilly, VA, USA, October 25-29, 2009. Proceedings*, volume 5823 of *Lecture Notes in Computer Science*, pages 682–697. Springer, 2009.