Distributed OWL EL Reasoning: The Story So Far

Raghava Mutharaju, Pascal Hitzler, and Prabhaker Mateti

Wright State University, OH, USA.
{mutharaju.2, pascal.hitzler, prabhaker.mateti}@wright.edu

Abstract. Automated generation of axioms from streaming data, such as traffic and text, can result in very large ontologies that single machine reasoners cannot handle. Reasoning with large ontologies requires distributed solutions. Scalable reasoning techniques for RDFS, OWL Horst and OWL 2 RL now exist. For OWL 2 EL, several distributed reasoning approaches have been tried, but are all *perceived* to be inefficient. We analyze this perception. We analyze completion rule based distributed approaches, using different characteristics, such as dependency among the rules, implementation optimizations, how axioms and rules are distributed. We also present a distributed queue approach for the classification of ontologies in description logic $\mathcal{EL}^+$ (fragment of OWL 2 EL).

1 Introduction

The rate at which data is generated is increasing at an alarming rate in this age of Big Data. Data processing techniques should also scale up correspondingly. This also holds true in the case of OWL ontologies and reasoning. Manually constructed ontologies would most likely remain small or medium-sized, in the order of several thousands or up to a few million axioms. Generating axioms automatically from streaming data such as traffic [13] or text [7] can result in very large ontologies. Also, in case of reasoning tasks such as classification, the number of inferred axioms keep increasing until the reasoning task terminates. In some cases the size of the result is 75 times that of the input axioms [24]. This turns out to be problematic for current reasoners in case of very large ontologies. A distributed approach to reasoning not only accommodates large ontologies but also provides more processing power.

While some progress has been made regarding scalable reasoning over RDFS, OWL Horst and OWL 2 RL [22, 23, 21, 20], applying similar techniques to OWL 2 EL turns out to be inefficient. In this paper, we investigate the reasons behind it as well as analyze other distributed approaches to reasoning over ontologies in description logic $\mathcal{EL}^+$, which is a fragment of OWL 2 EL. We also present a distributed version of the reasoning algorithm used in CEL reasoner [5]. The distributed approaches mentioned in [15] are explored in detail here.

Rest of the paper is as follows. Section 2 contains a brief description of $\mathcal{EL}^+$ and classification. In Section 3, three approaches to distributed reasoning are

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 $R(t) := R(t) \cup \{(X, Z)\}$

Table 1. Completion rules for classifying $\mathcal{EL}^+$ ontologies

described. Section 4 offers alternative evaluation strategies. In Section 5, some possible future directions are mentioned and Section 6 contains some related work. We conclude in Section 7.

2 Preliminaries

We briefly introduce the description logic $\mathcal{EL}^+$. Let the concept names be denoted by N_C, role names by N_R and $N_C^\top$ denotes N_C including $\top$. Concepts in $\mathcal{EL}^+$ are formed according to the grammar

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

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

The reasoning task that we consider here is *classification* – the computation of the complete subsumption hierarchy of all concept names occurring in the ontology. Other reasoning tasks such as concept satisfiability can be reduced to classification. Classification is computed using a set of completion rules shown in Table 1. It requires the input ontology $\mathcal{O}$ to be in *normal form*, where all concept inclusions have one of the forms

$$A \sqsubseteq B \mid A_1 \sqcap \ldots \sqcap A_n \sqsubseteq B \mid A \sqsubseteq \exists r.B \mid \exists r.A \sqsubseteq B$$

and all role inclusions have the form $r \sqsubseteq s$ or $r \circ s \sqsubseteq t$. $A, A_1, \ldots, A_n, B \in N_C^\top$ and $r, s, t \in N_R$.

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 rest of the paper, we assume that all of the ontologies we deal with are already in normal form.

The classification rules make use of two mappings S and R, $S\colon N_C^\top \mapsto 2^{N_C^\top}$, $R\colon N_R \mapsto 2^{(N_C^\top \times N_C^\top)}$. Intuitively, $B \in S(A)$ implies $A \sqsubseteq B$, while $(A, B) \in R(r)$ implies $A \sqsubseteq \exists r.B$. Before applying the rules, for each element $X \in N_C^\top$, $S(X)$ is initialized to contain $\{X, \top\}$, 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 Table 1. Here we consider two ways in which these rules are applied to the axioms, which are described in later sections.

Classification of ontologies using the completion rules is guaranteed to terminate in polynomial time relative to the size of the input ontology, and it is also sound and complete. Proofs can be found in [2]. For further background on description logics and how they relate to the Web Ontology Language OWL, please see [3, 11].

In this paper, we consider only completion rule based distributed approaches for classification.

We use the terms node and machine interchangeably throughout the paper.

3 Distributed Classification

We analyze three different distributed approaches to $\mathcal{EL}^+$ classification. Among them two have been published previously.

3.1 Prologue

Before embarking on a parallelization effort, it will be useful to check how amenable it is for parallelization. Note that we are using the term parallelization to mean the following – group of processes co-operating with each other to accomplish a common goal. These processes could either be running on the same machine or on different machines. Here, we are interested in the latter, in which case, there is no shared memory.

Dependency among the completion rules of Table 1 is shown in Figure 1. Every rule is dependent on one or more rules. So, in this case, applying rules to axioms cannot be an embarrassingly parallel computation. When these rules are applied in a distributed environment, some amount of communication is required among the nodes handling the rules which slows down the system.

Contrary to this, in RDFS and OWL Horst, little or no dependency exists among the rules and thus embarrassingly parallel computations are possible [23, 21]. As a result of this, in these cases, linear or sometimes better than linear performance with respect increasing nodes was possible.

3.2 MapReduce Approach

Taking the lead from the application of MapReduce to RDFS and OWL Horst reasoning, an attempt was made in [17, 24] to use it for $\mathcal{EL}^+$ reasoning.

MapReduce is a programming model for distributed processing of data on clusters of machines [8]. MapReduce task consists of two main phases: map

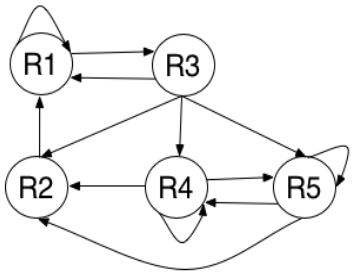

Fig. 1. Dependency among the rules is shown by a directed arrow. A $\rightarrow$ B indicates that the input of A is dependent on the output of B.

and reduce. In map phase, a user-defined function receives a key-value pair and outputs a set of key-value pairs. All the pairs sharing the same key are grouped and passed to the reduce phase. A user-defined reduce function is set up to process the grouped pairs. Completion of a task might involve several such map and reduce cycles. The map and reduce functions can be represented as

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

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

The completion rules in Table 1 are slightly modified to suit the key-value nature of MapReduce approach. The modified rules are given in Table 2. In the map phase, preconditions of the rules are checked and in the reduce phase, conclusion of the rules are computed. For each concept $X \in N_C^\top$ and $r \in N_R$, $S(X)$ is initialized to $\{X, \top\}$ and $R(r), P(X), Q(X)$ are initialized to $\varnothing$. Rules R1, R3 and R5 from Table 1 cannot be dealt with using MapReduce approach, since they have multiple join conditions, which is the reason to split them in the modified rules.

The general strategy used in this approach is given in Algorithm 1. At the end of each iteration, duplicates are removed from S, R, P, Q. Algorithm terminates when there are no changes made by the application of all the rules.

From [24], the performance of this approach and comparison with other reasoners is given in Table 3. The experiments were run on a Hadoop cluster with 8 nodes. Each node has a 2-core, 3GHz processor with 2GB RAM. Although the experiments were conducted on machines with less memory, the evaluation of [17] on machines with more memory (16GB) and larger ontologies has similar performance.

3.2.1 Analysis

1. *Pros* Aspects such as parallelization and fault tolerance are taken care of by the framework.

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 $Q(X) := Q(X) \cup \{\exists r.X \sqsubseteq B\}$	A
$\exists r.A \sqsubseteq B$ for r	**R3-2**	If $(X, Y) \in R(r)$ and $\exists r.Y \sqsubseteq B \in Q(X)$ 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**	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

Table 2. Revised completion rules 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 $s \circ t$.

2. *Cons* There are several disadvantages of this approach. a) Duplicates are generated and an extra step is required to remove the duplicates. b) Since there are dependencies among the rules (Figure 1), MapReduce approach may not best suited. c) In each iteration, the algorithm needs to consider only the newly generated data (compared to last iteration). It is difficult to detect and filter axioms that generate redundant inferences. d) In every iteration, axioms are again reassigned to the machines in the cluster. In the case of RDFS reasoning, schema triples are loaded in-memory and this assignment of schema triples to machines takes place only once. This is not possible in the case of $\mathcal{EL}^+$ reasoning.
3. *Axiom Distribution* Axioms are distributed randomly.
4. *Rule Distribution* In each iteration, all the machines in the cluster, apply the same rule on the local axioms.
5. *Optimizations* Rule R4 is taken care of in the reduce phase of rules R2 and R5. So rule R4 need not be applied again.

3.3 Distributed Queue Approach

Compared to the fixpoint iteration method of rule application, it is claimed that the queue based approach is efficient on a single machine [4]. In this section, we describe a distributed implementation of the queue approach and verify whether the claim also holds true in a distributed setting. First we briefly explain the queue approach on a single machine from [4] and then describe the distributed implementation of it.

For each concept in $N_C^\top$, a queue is assigned. Instead of applying the rules mechanically, in the queue approach, appropriate rules are *triggered* based on the type of entries in the queue. The possible entries in the queue are of the

$S(X) \leftarrow \{X, \top\}$, for each $X \in N_C^\top$
$R(r) \leftarrow \{\}$, for each $r \in N_R$
$P(X) \leftarrow \{\}$, for each $X \in N_C^\top$
$Q(X) \leftarrow \{\}$, for each $X \in N_C^\top$
repeat
  Old.$S(X) \leftarrow S(X)$;
  Old.$R(r) \leftarrow R(r)$;
  Old.$P(X) \leftarrow P(X)$;
  Old.$Q(X) \leftarrow Q(X)$;
  $P(X) := P(X) \cup$ apply R1-1;
  $S(X) := S(X) \cup$ apply R1-2;
  $R(r) := R(r) \cup$ apply R2;
  $Q(X) := Q(X) \cup$ (apply R3-1);
  $S(X) := S(X) \cup$ apply R3-2;
  $R(r) := R(r) \cup$ apply R4;
  $R(r) := R(r) \cup$ apply R5;
until *((Old.$S(X) = S(X)$) and (Old.$R(r) = R(r)$) and (Old.$P(X) = P(X)$) and (Old.$Q(X) = Q(X)$))*;

Algorithm 1: General strategy for applying rules in MapReduce approach

Ontology	#Axioms	ELK	jCEL	Pellet	MR Approach
1-GALEN	90000	2.3	116.2	742.4	6552.5
2-GALEN	178000	5.5	243.7	OOM	11952.5
4-GALEN	352000	11.6	OOM	OOM	19908.3
8-GALEN	703000	OOM	OOM	OOM	38268.7

Table 3. Classification time (in seconds) of MapReduce (MR) approach. OOM indicates Out Of Memory.

form $B_1, \ldots, B_n \rightarrow B'$ and $\exists r.B$ with $B_1, \ldots, B', B \in N_C^\top$ and $r \in N_R$. If $n = 0$, $B_1, \ldots, B_n \rightarrow B'$ is simply written as B'. $\widehat{\mathcal{O}}$ is a mapping from a concept to sets of queue entries as follows.

- if $A_1 \sqcap \ldots \sqcap A_n \sqsubseteq B \in \mathcal{O}$ and $A_i = A$, then $A_1 \sqcap \ldots \sqcap A_{i-1} \sqcap A_{i+1} \sqcap \ldots \sqcap A_n \rightarrow B \in \widehat{\mathcal{O}}(A)$
- if $A \sqsubseteq \exists r.B \in \mathcal{O}$, then $\exists r.B \in \widehat{\mathcal{O}}(A)$
- if $\exists r.A \sqsubseteq B \in \mathcal{O}$, then $B \in \widehat{\mathcal{O}}(\exists r.A)$

For each concept $A \in N_C^\top$, queue(A) is initialized to $\widehat{\mathcal{O}}(A) \cup \widehat{\mathcal{O}}(\top)$. For each queue, an entry is fetched and Algorithm 2 is applied. The procedure in Algorithm 3 is called by process(A, X) whenever a new pair of (A, B) is added to $R(r)$. Note that, for any concept A, $\widehat{\mathcal{O}}(A)$ does not change during the application of the two procedures (process, process-new-edge); $S(A)$, queue(A) and $R(r)$ keep changing.

In the distributed setup, axioms are represented as key-value pairs as shown in Table 4. For axioms of the form $A_1 \sqcap \ldots \sqcap A_n \sqsubseteq B$, for each A_i (key) in the conjunct, $(A_1, \ldots, A_{i-1}, A_{i+1}, \ldots, A_n, B)$ is associated as its value.

if $X = B_1, \ldots, B_n \rightarrow B'$ *and* $B' \notin S(A)$ **then**
  if $B_1, \ldots, B_n \in S(A)$ **then**
    continue with $X \leftarrow B'$;
  else
    return;
if X *is a concept name and* $X \notin S(A)$ **then**
  $S(A) \leftarrow S(A) \cup \{X\}$;
  queue$(A) \leftarrow$ queue$(A) \cup \widehat{\mathcal{O}}(X)$;
  forall the *concept names* B *and role names* r *with* $(B, A) \in R(r)$ **do**
    queue$(B) \leftarrow$ queue$(B) \cup \widehat{\mathcal{O}}(\exists r.X)$;
if X *is an existential restriction* $\exists r.B$ *and* $(A, B) \notin R(r)$ **then**
  process-new-edge(A, r, B);

Algorithm 2: process(A, X)

forall the *role names* s *with* $r \sqsubseteq^*_{\widehat{\mathcal{O}}} s$ **do**
  $R(s) \leftarrow R(s) \cup \{(A, B)\}$;
  queue$(A) \leftarrow$ queue$(A) \cup \bigcup_{\{B' | B' \in S(B)\}} \widehat{\mathcal{O}}(\exists s.B')$;
  forall the *concept names* A' *and role names* t, u **do**
    $t \circ s \sqsubseteq u \in \mathcal{O}$ and $(A', A) \in R(t)$ and $(A', B) \notin R(u)$ do
    process-new-edge(A', u, B);
  forall the *concept names* B' *and role names* t, u **do**
    $s \circ t \sqsubseteq u \in \mathcal{O}$ and $(B, B') \in R(t)$ and $(A, B') \notin R(u)$ do
    process-new-edge(A, u, B');

Algorithm 3: process-new-edge(A, r, B)

Axioms are distributed across the machines in the cluster based on their keys. A hash function, H maps a unique key, K, to a particular node, N, in the cluster.

$$H \colon K \mapsto N$$

For each concept A, care is taken to map $\widehat{\mathcal{O}}(A)$, queue(A) and $S(A)$ to the same node. This localizes the interaction (read/write) between these three sets, which in turn improves the performance. In order not to mix up the keys among these three sets, unique namespace is used along with the key. For example, $O : A, Q : A, S : A$, for $\widehat{\mathcal{O}}(A)$, queue(A) and $S(A)$ respectively, but, for the hash function, A is used in all the three cases.

After the axioms are loaded, each machine applies Algorithm 2 to only the queues local to it. In order to read/write to the non-local values, each machine uses the hash function, H. Each machine acts as a reasoner and cooperates with other machines to get the missing values and perform the classification task.

A single process called *Termination Controller*(TC), keeps track of the status of computation across all the nodes in the cluster. TC receives either *DONE* or *NOT-DONE* message from each machine. A double check termination strategy is

Axiom	Key	Value
$A \sqsubseteq B$	A	B
$A_1 \sqcap \ldots \sqcap A_n \sqsubseteq B$	A_i	$(A_1, \ldots, A_{i-1}, A_{i+1}, \ldots, A_n, B)$
$A \sqsubseteq \exists r.B$	A	(r, B)
$\exists r.A \sqsubseteq B$	(r, A)	B
$r \sqsubseteq s$	r	s
$r \circ s \sqsubseteq t$	r	(s, t)
	s	(r, t)

Table 4. Key-Value pairs for axioms

```
msgCount ← 0;
currentState ← NO-CHECK;
on pid ? status-msg → {
if status-msg = DONE then
    msgCount ← msgCount + 1;
    if msgCount = TOTAL-NODES then
        if currentState = NO-CHECK then
            currentState ← SINGLE-CHECK-DONE;
            broadcast(CHECK-AND-RESTART);
        else if currentState = SINGLE-CHECK-DONE then
            currentState ← DOUBLE-CHECK-DONE;
            broadcast(TERMINATE);
else if status-msg = NOT-DONE then
    msgCount ← 0;
    currentState ← NO-CHECK;
    pid ! CONTINUE-WORKING;
}
```

Algorithm 4: Termination Controller, TC

followed here. TC waits till it receives a *DONE* message from all the machines in the cluster. It then asks all the nodes to check if any local queues are non-empty. This is required because, after a node is done with OneIteration (Algorithm 5), there is a possibility of other nodes inserting values in the queues of this node. If this condition does indeed arise then a *NOT-DONE* message is sent to TC. TC resets its state to *NO-CHECK* and implements the double check termination strategy again. The pseudocode of TC is given in Algorithm 4. To simplify, TC is single threaded and works on only one message at a time. Process named *Job Controller* runs on each node of the cluster and implements the queue based algorithm. This is shown in Algorithm 6.

This approach is implemented in Java and the key-value store used is Redis[1]. Our system is called *DQuEL* and the source code is available at `https://github.com/raghavam/DQuEL`. We used a 13-node cluster with each node hav-

[1] `http://redis.io`

```
queues ← GetNonEmptyLocalQueues();
forall the queue A ∈ queues do
    forall the entry X ∈ A do
        process(A, X);
TC ! DONE;
```

Algorithm 5: OneIteration()

```
OneIteration();
on TC ? CHECK-AND-RESTART → {
queues ← GetNonEmptyLocalQueues();
if queues is ∅ then
    TC ! DONE;
else
    TC ! NOT-DONE;
} on TC ? CONTINUE-WORKING → OneIteration();
on TC ? TERMINATE → terminate-self;
```

Algorithm 6: Job Controller

ing two quad-core AMD Opteron 2300MHz processors and 12GB of heap size is available to JVM. Timeout limit was set to 2 hours.

Not-Galen, GO, NCI, SNOMED CT and 2-SNOMED ontologies were used for testing. The first three are obtained from `http://lat.inf.tu-dresden.de/~meng/toyont.html` and SNOMED CT can be obtained from `http://www.ihtsdo.org/snomed-ct`. 2-SNOMED is SNOMED replicated twice. The time taken by some popular reasoners such as Pellet, jCEL and ELK on these ontologies is given in Table 5. Table 6 shows the classification times of *DQuEL* with varying nodes.

3.3.1 Analysis Although the results are good for smaller ontologies, this approach turns out to be inefficient for larger ontologies such as SNOMED CT. The following two factors contributes to the inefficiency. a) Batch processing of axioms is not possible because each entry in the queue could be different from the one processed before. It is a known fact that batch processing especially involving communication over networks improves the performance drastically. Batch

Ontology	#Axioms	Pellet	jCEL	ELK
Not-Galen	8,015	12.0	3.0	1.0
GO	28,897	5.0	5.0	2.0
NCI	46,870	6.0	7.0	3.0
SNOMED CT	1,038,481	1,845.0	327.0	24.0
2-SNOMED	2,076,962	OOM	687.0	64.0

Table 5. Classification time (in seconds) of Pellet, jCEL and ELK reasoners

Ontology	1 node	7 nodes	13 nodes
Not-Galen	153.77	147.07	41.37
GO	165.94	147.28	43.12
NCI	205.62	55.52	30.21
SNOMED CT	TimeOut	TimeOut	TimeOut
2-SNOMED	TimeOut	TimeOut	TimeOut

Table 6. Classification time (in seconds) of DQuEL

processing has been used to the extent possible, but the approach described in the next section is more amenable to that. b) Not all data required for the queue operations is available locally. For example, data of the form $\widehat{\mathcal{O}}(\exists r.B)$ might be present on a different node.

1. *Pros* Good load balancing is possible since the hash function makes an attempt to distribute axioms across the cluster equally.
2. *Cons* Large ontologies like SNOMED CT generate many (X, Y) values which makes rule R3 (Table 1) computation slow compared to other rules. This problem is alleviated in the approach described next, by choosing r as the key in $R(r)$. This spreads $R(r)$ across the cluster and enables more nodes to work on it.
3. *Axiom Distribution* It is a random distribution of axioms.
4. *Rule Distribution* In each iteration, all the rules are applied by every node in the cluster.
5. *Optimizations* $R(r)$ sets involving role chains are duplicated as show in Table 4. This makes it easy to retrieve (X, Y) pairs associated with either r or s in $r \circ s \sqsubseteq t$.

3.4 Distributed Fixpoint Iteration Approach

In fixpoint iteration approach, the completion rules are applied on the axioms iteratively until there are no changes to $S(X), R(r)$. This idea is extended to the distributed setting [16]. The completion rules from Table 2 and the general strategy mentioned in Algorithm 1 are used here. Axioms are represented as key-value pairs.

All the axioms in a normalized ontology fall into one of the normal form categories mentioned in Table 1. This allows axioms to be split into disjoint collections. Each such collection is assigned to a group of nodes in the cluster. Only one rule can be applied on axioms of a particular normal form. So this leads to a clear assignment of axioms and rules to nodes.

Architecture of this approach is shown in Figure 2. The number of nodes per group need not be the same across all the groups. Higher number of nodes are generally assigned to groups handling rules involving roles since they are generally slower.

$U(X)$ is used instead of $S(X)$ in this approach. In the naive fixpoint iteration approach, in order to apply a rule such as R1 from Table 1 on axioms of the form

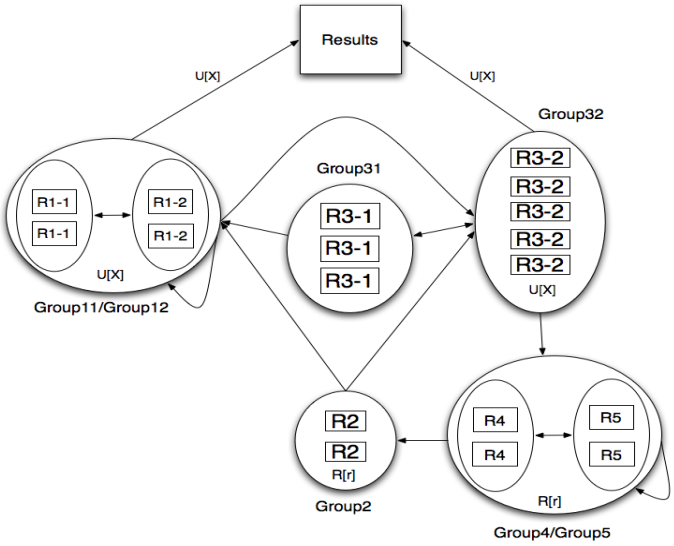

Fig. 2. Node assignment to rules and dependency among the completion rules. Each oval is a collection of nodes (rectangles).

Ontology	7 nodes	9 nodes	12 nodes
Not-Galen	43.0	42.27	41.06
GO	46.20	49.39	51.83
NCI	275.0	168.96	157.36
SNOMED CT	1,610.00	1,335.81	865.89
2-SNOMED	3,238.19	2,687.75	1,699.73

Table 7. Classification time (in seconds) of DistEL

$A_1 \sqcap \ldots \sqcap A_n \sqsubseteq B$, each $S(X)$ needs to be checked for the presence of $A_1, \ldots, A_n$. With $U(X)$, it turns into set intersection of conjuncts, which are generally small in number.

Termination is achieved with the help of barrier synchronization. At the end of each iteration, every node broadcasts a status message indicating whether any changes were made in this iteration and then waits for the other nodes to finish its current iteration. After receiving the update messages from all the nodes, if at least one node made an update then all the nodes continue with their next iteration. Algorithm terminates when no updates are made.

This approach is implemented in Java and makes use of Redis as the key-value store. The same cluster set up and ontologies as mentioned in the previous section are used here. This system is named *DistEL* and is available at `https://github.com/raghavam/DistEL`. The classification time of *DistEL* across several nodes is given in Table 7.

3.4.1 Analysis Although the runtimes of *DistEL* are high compared to existing reasoners, it is better than the other approaches described so far. With more optimizations such as dynamic load balancing, the results can be further improved.

MapReduce approach and the approach described here, use fixpoint iteration for classification. But compared to MapReduce approach, an important advantage is that nodes can communicate with each other. Since the completion rules are interdependent, inter-node communication makes this approach efficient relatively. Since a node deals with all axioms of the same type, batch processing can be used here. Communications involving network and database can be improved a lot with the help of batch processing.

1. *Pros* Axioms and rules are neatly divided across the cluster based on their type.
2. *Cons* Improper load balancing.
3. *Axiom Distribution* Axioms are distributed based on their normal form type.
4. *Rule Distribution* Group of nodes in the cluster handle the same type of rule. All the rules are applied in parallel across the cluster.
5. *Optimizations* a) Batch processing b) only the newly made changes in the previous iteration are considered for the next iteration and c) for $R(r) = \{(X, Y)\}$, instead of considering r as the key (as done in Queue approach), (Y, r) is chosen as the key and value is X. Since distribution of set $R(r)$ is based on the key, this leads to a better spread of $R(r)$ across the cluster.

4 Evaluation Strategy

Comparing the performance of single machine reasoners with distributed reasoners is unfair; not only due to the nature of the computation involved but also due to the following reason. All things being equal, if the time taken on a single machine is t then on n nodes, it takes $p * t/n$ where p is the overhead, $p \geq 1$. In the case of super linear speedup, $p < 1$. But, as we have seen from the results presented here, this reduction in runtime does not happen at all in the case of distributed reasoners. But are all things equal? Considering the steps in the algorithm in both the cases is the same, one main difference is the computations in case of single machine reasoners takes place in-memory whereas for distributed approaches mentioned here either a database or a file system was used. The performance varies vastly in these cases.

A simple comparison of speeds is shown in Table 8. Integers are read and written to a HashMap in the case of RAM. For Redis, pipeline (batch operation) read and write are used. The code used for this experiment is available at `https://gist.github.com/raghavam/2be48a98cae31c418678`. Admittedly, this is a rather simple experiment, but it shows the difference in read/write speeds for simple operations. For read operation, usage of RAM is 43 times faster than Redis and 26 times faster than file. For write operation also, there is a similar variation in performance. For random read and write operations, Redis performs better than a file.

Operation	#Items	RAM	Redis	File
Read	1,000,000	0.0861	3.719	
Write	1,000,000	0.1833	4.688	0.2181

Table 8. Comparison of speed (in seconds) for simple read, write operations when using RAM, Redis and File

A comparison should not be made between a distributed computation and a single machine computation since it is not a like-for-like comparison. But, since some sort of baseline is required, following two strategies can be considered.

1. Re-implement existing reasoners by making use of the storage system that is used in the distributed model. For example, use Redis or a file in jCEL or ELK.
2. Simulate distribution using existing reasoners by running a reasoner on each node of the cluster. A messaging system can be used to facilitate communicate and exchange of missing data on each node. But the performance in this case depends on the axiom distribution. So care should be taken to follow the same axiom distribution model in case of the distributed approach.

5 The Road Ahead

Although some progress has been made in distributed OWL 2 EL reasoning, current results clearly indicate that more needs to be done. Apart from further optimizations to the approaches presented here, following can be tried.

- Module based axiom distribution. For a given set of entities, a module includes all the axioms that are relevant to them [10]. If axioms are distributed based on modules, then perhaps inter-node communication can be reduced.
- Axiom distribution based on graph partitioning. If a graph of an ontology can be constructed then distribution of axioms based on vertex partitioning reduces the dependencies among the axioms.
- Hadoop variants. There are several variants to the core MapReduce approach such as Apache Spark[2], Iterative MapReduce[3], HaLoop[4] which might be more suitable than the core Hadoop's MapReduce.
- Other distributed frameworks. Peer-to-peer techniques such as use of MPI and alternative distributed frameworks such as Akka[5] can be tried. As mentioned earlier, peer-to-peer networks offer more control over communication between nodes when compared to MapReduce.
- Shared memory supercomputers. Since the completion rules are interdependent, may be it would be more efficient if shared memory, massively parallel supercomputers are used. But the disadvantage in this case is that these are specialized machines which are not commonly available.

[2] `http://spark.apache.org`
[3] `http://www.iterativemapreduce.org`
[4] `https://code.google.com/p/haloop`
[5] `http://akka.io`

6 Related Work

Apart from the work presented here, other approaches to distributed reasoning of OWL 2 EL ontologies have been tried. Distributed resolution technique was applied to $\mathcal{EL}^+$ classification in [19]. A peer-to-peer distributed reasoning approach was presented in [6]. But, both of them do not provide any evaluation. There is some work on parallelization of tableau algorithms related to various description logics [1, 14].

Though not distributed, parallelization of OWL 2 EL classification has been studied in [12, 18]. Classifying EL ontologies on a single machine using a database instead of doing it in memory has been tried in [9].

7 Epilogue

It is possible to have very large ontologies if axioms are generated automatically from streaming data or text. Reasoners should be capable of scaling up to these large ontologies. But, existing reasoners are severely handicapped by their use of only one machine. Scalable and distributed approaches to ontology reasoning is required. We reviewed and analyzed three distributed approaches to OWL EL ontology classification. Apart from this, we discussed some possible future directions and also evaluation strategies that can be followed to make the comparison fair between distributed and single machine approaches.

Acknowledgements. This work was supported by the National Science Foundation under award 1017225 "III: Small: TROn – Tractable Reasoning with Ontologies." Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

References

1. Aslani, M., Haarslev, V.: Parallel TBox Classification in Description Logics – First Experimental Results. In: Proceedings of the 19th European Conference on Artificial Intelligence. pp. 485–490. IOS Press, Amsterdam, The Netherlands (2010)
2. Baader, F., Brandt, S., Lutz, C.: Pushing the EL Envelope. LTCS-Report LTCS-05-01, Chair for Automata Theory, Institute for Theoretical Computer Science, Dresden University of Technology, Germany (2005), `http://lat.inf.tu-dresden.de/research/reports.html`
3. Baader, F., Calvanese, D., McGuinness, D., Nardi, D., Patel-Schneider, P. (eds.): The Description Logic Handbook: Theory, Implementation, and Applications. Cambridge University Press (2007)
4. Baader, F., Lutz, C., Suntisrivaraporn, B.: Is Tractable Reasoning in Extensions of the Description Logic EL Useful in Practice? In: Proceedings of the 2005 International Workshop on Methods for Modalities (M4M-05) (2005)
5. Baader, F., Lutz, C., Suntisrivaraporn, B.: CEL—A Polynomial-time Reasoner for Life Science Ontologies. In: Proceedings of the 3rd International Joint Conference on Automated Reasoning (IJCAR'06), Seattle, WA, USA, August 17-20, 2006.

Lecture Notes in Artificial Intelligence, vol. 4130, pp. 287–291. Springer-Verlag (2006)
6. Battista, A.D.L., Dumontier, M.: A Platform for Reasoning with OWL-EL Knowledge Bases in a Peer-to-Peer Environment. In: Proceedings of the 5th International Workshop on OWL: Experiences and Directions, Chantilly, VA, United States, October 23-24 2009. CEUR Workshop Proceedings, vol. 529. CEUR-WS.org (2009)
7. Cimiano, P.: Ontology Learning and Population from Text: Algorithms, Evaluation and Applications. Springer-Verlag New York, Inc., Secaucus, NJ, USA (2006)
8. Dean, J., Ghemawat, S.: 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. pp. 137–150. USENIX Association (2004)
9. Delaitre, V., Kazakov, Y.: Classifying ELH Ontologies In SQL Databases. In: Proceedings of the 5th International Workshop on OWL: Experiences and Directions (OWLED 2009), Chantilly, VA, United States, October 23-24, 2009
10. Grau, B.C., Horrocks, I., Kazakov, Y., Sattler, U.: Just the Right Amount: Extracting Modules from Ontologies. In: Proceedings of the 16th International Conference on World Wide Web. pp. 717–726. WWW '07, ACM, New York, NY, USA (2007), http://doi.acm.org/10.1145/1242572.1242669
11. Hitzler, P., Krötzsch, M., Rudolph, S.: Foundations of Semantic Web Technologies. Chapman & Hall/CRC (2010)
12. Kazakov, Y., Krötzsch, M., Simancik, F.: Concurrent Classification of EL Ontologies. In: 10th International Semantic Web Conference, Bonn, Germany, October 23-27. Lecture Notes in Computer Science, vol. 7031, pp. 305–320. Springer (2011)
13. Lécué, F., Tucker, R., Bicer, V., Tommasi, P., Tallevi-Diotallevi, S., Sbodio, M.L.: Predicting Severity of Road Traffic Congestion using Semantic Web Technologies. In: Proceedings of the 11th Extended Semantic Web Conference (ESWC2014), Anissaras, Crete, Greece, May 25–May 29, 2014. Springer (2014)
14. Liebig, T., Müller, F.: Parallelizing Tableaux-based Description Logic Reasoning. In: Proceedings of the 2007 OTM Confederated International Conference on On the Move to Meaningful Internet Systems - Volume Part II. pp. 1135–1144. OTM'07, Springer-Verlag, Berlin, Heidelberg (2007), http://dl.acm.org/citation.cfm?id=1780453.1780504
15. Mutharaju, R.: Very Large Scale OWL Reasoning through Distributed Computation. In: International Semantic Web Conference (2). Lecture Notes in Computer Science, vol. 7650, pp. 407–414. Springer (2012)
16. Mutharaju, R., Hitzler, P., Mateti, P.: DistEL: A Distributed EL+ Ontology Classifier. In: Liebig, T., Fokoue, A. (eds.) Proceedings of the 9th International Workshop on Scalable Semantic Web Knowledge Base Systems, Sydney, Australia. CEUR Workshop Proceedings, vol. 1046, pp. 17–32. CEUR-WS.org (2013)
17. Mutharaju, R., Maier, F., Hitzler, P.: A MapReduce Algorithm for EL+. In: Proceedings of the 23rd International Workshop on Description Logics (DL 2010), Waterloo, Ontario, Canada, May 4-7, 2010. CEUR Workshop Proceedings, vol. 573. CEUR-WS.org (2010)
18. Ren, Y., Pan, J.Z., Lee, K.: Parallel ABox reasoning of EL ontologies. In: Proceedings of the 2011 Joint International Conference on the Semantic Web. pp. 17–32. JIST'11, Springer, Heidelberg (2012)
19. Schlicht, A., Stuckenschmidt, H.: MapResolve. In: Web Reasoning and Rule Systems – 5th International Conference, RR 2011, Galway, Ireland, August 29-30, 2011. Lecture Notes in Computer Science, vol. 6902, pp. 294–299. Springer (2011)

20. Urbani, J., van Harmelen, F., Schlobach, S., Bal, H.E.: QueryPIE: Backward Reasoning for OWL Horst over Very Large Knowledge Bases. In: 10th International Semantic Web Conference, Bonn, Germany, October 23-27, 2011. Lecture Notes in Computer Science, vol. 7031, pp. 730–745. Springer (2011)
21. Urbani, J., Kotoulas, S., Maassen, J., Van Harmelen, F., Bal, H.: WebPIE: A Web-scale Parallel Inference Engine Using MapReduce. Journal of Web Semantics. 10, 59–75 (Jan 2012), http://dx.doi.org/10.1016/j.websem.2011.05.004
22. Urbani, J., Kotoulas, S., Oren, E., van Harmelen, F.: Scalable Distributed Reasoning Using MapReduce. In: Proceedings of the 8th International Semantic Web Conference, ISWC 2009, Chantilly, VA, USA, October 25-29, 2009
23. Weaver, J., Hendler, J.A.: Parallel Materialization of the Finite RDFS Closure for Hundreds of Millions of Triples. In: 8th International Semantic Web Conference, ISWC 2009, Chantilly, VA, USA, October 25-29, 2009. Lecture Notes in Computer Science, vol. 5823, pp. 682–697. Springer (2009)
24. Zhou, Z., Qi, G., Liu, C., Hitzler, P., Mutharaju, R.: Scale reasoning with fuzzy-EL+ ontologies based on MapReduce. In: Proceedings of the IJCAI-2013 Workshop on Weighted Logics for Artificial Intelligence, WL4AI-2013, Beijing, China, August 2013. pp. 87–93 (2013)