Parallelization of Conjunctive Query Answering over Ontologies

E. Patrick Shironoshita[1(✉)], Da Zhang[2], Mansur R. Kabuka[1,2], and Jia Xu[2]

[1] INFOTECH Soft, Inc., 1201 Brickell Avenue, Suite 220, Miami, FL 33131, USA
patrick@infotechsoft.com
[2] University of Miami, Coral Gables, FL 33124, USA

Abstract. Efficient query answering over Description Logic (DL) ontologies with very large datasets is becoming increasingly vital. Recent years have seen the development of various approaches to ABox partitioning to enable parallel processing. Instance checking using the enhanced most specific concept (MSC) method is a particularly promising approach. The applicability of these distributed reasoning methods to typical ontologies has been shown mainly through anecdotal observation. In this paper, we present a parallelizable, enhanced MSC method for the answering of ABox conjunctive queries, using a set of syntactic conditions that permit querying of large practical ontologies in reasonable time, and combining it with pattern matching to answer queries over role assertions. We also present execution time and efficiency of an implementation deployed over computing clusters of various sizes, showing the ability of the method to process instance checking for large scale datasets.

1 Introduction

Description Logics (DL) are now widely being used to model and represent structured and semi-structured data in different applications [1], particularly through the use of the Web Ontology Language (OWL). A core task for DL systems is to provide an efficient way to answer queries over the extensional level of the ontology, that is, to compute answers that are not only asserted, but logically implied by the ontology [2]. Considerable efforts have been dedicated to the optimization of algorithms for query answering [2–4]. One of the challenges faced in this era of increasing data wealth is to produce responsive results for queries over very large data sets [5]. However, even as reasoners for very expressive DLs have been created, performing reasoning over very large ABoxes is still prohibitive [2,6,7].

In the last few years, methods for parallel and distributed reasoning over expressive ontologies have been published [5,8–10]; these methods generally seek to make use of fast syntactic checks to generate a set of independent partitions that can be processed in parallel. In [11], we have proposed a method for instance checking, combining the work in [8,9] with the idea of a most specific concept (MSC) [12–14] to move the reasoning task from the very large ABox into a much smaller TBox. Empirical evaluation of the method has shown its ability to perform sound and complete instance checking over $\mathcal{SHI}$ DLs within reasonable time. Moreover, the method is inherently parallelizable, lending itself to implementation within clusters of commodity hardware.

In this paper, we examine this enhanced MSC method first described in [11], extend it to use in conjunctive queries, and evaluate its parallelization. First, we provide a detailed definition of the MSC method and of the syntactic conditions that enable its use for instance checking. Following this, we describe the use of the MSC method in conjunction with pattern matching to answer ABox conjunctive queries. Subsequently, results of tests performed for accuracy of the enhanced MSC method and of its extension to conjunctive queries are presented, as well as results from experiments over a parallelized implementation. Finally, we discuss our future work and provide our conclusions.

2 The Enhanced Most Specific Concept (MSC) Method

2.1 Basic MSC Method

A Description Logics (DL) knowledge base, also referred to as an ontology, is typically defined a tuple, denoted $\mathcal{K} = (\mathcal{T}, \mathcal{A})$, where the *terminological* component or TBox $\mathcal{T}$ contains definitions of concepts and roles, and the *assertional* component or ABox $\mathcal{A}$ contains assertions about membership of individuals in concepts and about role relations between individuals. The set of roles, concepts, and individual instances in an ontology are denoted respectively by $\mathbf{R}$, $\mathbf{C}$, and $\mathbf{I}$. The discussion in this paper assumes that the reader is familiar with DL concepts and notations; the reader is referred to [15] for details. For the discussion below, we will use the example illustrated in Table 1.

Definition 1 (Most Specific Concept). [12,13] *Let $\mathcal{K} = \{\mathcal{T}, \mathcal{A}\}$ be an ontology, and a be an individual in $\mathbf{I}$. The most specific concept for a w.r.t. $\mathcal{A}$, written $\mathrm{MSC}_{\mathcal{T}}(\mathcal{A}, a)$, is a concept such that for every concept D where $\mathcal{K} \models D(a)$, $\mathcal{T} \models \mathrm{MSC}_{\mathcal{T}}(\mathcal{A}.a) \sqsubseteq D$.*

If $\mathrm{MSC}_{\mathcal{T}}(\mathcal{A}, a)$ can be derived, then, to test whether $\mathcal{K} \models Q(a)$ holds for an arbitrary concept Q it suffices to test if $(\mathcal{T} \cup \{Q\}) \models \mathrm{MSC}_{\mathcal{T}\cup\{Q\}}(\mathcal{A}, a) \sqsubseteq Q$. We call the concept Q the *query*. Note that Q needs to be inserted into the TBox in simple form, which may require in turn the insertion of additional named concepts as well as general concept inclusion (GCI) axioms to express the necessary equivalences for these inserted concepts. In the remainder of this paper, we will assume that the query Q has been inserted into the TBox so that $\mathrm{MSC}_{\mathcal{T}}(\mathcal{A}, a)$ denotes the MSC calculated including Q.

Computation of the MSC for a given individual a as defined above can be performed using a *rolling up* procedure adapted from the one first introduced in [16].

Definition 2 (Basic MSC Rollup Procedure). *Provided that the ABox does not contain assertion cycles, computation of the MSC can be performed recursively as follows:*

Table 1. Example ABox

TBox
Headmaster $\sqsubseteq$ Professor
Professor $\sqsubseteq$ Person
MagicCourse $\sqsubseteq$ Course
Muggle $\sqsubseteq$ $\neg$Wizard
takesCourse.MagicCourse $\sqsubseteq$ Wizard
$\exists$isHeadOf.School $\sqcap$ Person $\sqsubseteq$ Headmaster
ABox
School(hogwarts)
Professor(albus)
Professor(severus)
MagicCourse(potions)
MagicCourse(transfiguration)
Course(math)
Student(harry)
Muggle(dudley)
isHeadOf(hogwarts, albus)
takesCourse(harry, transfiguration)
taughtBy(transfiguration, albus)
taughtBy(potions, severus)

1. *for a given individual a, start with an empty* $\mathrm{MSC}_{\mathcal{T}}(\mathcal{A}, a)$*;*
2. *for every concept assertion $C(a)$,*
   $\mathrm{MSC}_{\mathcal{T}}(\mathcal{A}, a) \leftarrow \mathrm{MSC}_{\mathcal{T}}(\mathcal{A}, a) \sqcap C$*;*
3. *for every role assertion $R(a, b)$,*
   $\mathrm{MSC}_{\mathcal{T}}(\mathcal{A}, a) \leftarrow \mathrm{MSC}_{\mathcal{T}}(\mathcal{A}, a) \sqcap \exists R.\mathrm{MSC}_{\mathcal{T}}(\mathcal{A} \backslash \{R(a, b)\}, b)$*;*
4. *for every individual equality assertion $a = a'$,*
   $\mathrm{MSC}_{\mathcal{T}}(\mathcal{A}, a) \leftarrow \mathrm{MSC}_{\mathcal{T}}(\mathcal{A}, a) \sqcap \mathrm{MSC}_{\mathcal{T}}(\mathcal{A}, a')$*.*

So, in the example ABox above, the MSC for `severus` is

$$\mathrm{MSC}_{\mathcal{T}}(\mathcal{A}, \texttt{severus}) = \texttt{Professor} \sqcap \exists \texttt{taughtBy}^{-}.\texttt{MagicCourse} \tag{1}$$

It is important to note that while class assertions generate relatively simple concepts, role assertions are capable of generating very complex concepts. Suppose for example that ABox $\mathcal{A}_n$ consists of role assertions $R_0(a_0, a_1)$, $R_1(a_1, a_2)$, $\ldots$, $R_n(a_n, a_{n+1})$; then, the MSC of a_0 is:

$$\mathrm{MSC}_{\mathcal{T}}(\mathcal{A}_n, a_0) = \exists R_1.(\exists R_2.(\ldots(\exists R_n.\mathrm{MSC}_{\mathcal{T}}(a_{n+1}))\ldots)) \tag{2}$$

Thus, in the example ABox of Table 1, the MSC for `harry` is

$$\begin{aligned}\mathrm{MSC}_{\mathcal{T}}(\mathcal{A}, \texttt{harry}) = \texttt{Student} \sqcap \exists \texttt{takesCourse}^{-}.&\\ (\texttt{MagicCourse} \sqcap \exists \texttt{taughtBy}.&\\ (\texttt{Professor} \sqcap \texttt{isHeadOf}^{-}.\texttt{School}))&\end{aligned} \tag{3}$$

Two main issues preclude this basic MSC method from proving fully useful with expressive ontologies. First, if assertion cycles are present in the ABox, the method does not terminate. For example, consider what would happen if the following is inserted in the TBox:

$$\texttt{isProtegeOf}(\texttt{albus}, \texttt{harry}) \tag{4}$$

In this case, a cycle forms among `harry`, `transfiguration`, and `albus`.

Second, unless the ABox is highly disconnected, the method has the potential to generate very large concepts, of size in the same order of the ABox itself. Consider what happens if the following assertion is added to the ABox:

$$\texttt{takesCourse}(\texttt{harry}, \texttt{potions}) \tag{5}$$

In this case, all individuals become connected with each other, and the MSC of every individual ends up being the same size of the ABox.

Xu et al. [11] define a set of improvements that result in the *Enhanced MSC Method*. This enhancement consists of two parts: a mechanism to address assertion cycles, and a set of syntactic conditions to reduce the size of the MSCs in practical ontologies.

2.2 MSC Computation with Assertion Cycles

To address assertion cycles, Xu et al. [11] use nominals to indicate the joint node of a cycle, as previously suggested in [13,17].

Definition 3 (MSC Rollup of Assertion Cycles). *When a cycle is found starting and ending at individual a_c, the individual is represented by its corresponding nominal class $\{a_c\}$, and the conversion of role assertions within the cycle requires modification of the rollup method as follows: If a_c is an individual where a cycle is found, select a direction to go through the cycle and:*

- *for $R(a_c, x)$, $x \neq a_c$*
  $\mathrm{MSC}_{\mathcal{T}}(\mathcal{A}, a_c) \leftarrow \mathrm{MSC}_{\mathcal{T}}(\mathcal{A}, a_c) \sqcap (\{a_c\} \sqcap \exists R.\mathrm{MSC}_{\mathcal{T}}(\mathcal{A} \backslash \{R(a_c, x)\}, x))$
- *for $R(y, a_c)$, $y \neq a_c$*
  $\mathrm{MSC}_{\mathcal{T}}(\mathcal{A}, y) \leftarrow \mathrm{MSC}_{\mathcal{T}}(\mathcal{A}, y) \sqcap \exists R.\{a_c\}$
- *for $R(a_c, a_c)$,*
  $\mathrm{MSC}_{\mathcal{T}}(\mathcal{A}, a_c) \leftarrow \mathrm{MSC}_{\mathcal{T}}(\mathcal{A}, a_c) \sqcap \{a_c\}$
  $\sqcap \exists R.\{a_c\}$
- *for any other $R(x, y)$ in the cycle, for $x \neq a_c$ and $y \neq a_c$,*
  $\mathrm{MSC}_{\mathcal{T}}(\mathcal{A}, x) \leftarrow \mathrm{MSC}_{\mathcal{T}}(\mathcal{A}, x) \sqcap \exists R.\mathrm{MSC}_{\mathcal{T}}(\mathcal{A} \backslash \{R(x, y)\}, y)$

Thus, an ABox $\mathcal{A}_c = \{R_0(a_0, a_1), R_1(a_1, a_2), \ldots, R_n(a_n, a_0)\}$ has an assertion cycle, then the MSC is obtained as follows:

$$\mathrm{MSC}_{\mathcal{T}}(\mathcal{A}_c, a_0) = \{a_0\} \sqcap \exists R_1.(\exists R_2.(\ldots(\exists R_n.\{a_0\})\ldots)) \tag{6}$$

So, suppose that the ABox in Table 1 is augmented with the assertion in Eq. (4), then the MSC for `harry` becomes

$$\begin{aligned}\mathrm{MSC}_{\mathcal{T}}(\mathcal{A}, \texttt{harry}) = &\{\texttt{harry}\} \sqcap \texttt{Student} \sqcap \exists \texttt{takesCourse}^{-}.\\ &(\texttt{MagicCourse} \sqcap \exists \texttt{taughtBy}.\\ &(\texttt{Professor} \sqcap \texttt{isHeadOf}^{-}.\texttt{School}\\ &\sqcap \texttt{isProtegeOf}^{-}.\{\texttt{harry}\}))\end{aligned} \tag{7}$$

Use of the MSC method to perform instance retrieval is straightforward. Suppose it is desired to query an ABox to retrieve all instances of a concept Q. It suffices to generate $\mathrm{MSC}_{\mathcal{T}}(\mathcal{A}, a)$ for every individual a in the ABox, and then accept every individual where $(\mathcal{T} \cup Q) \models \mathrm{MSC}_{\mathcal{T}}(\mathcal{A}, a) \sqsubseteq Q$. Note that the query concept Q is added to the TBox being verified.

2.3 Syntactic Conditions

In [11], syntactic conditions verifiable in polynomial time or better were defined, to enable reduction on the size of the MSC and thus permit TBox reasoning in tractable time. These conditions are based on the following:

Lemma 1. *Given two individuals a and b and a role R, a role assertion $R(a, b)$ influences the classification of individual a into concept A if and only if*

$$\mathcal{K} \models \exists R.B \sqcap A_0 \sqsubseteq A \tag{8}$$

where $\mathcal{K} \models B(b)$ and where $A_0 \not\sqsubseteq A$ summarizes information about a not contained in A.

Proposition 1. *Let $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ be a $\mathcal{SHI}$ ontology containing named concept A, concepts A_0 and B, and role R. If Eq. (8) holds, there must exist some GCIs in $\mathcal{T}$ of the form*

$$\exists R'.C_1 \bowtie C_2 \sqsubseteq C_3 \tag{9}$$

where $R \sqsubseteq R'$, $\bowtie$ is a placeholder for either $\sqcap$ or $\sqcup$, and C_i's are concepts.

Proof of this proposition can be found in [8].

Proposition 1 directly leads to a first syntactic condition denoted SYN_COND.

Definition 4 (SYN_COND). *Role assertions of the form $R(a, b)$ are said to be **true** for* SYN_COND *if role R participates in at least one axiom that can be logically converted to the form of Eq. 9 for some $R \sqsubseteq R'$, **false** otherwise.*

Assertions $R(a, b)$ with SYN_COND = **false** do not affect the classification of a unless either R or some R' such that $R \subseteq R'$ exist in the query, and can be safely removed from the calculation of $\text{MSC}_{\mathcal{T}}(\mathcal{A}, a)$. By symmetry to the inverse role R^{-}, this condition also applies to b. In practice, the axioms more likely to be found are of the form $C \sqsubseteq \exists R.D$, which is equivalent to $C \sqcap \top \sqsubseteq \exists R.D$. Also note that $(C \sqsubseteq \forall R.D) \equiv (\exists R.\neg C \sqsubseteq \neg D)$. In the example in Table 1, assertions with role `isHeadOf` have SYN_COND = **true**, while assertions with roles `takesCourse` and `taughtBy` have SYN_COND = **false** and can be safely removed from consideration, unless the roles exist in the query itself. Note that in this case, the MSC for `harry` is reduced to

$$\text{MSC}_{\mathcal{T}}(\mathcal{A}, \texttt{harry}) = \texttt{Student} \sqcap \exists \texttt{takesCourse}^{-}.\texttt{MagicCourse} \tag{10}$$

A second syntactic condition presented in [11] relies on the identification of explicit concept assertions and disjointness axioms that indicate that an individual cannot be classified to an existential restriction:

Definition 5 (SYN_COND_DJ). *Role assertions $R(a, b)$ with* SYN_COND = ***true*** *are said to be* ***true*** *for* SYN_COND_DJ *if there do not exist any of:*

- *an explicit concept assertion $B_0(b)$ such that $\mathcal{K} \models B_0 \sqsubseteq \neg C_1$;*
- *an explicit concept assertion $A_0(a)$ such that $\mathcal{K} \models A_0 \sqsubseteq \neg C_3$; or*
- *an explicit concept assertion $A_0(a)$ such that $\mathcal{K} \models A_0 \sqsubseteq \neg(C_3 \sqcup \neg C_2)$*

for C_1, C_2, an C_3 as in Eq. (9) and $R \sqsubseteq R'$.

Role assertions with SYN_COND_DJ = **false** can be safely removed from the calculation of the MSC of any individual, since they do not affect classification of either a or b. For example, in Table 1, role assertions with role `takesCourse` have SYN_COND = **true** due to the assertion `takesCourse.MagicCourse` $\sqsubseteq$ `Wizard`. However, suppose that the following assertion were inserted into the ABox:

$$\texttt{takesCourse}(\texttt{dudley}, \texttt{math}) \tag{11}$$

This assertion has SYN_COND_DJ = **false**, since `dudley` is an instance of `Muggle`, which in turn is disjoint with `Wizard`, the filler concept in the assertion above.

Definition 6 (SYN_COND_SC). *Role assertions $R(a, b)$ with* SYN_COND = ***true*** *are said to be* ***true*** *for* SYN_COND_SC *if, for every GCI of the form in Eq. (9) there exists an explicit concept assertion $A_0(a)$ such that $\mathcal{K} \models A_0 \sqsubseteq C_3$.*

Role assertions with SYN_COND_SC = **true** are redundant for the classification of a, and can therefore be safely removed from the calculation of the MSC of any individual. For example, the role assertion `Headmaster(albus)` would have SYN_COND_SC = **true**.

Application of these three syntactic conditions to the computation of the MSC of a given individual results in a significant reduction in the size of the MSC for practical ontologies. The reasons for this reduction will be established in more detail in the next section, but first we provide a definition for the parallelization of the algorithm.

2.4 Parallelization of the MSC Method

The MSC method, including the syntactic conditions detailed above, is highly parallelizable, due to the following:

Proposition 2. *For a given knowledge base $\mathcal{K} = (\mathcal{T}, \mathcal{A})$, computation of* $\mathrm{MSC}_{\mathcal{T}}(\mathcal{A}, a)$ *for an individual a can be performed independently of the computation of the MSC of any other individual.*

Proof of this proposition follows directly from the definition of the MSC computation in Definitions 2 and 3, as well as in the definition of the syntactic conditions, where it should be clear that MSC computation depends only on the initial state of the knowledge base. This means that the calculation of the MSCs for all individuals can be done in parallel. Instance checking then needs to be performed against $\mathcal{T} \cup \mathrm{MSC}_{\mathcal{T}}(\mathcal{A}, a)$ for every individual a.

As is shown later in Sect. 4, parallelization of the MSC method allows for the processing of extremely large ABoxes within clusters of commodity hardware. The resulting computational complexity is sublinear with respect to the size of the ABox, which indicates linear complexity for single-machine implementation.

3 Conjunctive Query Answering and SPARQL

The enhanced MSC method described in the previous section provides an effective, parallelizable algorithm for instance checking. In this section, we discuss its use for ABox conjunctive query answering.

Definition 7 ABox Conjunctive Query [7]**.** *Let* $\mathbf{C}$*,* $\mathbf{R}$ *and* $\mathbf{I}$ *denote the set of concepts, roles, and individuals in the ABox as before, and let* $\mathbf{V}$ *denote a set of variables disjoint with* $\mathbf{C}$*,* $\mathbf{R}$ *and* $\mathbf{I}$*. An* atom *is an expression $C(t)$, $R(t, t')$, or $t = t'$, where t and t' are members of* $\mathbf{V} \cup \mathbf{I}$*. An* ABox conjunctive query *is a collection of atoms.*

The result set of an ABox conjunctive query is a set of tuples, where each tuple contains a set of individuals that map to each variable in a conjunctive query. In the following discussions, we will use the notation $?x$ where necessary to refer specifically to variables, where x is any lowercase letter, borrowing from SPARQL notation.

ABox conjunctive queries are expressible in the SPARQL query language for RDF. SPARQL is designed as an extensible language where different entailment regimes can be employed, thus enabling the use of logic reasoning to derive results. Several works have been published that employ description logic entailment, particularly when using DL entailment regimes over OWL knowledge bases [18–20]. There are also efforts to enable SPARQL querying of large RDF graphs using distributed computing techniques [21,22]; however, these efforts do not include the use of DL entailment and are thus limited to the lesser expressive RDF semantics. It must be noted that SPARQL can be used for other queries - for example, they can be used to query for all class memberships for a single individual [20]. Arguably, however, ABox conjunctive queries are the most widely used, as they are designed to retrieve instances and their associated data values, in a manner analogous to SQL queries in relational databases.

Resolution of ABox conjunctive query atoms of the form $C(t)$ using the MSC method is straightforward, as this is equivalent to instance retrieval for a given class. On the other hand, query atoms of the form $R(t, t')$ and $t = t'$ require additional processing to ensure that all possible individual equalities are taken into account. For example, suppose that in the example in Table 1, the following assertion were added:

$$\texttt{albus} = \texttt{dumbledore} \tag{12}$$

Then, a query for `?a` $=$ `albus` should return both `albus` and `dumbledore`. Moreover, a query for `isHeadOf(?b,?a)` would need to return the tuples (`hogwarts`, `albus`) and (`hogwarts`, `dumbledore`).

For $R(t, t')$, one possible solution is to simply traverse the ABox and perform *pattern matching*:

Definition 8 Pattern Matching. *Triples $R'(a, b)$ are matched to a query $R(t, t')$ if all the following conditions are fulfilled:*

(a) $R' \sqsubseteq R$;
(b) if t (t') is an individual, then $t = a$ ($t' = b$); and
*(c) if t (t') is a variable, then **true**;*

If all conditions are fulfilled, then the triple $R(a, b)$ produces a solution to the query atom by mapping t to a if t is a variable, and similarly t' to b. For example, the query atom `taughtBy(?a, ?b)` results in two solution sets, (`transfiguration`, `albus`) and (`potions`, `severus`).

Note however that this means that it is necessary to determine if the knowledge base entails individual equality; if the assertion in Eq. (12) were included in the ABox, then (`transfiguration`, `dumbledore`) would also be a solution to the query. Of course, resolution of equality is also trivially necessary to resolve atoms of the form $t = t'$.

It is possible to avoid this issue if the *unique name assumption* (UNA) is made, similar to database querying. Alternatively, it has been shown that for ontologies in the *DL-Lite* family, reasoning without the UNA can be reduced to reasoning with the UNA in polynomial time, through the resolution of functional properties and of the symmetric-reflexive-transitive closure of the individual equality assertions [23].

Proposition 3. *In a $\mathcal{SHI}$ DL, reasoning without the UNA can be reduced to reasoning with the UNA in polynomial time.*

Proof. The differences between the various *DL-Lite* families, and particularly the *DL-Lite*$^{\mathcal{HF}}_{core}$ version, and a $\mathcal{SHI}$ DL, reside in restrictions regarding class constructs. Given that neither *DL-Lite* nor $\mathcal{SHI}$ allow nominals in the TBox, it is not possible for any class assertion to result in an inference of individual equality. Therefore, the results for *DL-Lite* regarding reduction to UNA in [23] are also applicable to $\mathcal{SHI}$ DLs.

Clearly then, ABox conjunctive queries, including those expressed in the SPARQL query language, can be resolved using the enhanced MSC method for class query atom resolution, and pattern matching for role query atom resolution. Since pattern matching is also inherently parallelizable, as it involves verification of every triple in the ABox independent of the others, the entire query answering algorithm can be implemented as an iterative parallel solution.

Corollary 1. *In a $\mathcal{SHIQ}$ DL, reasoning without the UNA can be reduced to reasoning with the UNA in polynomial time.*

The proof follows from Proposition 3.

The MSC method can be extended to DL with expressivity $\mathcal{SHIQ}$ if UNA is assumed, which can be done using the result above. Details of such extension are outside the scope of this paper, but are straightforward, and generally follow the extensions to equality-free module extraction detailed in [8].

4 Experimental Evaluation

4.1 MSC Method Accuracy

To test both the accuracy and to measure parallelization speedup of the enhanced MSC method, we set up clusters of compute-optimized instances through Amazon Web Services (AWS)[1]. The enhanced MSC method with syntactic condition correction was implemented in Java and Scala to work over Apache Spark[2], installed over Hadoop HDFS and YARN.

For the accuracy tests, we used an in-memory version of our enhanced MSC implementation. We set up clusters of two c4.xlarge machines, each containing 4 virtual CPUs and 7.5 GB of memory, to run the enhanced MSC method, and compared the results against the HermiT reasoner version 3.8.1[3], running on a single c4.xlarge machine; HermiT was chosen as a comparison standard due to

Table 2. Times (in seconds) for execution of class and existential restriction queries

	Classes		Existential restrictions	
	MSC	HermiT	MSC	HermiT
Min.	7.82	0.67	7.30	0.76
Max.	19.66	780.20	26.94	2,848.68
Avg.	8.38	19.42	9.14	489.95
Std. Dev.	1.42	90.50	3.24	698.14
Median	8.16	0.79	8.19	135.94

1 aws.amazon.com.
2 https://spark.apache.org/.
3 http://www.hermit-reasoner.com.

its stability and speed; future tests will be done against other reasoners such as Pellet[4] and Konclude[5]. These tests were performed against a single department dataset for the University Ontology Benchmark (UOBM) [24], containing about 150,000 triples. The UOBM TBox was modified to convert cardinality restrictions to existential restrictions, since our current implementation only handles expressivity up to $\mathcal{SHI}$. This modified version can be provided upon request. The UOBM Tbox contains a total of 113 named classes and 35 object properties. We tested our enhanced MSC implementation against all 35 named class queries, and all 3,955 possible single-depth existential restriction queries, and verified 100% agreement between our enhanced MSC method implementation and HermiT. In addition, we recorded the running time for all query executions - the results can be seen in Table 2. It is interesting to note that the enhanced MSC method performs much better than HermiT for existential restrictions. Also note the large standard deviation found when running a hypertableaux-based reasoner like HermiT, where a few queries take a very long time to finish. This result also suggests the possibility of using enhanced MSC in tandem with a traditional reasoner when dealing with smaller datasets.

4.2 Parallelization

To test the parallelization of the MSC method, we used the c3.8xlarge instances, which provide 32 virtual CPUs and 60 GBs of storage. We used the Lehigh University Benchmark (LUBM) [25] to generate data sets of up to 500 million triples. LUBM was chosen as an initial test ontology due to its ability to generate datasets of varying size, while providing a reasonably expressive TBox. These data sets were stored using the TitanDB[6] graph database interface over an Apache HBase[7] backend. Both TitanDB and HBase were installed over Hadoop HDFS 2.7. Our prototype application over Spark accesses TitanDB through its standard Java interface. Data distribution and replication are performed by the database and are transparent to our application.

A test was performed to evaluate the scalability of the parallel MSC method over the number of triples in the ABox. This test was performed over a cluster of 10 c3.8xlarge machines in AWS. The results are shown in the log-log diagram in Fig. 1. As can be observed, the method shows clear sub-linear performance with respect to the size of the data set, as expected from an algorithm with linear performance in a sequential machine.

The performance with respect to the number of machines was evaluated in two parts. First, to evaluate under small cluster conditions, a 500,000 triple set was assembled and used to test against 1 to 10 machines. The execution time and the efficiency with respect to single-machine execution are shown in Fig. 2(a). To obtain evaluation for large numbers of machines, we used the ABox with 500

[4] https://github.com/stardog-union/pellet.
[5] http://derivo.de/produkte/konclude/.
[6] http://thinkaurelius.github.io/titan/.
[7] http://hbase.apache.org/.

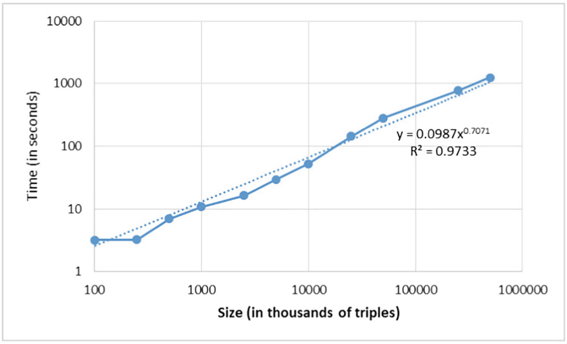
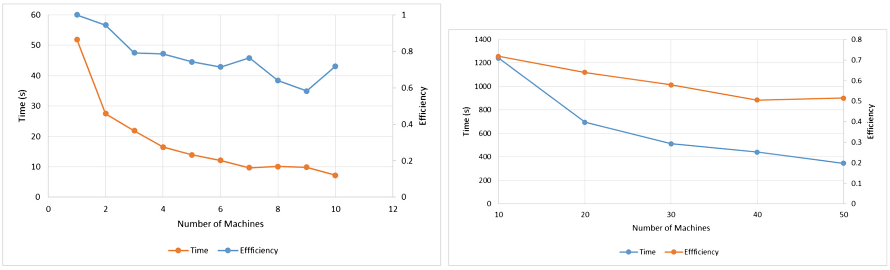


Fig. 1. Execution time vs. data set size with LUBM datasets, in AWS, for 10 c3.8xlarge machines with 32 cores each.

million triples and ran it against 10 to 50 machines; to provide a more realistic estimation, the efficiency value was corrected against the result for 500,000 triples in 10 machines. These higher scalability results are shown in Fig. 2(b).

In terms of raw performance, the parallel enhanced MSC algorithm was capable of performing instance checking for a dataset with 500 million triples and over 110 million individual instances in about 1,240 s, or around 20 min, using a cluster of 10 machines and a total of 320 execution cores. Using 50 machines, the execution time was 346 s, or somewhat less than 6 min. As a comparison, although the difference in algorithms means that execution times are not directly comparable, Oracle reports full ABox inference over 869 million triples in 62 min, and query performance over this pre-reasoned ABox in about 4.3 min, using specialized hardware [26]. It is also important to note that, since tests were performed over an uncontrolled environment, external perturbations could have affected some measurements. Nevertheless, it is clear that the MSC method provides performance comparable with top-of-the-line database technologies and very broad scalability.



(a) Efficiency for small # of machines (b) Efficiency for large # of machines

Fig. 2. Execution time and efficiency of parallelization.

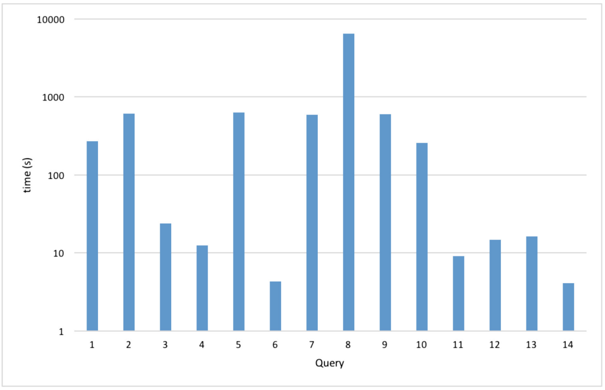


Fig. 3. Execution time for LUBM SPARQL queries, single node.

These results demonstrate that the enhanced MSC method is inherently parallelizable, enabling it to work with large scale ABoxes, and shows that it is useful with practical ontologies.

4.3 SPARQL Query Accuracy

An initial, non-parallelized prototype of a SPARQL query engine using the enhanced MSC method combined with pattern matching has been created. This initial prototype has been tested for accuracy against a LUBM(1, 0) dataset using a set of 14 test queries provided by the LUBM creators [25]. Testing shows 100% accuracy for all queries.

Speed of execution was measured by running the prototype on a VMWare virtual machine configured over a Dell PowerEdge R710 machine using an Intel Xeon E5620 CPU @ 2.40 GHz, running VSphere 4 Hypervisor. The virtual machine was set up for 4 cores and 64 GB of memory. The results are shown in Fig. 3. We expect to achieve significantly faster times using the enhanced MSC method, and are additionally working on the use of optimizations as described in [18].

5 Discussion and Future Work

The enhanced MSC method is inherently parallelizable, since instance checking for every individual in the ABox can be performed independently. Coupled with recent advances in cluster computing such as Apache Spark, large triple stores can be queried efficiently using commodity hardware clusters or cloud platforms. In combination with pattern matching, the method can also be used for answering conjunctive queries over the ABox, including those that can be formulated using the SPARQL query language.

We are currently working on the implementation of the ABox conjunctive query answering prototype to work over Spark, in order to test speedup and efficiency in its parallelization. We are also exploring the use of some optimization mechanisms that can be used to reduce the execution time. A parallelized implementation will also enable us to perform tests with other existing benchmarks such as the DBPedia SPARQL Benchmark (DBPSB). In addition, we are working on improvements in the efficiency of the parallelization of the MSC method. In particular, we are looking into combining multiple individuals that form part of the same connected component in the ABox in the same parallel task, since it can be seen in Definitions 2 and 3 that portions of the MSC computation can be shared among individuals provided that they are connected to each other.

6 Conclusion

In this paper, we have presented a parallel implementation of the enhanced MSC method and an extension to conjunctive queries using pattern matching, and we have evaluated execution time and efficiency as we varied the size of the data and the number of processors used. Since the method performs independent checking for every individual in the ABox, it is inherently parallelizable. The results show sub-linear performance with respect to the size of the ABox, which stems from its performance in linear time in the computation of the MSCs, and almost constant time for reasoning due to the small size of the resulting MSC.

Acknowledgements. This work is supported by grant # R44GM097851 from the National Institute of General Medical Sciences (NIGMS), part of the U.S. National Institutes of Health (NIH).

References

1. Horrocks, I.: Ontologies and the semantic web. Commun. ACM **51**(12), 58–67 (2008)
2. Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., Rosati, R.: Data complexity of query answering in description logics. Artif. Intell. **195**, 335–360 (2013)
3. Möller, R., Haarslev, V., Wessel, M.: On the scalability of description logic instance retrieval. In: Freksa, C., Kohlhase, M., Schill, K. (eds.) KI 2006. LNCS (LNAI), vol. 4314, pp. 188–201. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-69912-5_15
4. Motik, B., Shearer, R., Horrocks, I.: Optimized reasoning in description logics using hypertableaux. In: Pfenning, F. (ed.) CADE 2007. LNCS (LNAI), vol. 4603, pp. 67–83. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-73595-3_6
5. Priya, S., Guo, Y., Spear, M., Heflin, J.: Partitioning OWL knowledge bases for parallel reasoning, pp. 108–115. IEEE, June 2014
6. Donini, F.M.: Complexity of reasoning. In: Baader, F., Calvanese, D., McGuinness, D.L., Nardi, D., Patel-Schneider, P.F. (eds.) The Description Logic Handbook: Theory, Implementation, and Applications, pp. 96–136. Cambridge University Press, New York (2003)
7. Glimm, B., Horrocks, I., Lutz, C., Sattler, U.: Conjunctive query answering for the description logic SHIQ. arXiv:1111.0049 [cs], October 2011
8. Xu, J., Shironoshita, P., Visser, U., John, N., Kabuka, M.: Extract ABox modules for efficient ontology querying. arXiv:1305.4859 [cs], May 2013

9. Xu, J., Shironoshita, P., Visser, U., John, N., Kabuka, M.: Module extraction for efficient object queries over ontologies with large ABoxes. AIA **2**(1), 8–31 (2015)
10. Wandelt, S., Möller, R.: Towards ABox modularization of semi-expressive description logics. Appl. Ontol. **7**(2), 133–167 (2012)
11. Xu, J., Shironoshita, P., Visser, U., John, N., Kabuka, M.: Converting instance checking to subsumption: a rethink for object queries over practical ontologies. Int. J. Intell. Sci. **05**(01), 44–62 (2015). arXiv:1412.7585
12. Nebel, B.: Reasoning and Revision in Hybrid Representation Systems. LNCS (LNAI), vol. 422. Springer, Heidelberg (1990). https://doi.org/10.1007/BFb0016445
13. Donini, F., Era, A.: Most specific concepts for knowledge bases with incomplete information. In: Proceedings of CIKM, Baltimore, MD, USA, pp. 545–551, November 1992
14. Donini, F.M., Lenzerini, M., Nardi, D., Schaerf, A.: Deduction in concept languages: from subsumption to instance checking. J. Logic Comput. **4**(4), 423–452 (1994)
15. Baader, F., Calvanese, D., McGuinness, D.L., Nardi, D., Patel-Schneider, P.F. (eds.): The Description Logic Handbook: Theory, Implementation, and Applications. Cambridge University Press, New York (2003)
16. Horrocks, I., Tessaris, S.: A conjunctive query language for description logic ABoxes. In: AAAI/IAAI, pp. 399–404 (2000)
17. Schaerf, A.: Reasoning with individuals in concept languages. Data Knowl. Eng. **13**(2), 141–176 (1994)
18. Kollia, I., Glimm, B., Horrocks, I.: SPARQL query answering over OWL ontologies. In: Antoniou, G., Grobelnik, M., Simperl, E., Parsia, B., Plexousakis, D., De Leenheer, P., Pan, J. (eds.) ESWC 2011. LNCS, vol. 6643, pp. 382–396. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-21034-1_26
19. Jing, Y., Jeong, D., Baik, D.K.: SPARQL graph pattern rewriting for OWL-DL inference queries. Knowl. Inf. Syst. **20**(2), 243–262 (2009)
20. Sirin, E., Parsia, B.: SPARQL-DL: SPARQL query for OWL-DL. In: In 3rd OWL Experiences and Directions Workshop (OWLED-2007) (2007)
21. Myung, J., Yeon, J., Lee, S.: SPARQL basic graph pattern processing with iterative MapReduce. In: Proceedings of the 2010 Workshop on Massive Data Analytics on the Cloud, MDAC 2010, pp. 6:1–6:6. ACM, New York (2010)
22. Schätzle, A., Przyjaciel-Zablocki, M., Hornung, T., Lausen, G.: PigSPARQL: a SPARQL query processing baseline for big data. In: Proceedings of the 12th International Semantic Web Conference (Posters & Demonstrations Track), ISWC-PD 2013, Aachen, Germany, vol. 1035, pp. 241–244. CEUR-WS.org (2013)
23. Artale, A., Calvanese, D., Kontchakov, R., Zakharyaschev, M.: The DL-lite family and relations. J. Artif. Int. Res. **36**(1), 1–69 (2009)
24. Ma, L., Yang, Y., Qiu, Z., Xie, G., Pan, Y., Liu, S.: Towards a complete OWL ontology benchmark. In: Sure, Y., Domingue, J. (eds.) ESWC 2006. LNCS, vol. 4011, pp. 125–139. Springer, Heidelberg (2006). https://doi.org/10.1007/11762256_12
25. Guo, Y., Pan, Z., Heflin, J.: LUBM: a benchmark for OWL knowledge base systems. Web Semant. **3**(2–3), 158–182 (2005)
26. W3C: Large Triple Stores - W3c Wiki (2015)