Reflections on: Reasoning and Querying Web-scale Open Data based on DL-Lite$_\mathcal{A}$ in a Divide-and-Conquer Way

Zhenzhen Gu[1], Songmao Zhang[2], and Cungen Cao[1]

[1] Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China
[2] Academy of Mathematics and Systems Sciences, Chinese Academy of Sciences, Beijing, China
guzhenzhen0720@163.com smzhang@math.ac.cn cgcao@ict.ac.cn

1 Introduction

Problem Description. Although the sheer size of the open datasets [2] in Linked Open Data (LOD) [3] can be viewed as a positive sign for Semantic Web initiatives, it causes performance bottlenecks for systems and tools that provide data managing and query answering services over the LOD datasets. Moreover, as the growth of the schema knowledge described by OWL, realizing efficient reasoning and expressive query answering has become a challenging task for both data publishers and end users. Most of the studies in this respect (for example, [4–6]) focused on dealing with the scalability issue whereas inferring knowledge through reasoning is often ignored. Moreover, when reasoning is considered, most of the time only lightweight RDFS reasoning can be supported.

Proposal of DL-Lite$_\mathcal{A}$. Motivated by a statistic analysis of the BTC 2012 dataset[3], we propose to use DL-Lite$_\mathcal{A}$ [10] techniques to reason and query the web-scale open datasets (knowledge bases) described by Semantic Web standards like RDF and OWL due to the low reasoning complexity and suitable expressivity of the language. The distinguished feature of DL-Lite$_\mathcal{A}$ is FOL-rewritability. This means that in DL-Lite$_\mathcal{A}$, by schema knowledge reasoning, both satisfiability checking and conjunctive query (CQ) answering can be eventually reduced to answering queries over the data layers of the corresponding knowledge bases (KBs). Moreover the separation between schema knowledge and data layer reasoning makes it possible to use highly optimized database management systems and engines in practice for query answering.

Bottleneck of DL-Lite$_\mathcal{A}$. Unfortunately, even with low reasoning complexity, when facing the real-life scalability challenge, the actual reasoning and CQ answering realized by DL-Lite$_\mathcal{A}$ may become infeasible by the following two factors. Firstly, for both satisfiability checking and CQ answering, a polynomial size of queries may need to be answered over the data layers of the corresponding KBs w.r.t. the size of the schema knowledge of these KBs. Secondly, for the KBs with massive individual assertions, evaluating a single query over the data layers may be highly time-consuming. The combination of these two factors makes

[3] https://km.aifb.kit.edu/projects/btc-2012/

the situation even worse, calling for new techniques for DL-Lite$_\mathcal{A}$ to tackle the scalability and efficiency challenge.

Our Solution–Divide-and-Conquer. In computer science, divide-and-conquer is the basis of efficient algorithms for all kinds of problems. This impels us to seek for a divide-and-conquer reasoning and query answering approach for DL-Lite$_\mathcal{A}$, with the basic idea of partitioning both KBs and CQs into smaller chunks and decomposing the original reasoning and query answering tasks into a group of independent sub-tasks such that the overall performance can be improved by taking advantage of the parallelization and distribution techniques.

The challenge for designing such an approach lies in how to carry out KB and CQ partitioning and reasoning reduction in a sound and complete way. Motivated by hash partitioning of RDF graphs (the partitions with the local feature of star-query answering) [15], we expect the smaller KB chunks to have the local feature for both satisfiability checking and simple-query answering. Here ***simple-queries*** (SQs) are the CQs whose query atoms share a common variable or individual. For CQ answering, we expect to partition a CQ into smaller SQs and evaluate them over smaller KB chunks.

Under these expectations, our divide-and-conquer approach is constructed from both theoretical and practical perspectives. Theoretically, definitions of KB partitions and CQ partitions are presented and the sufficient and necessary conditions are identified to determine whether a KB partition holds the desired features and whether a CQ can be answered in the desired way. Practically, based on the theoretical results, the concrete ways of partitioning KBs and CQs and evaluating CQ partitions over KB partitions are described. Moreover, a strategy of optimizing the procedure of evaluating CQ partitions over KB partitions is provided to improve the overall query answering performance. To verify our proposal and approach, experiments are conducted on two web-scale open datasets, i.e., DBpedia and BTC 2012 dataset, and the testing results indicate that the proposed approach opens new possibilities for realizing performance-critical applications on the Web with both expressivity and high scalability.

To the best of our knowledge, this is the first study on partitioning DL-Lite$_\mathcal{A}$ KBs and CQs for the purpose of efficiently reasoning with and querying large-scale datasets. We assume that the readers are familiar with DL-Lite$_\mathcal{A}$ and CQs. Next, we illustrate our main scientific contributions, containing KB partitioning, query partitioning, optimization and experiments. The technique details can be seen in [1]. In this paper, for a CQ Q and DL-Lite$_\mathcal{A}$ KB $\mathcal{K}$, we use $\mathsf{Ans}(Q, \mathcal{K})$ to denote the set of all the certain answers of Q over $\mathcal{K}$. Moreover, we use $\mathsf{Hd}(Q)$ and $\mathsf{Bd}(Q)$ to denote the head and body of Q, respectively. For two tuples $\boldsymbol{x}$ and $\boldsymbol{u}$ with the same dimension, we use $[\boldsymbol{x}/\boldsymbol{u}]$ to denote a substitution.

2 DL-Lite$_\mathcal{A}$ KB partitioning

Here, we discuss partitioning DL-Lite$_\mathcal{A}$ KBs into small chunks with the desired feature, containing the formalization of KB partitions, the conditions of detecting the desired KB partitions and the concrete way of computing such KB partitions.

Formalizing KB partitions. Before defining DL-Lite$_\mathcal{A}$ KB partitions, we first provide a way of computing a smaller sub-TBox for a given sub-ABox.

Procedure 1. For a DL-Lite$_\mathcal{A}$ KB $(\mathcal{T},\mathcal{A})$ and sub-ABox $\mathcal{A}'\subseteq\mathcal{A}$, we use $\mathcal{T}|_{\mathcal{A}'}$ to denote a sub-TBox of $\mathcal{T}$ constructed by the steps 1–2. Let $\mathcal{T}|_{\mathcal{A}'}=\emptyset$. Then:

Step 1. For each $\alpha\in\mathcal{T}$, add α to $\mathcal{T}|_{\mathcal{A}'}$ if α has the form $\gamma\sqsubseteq\neg\eta$ or $\mathsf{Fun}(S)$ and each name in α occurs in $(\mathcal{T}|_{\mathcal{A}'},\mathcal{A}')$, or α has the form $\gamma\sqsubseteq\eta$ and the name in γ occurs in $(\mathcal{T}|_{\mathcal{A}'},\mathcal{A}')$;

Step 2. Iterate Step 1, until $\mathcal{T}|_{\mathcal{A}'}$ do not change anymore.

Since for a sub-ABox, not all the axioms are relevant in terms of satisfiability checking and CQ answering, and smaller TBoxes can make the considered reasoning tasks be realized more quickly. The correctness is shown below:

Lemma 1. *For a DL-Lite$_\mathcal{A}$ KB $(\mathcal{T},\mathcal{A})$ and $\mathcal{A}'\subseteq\mathcal{A}$, (1) $(\mathcal{T}|_{\mathcal{A}'},\mathcal{A}')$ is satisfiable iff $(\mathcal{T},\mathcal{A}')$ is satisfiable; and (2) $\mathsf{Ans}(Q,(\mathcal{T}|_{\mathcal{A}'},\mathcal{A}'))=\mathsf{Ans}(Q,(\mathcal{T},\mathcal{A}'))$ for CQ Q.*

By Lemma 1, partitions of DL-Lite$_\mathcal{A}$ KBs are defined in the definition below.

Definition 1. *For DL-Lite$_\mathcal{A}$ KB $\mathcal{K}=(\mathcal{T},\mathcal{A})$, the set $\mathcal{S}=\cup_{i=1}^{n}\{(\mathcal{T}_i,\mathcal{A}_i)\}$ of DL-Lite$_\mathcal{A}$ KBs is called a partition of $\mathcal{K}$ if $\mathcal{A}=\cup_{i=1}^{n}\mathcal{A}_i$ and $\mathcal{T}_i=\mathcal{T}|_{\mathcal{A}_i}$. $\mathcal{S}$ is called a SCSQA-local partition of $\mathcal{K}$ iff (1) $\mathcal{K}$ is satisfiable iff each KB in $\mathcal{S}$ is satisfiable; and (2) if $\mathcal{K}$ is satisfiable then $\mathsf{Ans}(Q,\mathcal{K})=\cup_{\mathcal{K}'\in\mathcal{S}}\mathsf{Ans}(Q,\mathcal{K}')$ for each SQ Q.*

Detecting SCSQA-local partitions. By analyzing the algorithms of satisfiability checking and CQ rewriting [10], we found that in DL-Lite$_\mathcal{A}$, both satisfiability checking and SQ answering can be reduced to answering SQs over the ABoxes of the corresponding KBs, shown in the lemma below.

Lemma 2. *For a DL-Lite$_\mathcal{A}$ KB $\mathcal{K}=(\mathcal{T},\mathcal{A})$, (1) there exists a set $\mathcal{Q}$ of SQs so that $\mathcal{K}$ is satisfiable iff $\cup_{Q\in\mathcal{Q}}\mathsf{Ans}(Q,(\emptyset,\mathcal{A})=\emptyset$; and (2) if $\mathcal{K}$ is satisfiable then for each SQ Q, there exists a set $\mathcal{Q}$ of SQs so that $\mathsf{Ans}(Q,\mathcal{K})=\cup_{Q'\in\mathcal{Q}}\mathsf{Ans}(Q',(\emptyset,\mathcal{A}))$.*

All the query atoms in a SQ share a common individual or variable. Therefore, by Lemma 2, we can get that in the scenario of KB partitioning, for the local feature of satisfiability checking and SQ answering, the assertions in the original KB and sharing a common individual should be put in an identical sub-KB.

Proposition 1. *For DL-Lite$_\mathcal{A}$ KB $\mathcal{K}=(\mathcal{T},\mathcal{A})$ and partition $\mathcal{S}$ of $\mathcal{K}$, if for each $U\subseteq\mathcal{A}$ satisfying that all the assertions in U share a common individual, there exists $(\mathcal{T}',\mathcal{A}')\in\mathcal{S}$ such that $U\subseteq\mathcal{A}'$, then $\mathcal{S}$ is a SCSQA-local partition of $\mathcal{K}$.*

Without redundant individual assertions [1] (Theorem 1), the condition in Proposition 1 can be used as a necessary and sufficient condition.

Computing SCSQA-local partitions. By Proposition 1, SCSQA-local partitions of DL-Lite$_\mathcal{A}$ KBs can be obtained by the following procedure.

Procedure 2. For a DL-Lite$_\mathcal{A}$ KB $\mathcal{K}=(\mathcal{T},\mathcal{A})$, a SCSQA-local partition $\mathcal{S}$ of $\mathcal{K}$ can be obtained by the following steps 1–2:

Step 1. Compute a partition $\mathcal{A}_1,\cdots,\mathcal{A}_n$ of $\mathcal{A}$ by the following two sub-steps a–b:
   a. Compute the set $\mathsf{U}=\{\mathrm{U}_a|a\in\mathsf{I}\}$ where I is the set of all the individuals of $\mathcal{A}$, and U_a is the set consisting of all the assertions in $\mathcal{A}$ that contain a.
   b. Compute a partition $\mathsf{U}_1,\cdots,\mathsf{U}_n$ of U. For each U_i, let $\mathcal{A}_i=\cup_{e\in\mathsf{U}_i}e$.

Step 2. For each $1\leq i\leq n$, compute the TBox $\mathcal{T}|_{\mathcal{A}_i}$ for $\mathcal{A}_i$. Let $\mathcal{S}=\cup_{i=1}^{n}\{(\mathcal{T}|_{\mathcal{A}_i},\mathcal{A}_i)\}$.

Different ways of partitioning U in step 1.b lead to different SCSQA-local partitions of $\mathcal{K}$. In terms of distribution and parallelization, the partitions generating more balanced, smaller sized sub-ABoxes are more likely to achieve the efficiency improvement. For a given size *asize* of ABoxes in KB partition, a partition of U generating sub-ABoxes sized no more than *asize* can be obtained by:

b1. Compute an ordered tuple $\mathsf{O} = \mathrm{U}_1, \cdots, \mathrm{U}_m$ of the elements in U such that $|\mathrm{U}_i| \leq |\mathrm{U}_{i+1}|$ for $1 \leq i \leq m-1$;

b2. Continue the following procedure P, until O does not contain any element.

P. Generate a sub-ABox $\mathcal{A}'$. Let $\mathcal{A}' = \emptyset$. For i from $|\mathsf{O}|$ to 1, if $|\mathcal{A}' \cup \mathsf{O}[i]| \leq$ *asize* then let $\mathcal{A}' = \mathcal{A}' \cup \mathsf{O}[i]$; otherwise for j from 1 to $i-1$, if $|\mathcal{A}' \cup \mathsf{O}[j]| \leq$ *asize* then let $\mathcal{A}' = \mathcal{A}' \cup \mathsf{O}[j]$, otherwise break the first loop. Remove all the elements used to generate $\mathcal{A}'$ from O.

The complexity of computing SCSQA-local partitions of DL-Lite$_\mathcal{A}$ KBs is shown below. Low complexity makes the proposed approach appealing for partitioning a large-scale KB into smaller, independent sub-KBs with the desired local feature.

Theorem 1. *For a KB $\mathcal{K} = (\mathcal{T}, \mathcal{A})$, a SCSQA-local partition of $\mathcal{K}$ can be computed in $O(|\mathcal{T}|^2)$ w.r.t. $|\mathcal{T}|$ and in $O(|\mathcal{A}|)$ w.r.t. $|\mathcal{A}|$.*

3 Conjunctive query partitioning and evaluation

Here, we discuss CQ partitioning and evaluation, containing the formalization of CQ partitions, the conditions of detecting CQ partitions and the concrete way of computing and evaluating CQ partitions.

Formalizing CQ partitions. CQ partitions are defined in the definition below.

Definition 2. *For a CQ Q, we say the set $\mathcal{Q} = \cup_{i=1}^{n}\{q_i\}$ of CQs is a partition of Q if (1) $\mathsf{Bd}(Q) = \uplus_{i=1}^{n}\mathsf{Bd}(q_i)$ and (2) for each $1 \leq i \leq n$, $\mathsf{Hd}(q_i)$ consists of all the variables in $\mathsf{Bd}(q_i)$ that also occur either in $\mathsf{Hd}(Q)$ or $\cup_{k=1,k\neq i}^{n}\mathsf{Bd}(q_k)$. We say that $\mathcal{Q}$ is a SQ-partition of Q iff all the queries in $\mathcal{Q}$ are SQs.*

The concrete way of merging the certain answers of the sub-queries and the correctness are shown in the following definition and lemma, respectively.

Definition 3. *For a partition $\mathcal{Q} = \cup_{i=1}^{n}\{q_i\}$ of a CQ Q and a partition $\mathcal{S}$ of a DL-Lite$_\mathcal{A}$ KB $\mathcal{K}$, we define:*

$$\mathsf{Ans}(Q, \mathcal{Q}, \mathcal{S}) = \{\boldsymbol{x}[\boldsymbol{x}_1/\boldsymbol{u}_1, \cdots, \boldsymbol{x}_n/\boldsymbol{u}_n] | (\boldsymbol{u}_1, \cdots, \boldsymbol{u}_n) \in \Lambda_1 \times \cdots \times \Lambda_n\}$$

where $\boldsymbol{x} = \mathsf{Hd}(Q)$, for each $1 \leq i \leq n$, $\boldsymbol{x}_i = \mathsf{Hd}(q_i)$ and $\Lambda_i = \cup_{\mathcal{K}' \in \mathcal{S}}\mathsf{Ans}(q_i, \mathcal{K}')$, and for each $1 \leq i, j \leq n$, $1 \leq k \leq |\boldsymbol{u}_i|$ and $1 \leq l \leq |\boldsymbol{u}_j|$, if $\boldsymbol{x}_i[k] = \boldsymbol{x}_j[l]$ then $\boldsymbol{u}_i[k] = \boldsymbol{u}_j[l]$.

Lemma 3. $\mathsf{Ans}(Q, \mathcal{Q}, \mathcal{S}) \subseteq \mathsf{Ans}(Q, \mathcal{K})$ *holds.*

By the semantics of CQs, we know that distinguished (dist.) variables and non-dist. variables of CQs [1] are interpreted in different ways. Based on this, we define the concept of SQ-reducibility and show that SQ-reducible CQs can be answered in the descried way, i.e., by first partitioning CQs into SQs and then answering the smaller SQs over SCSQA-local partitions of DL-Lite$_\mathcal{A}$ KBs.

Definition 4. *For a CQ Q, we say Q is SQ-reducible iff it has a SQ-partition $\mathcal{Q}$ such that $\mathsf{Hd}(q) \subseteq \mathsf{Q}$ for each $q \in \mathcal{Q}$.*

Lemma 4. *For a CQ Q, if Q is SQ-reducible then there exists a SQ-partition $\mathcal{Q}$ of Q such that for each satisfiable DL-Lite$_\mathcal{A}$ KB $\mathcal{K}$ and each SQSQA-local partition $\mathcal{S}$ of $\mathcal{K}$, $\mathsf{Ans}(Q,\mathcal{K}) = \mathsf{Ans}(Q,\mathcal{Q},\mathcal{S})$ holds.*

Detecting SQ-reducible CQs. Not all CQs are SQ-reducible. This means that there exist CQs Q and DL-Lite$_\mathcal{A}$ KBs $\mathcal{K}$ such that evaluating any SQ-partition of Q over any SCSQA-local partition of $\mathcal{K}$ cannot obtain all the certain answers of Q over $\mathcal{K}$. Thus in order to evaluate as many CQs in the desired way as possible, we need to provide a *iff* condition to decide whether a CQ is SQ-reducible.

Theorem 2. *A CQ Q is SQ-reducible iff there do not exist three query atoms $P(?x,?y)$, α and α' in Q such that $?x$ and $?y$ are non-dist. variables of Q, α contains $?x$ but not $?y$, and α' contains $?y$ but not $?x$.*

Theorem 3. *Detecting whether a CQ Q is SQ-reducible can be done in $O(|Q|)$.*

Next, we present the concrete ways of partitioning and evaluating SQ-reducible CQs and non-SQ-reducible CQs.

Partitioning and evaluating SQ-reducible CQs. A SQ-reducible CQ may have many SQ-partitions, so we need to first detect the SQ-partitions that can capture all the certain answers of CQs and then provide the procedure of computing the desired SQ-partitions and the complexity of computing such partitions.

Theorem 4. *For a SQ-reducible CQ Q, let $\mathcal{Q}$ be any SQ-partition of Q such that $\mathsf{Hd}(q) \subseteq \mathsf{Hd}(Q)$ for each $q \in \mathcal{Q}$. Then $\mathsf{Ans}(Q,\mathcal{K}) = \mathsf{Ans}(Q,\mathcal{Q},\mathcal{S})$ holds, for any satisfiable DL-Lite$_\mathcal{A}$ KB $\mathcal{K}$ and any SCSQA-local partition $\mathcal{S}$ of $\mathcal{K}$.*

Procedure 3. For a SQ-reducible CQ Q, a SQ-partition $\mathcal{Q}$ of Q satisfying that $\mathsf{Hd}(q) \subseteq \mathsf{Hd}(Q)$ for each $q \in \mathcal{Q}$ can be obtained by the following steps 1–2.

Step 1. Let $\mathsf{A} = \mathsf{Bd}(Q)$ and $\mathsf{U} = \emptyset$. Compute a partition U of $\mathsf{Bd}(Q)$ by the steps a-b.
   a. For each non-dist. variable x of Q, compute the set U_x consisting of all the atoms in A that contain x, and let $\mathsf{U} = \mathsf{U} \cup \{\mathrm{U}_x\}$ and $\mathsf{A} = \mathsf{A} - \mathrm{U}_x$. Then for every two different non-dist. variables x and y of Q that occur in a same atom in Q, replace U_x and U_y in U with $\mathrm{U}_x \cup \mathrm{U}_y$;
   b. For each dist. variable or individual x of Q, compute the set U_x consisting of all the atoms in A that contain x, and let $\mathsf{U} = \mathsf{U} \cup \{\mathrm{U}_x\}$ and $\mathsf{A} = \mathsf{A} - \mathrm{U}_x$.

Step 2. For $\mathrm{U} \in \mathsf{U}$, generate a CQ q such that $\mathsf{Bd}(q) = \mathrm{U}$ and $\mathsf{Hd}(q)$ consists of all the variables occurring in U that also occur in $\mathsf{Hd}(Q)$ or $\mathsf{Bd}(Q) - \mathrm{U}$, add q to $\mathcal{Q}$.

Theorem 5. *For a SQ-reducible CQ Q, a partition $\mathcal{Q}$ of Q satisfying $\mathsf{hd}(q) \subseteq \mathsf{hd}(Q)$ for each $q \in \mathcal{Q}$ can be obtained in $O(|Q|)$.*

Partitioning and evaluating non-SQ-reducible CQs. If a CQ is not SQ-reducible, then non-SQ-partitions need to be evaluated to obtain all the certain answers of this CQ. By the semantics of CQs, we find that if a certain answer of a CQ Q over a KB is obtained by binding some non-dist. variables NV to anonymous elements, then in the scenario of KB and query partitioning, this certain answer can solely be obtained by evaluating a partition of Q in which the sub-queries do not contain the variables in NV as dist. variables. The definition below illustrates the partitions that can capture such certain answers of Q.

Definition 5. *For a CQ Q and set NV of non-dist. variables of Q, we use $Q|_{SQ}^{NV}$ to denote a partition of Q satisfying that (1) each variable in NV solely occurs in one query in $\mathcal{Q}$, and (2) for each non-simple query $q \in \mathcal{Q}$ and for every two different atoms α and α' in q, there exists a sequence $\beta_1, \cdots, \beta_n$ of atoms in q such that $\beta_1 = \alpha$, $\beta_n = \alpha'$ and β_i and β_{i+1} share a variable in NV for $1 \leq i \leq n-1$.*

The concrete way of answering non-SQ-reducible CQs and the procedure of computing the desired partitions as well as the complexity are shown below.

Theorem 6. *Given a non-SQ-reducible CQ Q, for any satisfiable DL-Lite$_\mathcal{A}$ KB $\mathcal{K}$ and any SCSQA-local partition $\mathcal{S}$ of $\mathcal{K}$, $\mathsf{Ans}(Q, \mathcal{K}) = \cup_{NV' \in 2^{NV}} \mathsf{Ans}(Q, Q|_{SQ}^{NV'}, \mathcal{S})$ holds where NV is the set consisting of all the non-dist. variables of Q.*

Procedure 5. For a non-SQ-reducible CQ Q and set NV of some non-dist. variables of Q, the partition $Q|_{SQ}^{NV}$, can be obtained by the following two steps:

Step 1. Let $\mathsf{A} = \mathsf{Bd}(Q)$ and $\mathsf{U} = \emptyset$. Compute a partition U of $\mathsf{Bd}(Q)$ by the steps a–c.
   a. Compute an undirected graph G with node set A. For each $(\alpha, \alpha') \in (\mathsf{A})^2$, there exists an edge between α and α' iff they share a variable in NV;
   b. For each connected component l of G, generate an atom set U consisting of all the nodes in l, and set $\mathsf{U} = \mathsf{U} \cup \{\mathrm{U}\}$ and $\mathsf{A} = \mathsf{A} - \mathrm{U}$;
   c. For each variable or individual x occurring in A, compute a set U consisting of all the atoms in A that contain x, and set $\mathsf{U} = \mathsf{U} \cup \{\mathrm{U}\}$ and $\mathsf{A} = \mathsf{A} - \mathrm{U}$.

Step 2. For each $\mathrm{U} \in \mathsf{U}$, generate a CQ q so that $\mathsf{Bd}(q) = \mathrm{U}$ and $\mathsf{Hd}(q)$ consists of all the variables in U that also occur in $\mathsf{Hd}(Q)$ or $\mathsf{Bd}(Q) - \mathrm{U}$, and add q to $\mathcal{Q}$.

Theorem 7. *The complexity of computing the partition $Q|_{SQ}^{NV}$ of Q is $O(|Q|^2)$.*

4 Optimizing the procedure of query partition evaluation

Although query partitioning can reduce the final number of queries needed to be answered over the ABox of a KB significantly, a large amount of intermediate results generated by answering sub-queries may also slow down the overall query answering procedure. Thus we further provide an optimization strategy to reduce the intermediate results as many and as early as possible by transferring values between sub-queries, as shown in the following example and proposition.

Example 1. Let $\mathcal{S}$ be a partition of a KB. Consider the partition $\mathcal{Q}$ of a CQ Q:

$$Q\colon\ R(a, ?x) \wedge P(?x, ?y) \wedge S(?y, ?z) \rightarrow q(?x, ?y, ?z)$$
$$\mathcal{Q} = \{q_1\colon R(a, ?x) \wedge P(?x, ?y) \rightarrow q(?x, ?y),\quad q_2\colon S(?y, ?z) \rightarrow q(?y, ?z)\}$$

After answering q_1 over the KBs in $\mathcal{S}$, suppose we obtain the answer set $\Lambda = \cup_{i=1}^{3}\{(a_i, b_i)\}$. If we transfer the bindings of $?y$ to q_2 before it is evaluated, i.e., replacing answering q_2 with answering the CQs $\cup_{i=1}^{3}\{S(b_i, ?z) \rightarrow q(b_i, ?z)\}$, then we can avoid computing the answers of q_2 which do not bind $?y$ to b_1, b_2 or b_3.

Proposition 2. *For a partition $\mathcal{Q} = \cup_{i=1}^{n}\{q_i\}$ of a CQ Q and a partition $\mathcal{S}$ of a satisfiable DL-Lite$_\mathcal{A}$ KB, the following equation holds:*

$$\mathsf{Ans}(Q, \mathcal{Q}, \mathcal{S}) = \{\boldsymbol{x}[\boldsymbol{x}_1/\boldsymbol{u}_1, \cdots, \boldsymbol{x}_n/\boldsymbol{u}_n] | (\boldsymbol{u}_1, \cdots, \boldsymbol{u}_n) \in \Lambda_1|_{\mathcal{B}_1} \times \cdots \times \Lambda_n|_{\mathcal{B}_n}\}$$

where $\boldsymbol{x} = \mathsf{Hd}(Q)$, for $1 \leq i \leq n$, $\boldsymbol{x}_i = \mathsf{Hd}(q_i)$, $\mathcal{B}_1 = \emptyset$ and for $1 < k \leq n$:

$$\mathcal{B}_k = \{[\boldsymbol{x}_1/\boldsymbol{u}_1, \cdots, \boldsymbol{x}_{k-1}/\boldsymbol{u}_{k-1}]|_{\boldsymbol{x}_k} | (\boldsymbol{u}_1, \cdots, \boldsymbol{u}_{k-1}) \in \Lambda_1|_{\mathcal{B}_1} \times \cdots \times \Lambda_{k-1}|_{\mathcal{B}_{k-1}}\}$$

where $\Lambda_1|_{\mathcal{B}_1} = \cup_{\mathcal{K} \in \mathcal{S}} \mathsf{Ans}(q_1, \mathcal{K})$ and for $1 < j \leq n$, $\Lambda_j|_{\mathcal{B}_j} = \cup_{\mathcal{K} \in \mathcal{S}} \cup_{\xi \in \mathcal{B}_j} \mathsf{Ans}(q_j\xi, \mathcal{K})$.

In Proposition 2, $\mathcal{B}_k$ denotes the set of the bindings of some variables of q_k obtained from the certain answers of the sub-queries evaluated before q_k. By observing the CQ rewriting procedure in DL-Lite$_\mathcal{A}$, we find that the transferred variable bindings do not affect the query rewriting procedure. This means that when answering a sub-query q over a sub-KB $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ with a set $\mathcal{B}$ of variable bindings, the CQs obtained by first transferring the bindings in $\mathcal{B}$ to q and then rewriting the resultant queries over $\mathcal{T}$ are equivalent to the CQs obtained by first rewriting q over $\mathcal{T}$ and then transferring the bindings in $\mathcal{B}$ to the rewritten queries. Thus we actually need one time of query rewriting rather than $|\mathcal{B}|$. Hence the procedure of evaluating the partitions of queries can be further optimized.

Proposition 3. *For a CQ q, satisfiable DL-Lite$_\mathcal{A}$ KB $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ and set $\mathcal{B}$ of functions ξ that maps some variables of q to names, then we can get that:*

$$\cup_{\xi\in\mathcal{B}}\texttt{PerfectRef}(q\xi, \mathcal{T}) = \cup_{\xi\in\mathcal{B}}\{q'\xi | q' \in \texttt{PerfectRef}(q, \mathcal{T})\}$$
$$\cup_{\xi\in\mathcal{B}}\mathsf{Ans}(q\xi, \mathcal{K}) = \cup_{q'\in\mathsf{PerfectRef}(q,\mathcal{T})} \cup_{\xi\in\mathcal{B}} \mathsf{Ans}(q'\xi, (\emptyset, \mathcal{A}))$$

where PerfectRef *denotes one classical query rewriting algorithm in DL-Lite$_\mathcal{A}$.*

Experiments presented in the subsequent section demonstrate that the value transferring strategy can significantly improve the query answering performance.

5 The evaluation

In order to demonstrate the efficiency and rationality of our approach and proposal, we have implemented a prototype system, where multi-thread techniques are adopted to check the satisfiability of and answer a single query over the sub-KBs in a KB partition in a parallel way. We run the system on two web-scale, real-world datasets, i.e., DBpedia_200 dataset extended with DBpedia ontology and BTC 2012 dataset. Next, we present the main results of the experiments.

SCSQA-local partition computation. After materializing the name equivalence relations, two DL-Lite$_\mathcal{A}$ KBs $\mathcal{K}_{dbp}$ and $\mathcal{K}_{btc}$ are constructed, respectively with 233,227 axioms and 0.2 billion individual assertions and 339,519 axioms and 1.04 billion individual assertions. We randomly choose two numbers, 28 million and 52 million, to respectively limit the size of the sub-ABoxes in KB partitions. Then, a SCSQA-local partition $\mathcal{S}_{dbp}$ of $\mathcal{K}_{dbp}$ with 14 sub-KBs and a SCSQA-local partition $\mathcal{S}_{btc}$ of $\mathcal{K}_{btc}$ with 30 sub-KBs are computed by ***Procedure 2***.

Satisfiability checking. Without KB partition, checking the satisfiability of $\mathcal{K}_{dbp}$ totally cost 6 minutes, and for $\mathcal{K}_{btc}$, after 6 hours an out-of-memory error was reported caused by the failure in completing generating the queries needed to be evaluated over its ABox. However, facilitated by KB partitioning equipped with parallelization, satisfiability checking of $\mathcal{K}_{dbp}$ and $\mathcal{K}_{btc}$ has been successfully done in 0.07 and 0.15 minutes, respectively. The testing results indicate that KB partitioning equipped with parallelization can significantly improve the performance of checking the satisfiability of web-scale DL-Lite$_\mathcal{A}$ KBs.

Query answering checking. In order to detect the effect of KB and query partitioning as well as sub-query value transfer on the performance of CQ answering, we design 14 CQs (8 SQs and 6 non-SQs) for $\mathcal{K}_{dbp}$ and $\mathcal{K}_{btc}$, respectively.

The SQ answering results indicate that no matter a SQ has a large or small number of rewritten queries over the original KB or it can be evaluated efficiently or inefficiently, in the situation of KB partitioning equipped with parallelization, this SQ can be answered in an extremely efficient way. On the other hand, the results of evaluating the tested non-SQ queries indicate that: (1) when a CQ cannot be evaluated over the original KB efficiently, query partitioning combined with KB partitioning and sub-query value transfer can make this query to be evaluated in a very efficient way; (2) no matter with or without KB partitioning, query partitioning can reduce the number of rewritten CQs significantly; (3) even when query partitioning cannot reduce the number of the rewritten queries, evaluating query partitions over KB partitions with sub-query value transfer can speed up the procedure; and (4) for the queries already efficiently answerable over the original KBs, our approach may not further improve the performance.

Comparing with related systems. To demonstrate the rationality and efficiency of our proposal and approach, we have compared our system with the two state-of-the-art, centralized triple stores, Jena-TDB and Virtuoso which support RDFS reasoning. The comparison results can be summarized as: (1) even with KB and query partitioning, our approach is not as efficient as the two RDF stores which evaluate queries directly without taking reasoning into consideration; (2) when reasoning is considered, the performance of the two triple stores drops significantly, and they do not perform as well as our approach; and (3) query answering without considering reasoning can miss many certain answers.

6 Comparison with related work

Compared with the works on querying LOD [4–9], our approach pursues both expressivity and scalability, capturing more schema knowledge and handling more expressive queries with the support of DL-Lite$_\mathcal{A}$ techniques. For scalability, we provide a divide-and-conquer reasoning and query answering approach for DL-Lite$_\mathcal{A}$ so that the massive knowledge in LOD can be managed by parallel and distribution techniques.

Compared with the works on DL KB modularity [12–14], we study a tractable language DL-Lite$_\mathcal{A}$, and concentrate on satisfiability checking and CQ answering rather than entailment checking. Moreover, we provide a way of ABox partitioning with linear data complexity where reasoning is not necessary. Compared with the works on RDF graph partitioning [15–17], we consider RDF graphs as DL-Lite$_\mathcal{A}$ KBs so that constraints in these graphs can be captured.

Compared with the works on query rewriting optimization [18–21], our approach features query rewriting optimization in DL-Lite$_\mathcal{A}$ from the distinctive perspective of query and KB partitioning, and the experiments on real-world datasets show that the number of rewritten queries can be significantly reduced.

Compared with the works [22, 23] on fast satisfiability checking, we propose to improve the performance in DL-Lite$_\mathcal{A}$ from the perspective of KB partitioning. Experiments on web-scale datasets demonstrate the efficiency of our approach.

Acknowledgments. This work has been supported by the National Key Research and Development Program of China under grants 2017YFC1700300 and 2017YFB1002300.

References

1. Gu, Z., Zhang, S., Cao, C,: Reasoning and Querying Web-scale Open Data based on DL-Lite$_{\mathcal{A}}$ in a Divide-and-Conquer Way. J. Web Semant. 55: 122-144 (2019).
2. O'Riain, S., Curry, E., Harth, A.: XBRL and open data for global financial ecosystems: A linked data approach. Int. J. Accounting Inf. Systems. 13 (2):141-162 (2012).
3. Bauer, F., Kaltenböck, M.: Linked open data: The essentials, Edition mono/monochrom, Vienna, Austria, 2011.
4. Bishop, B., Kiryakov, A., Ognyanov, D., Peikov, I., Tashev, Z., Velkov, R.: FactForge: A fast track to the web of Data. Semantic Web, 2 (2) (2011).
5. Schwarte, A., Haase, P., Hose, K., Schenkel, R., Schmidt, M.: FedX: Optimization Techniques for Federated Query Processing on Linked Data. In: ISWC, 2011.
6. Umbrich, J., Hogan, A., Polleres, A., Decker, S.: Link Traversal Querying for a diverse Web of Data. Semantic Web, 6 (6) (2012) 585-624.
7. Montoya, G., Skaf-Molli, H., Molli, P., Vidal, M. E.: Federated SPARQL Queries Processing with Replicated Fragments. In: ISWC, 2015.
8. Hartig, O., Freytag, J. C.: Foundations of traversal based query execution over Linked Data. In: ACM Hypertext, 2012.
9. Sabri, M. M.: A Hybrid Framework for Online Execution of Linked Data Queries. In: WWW, 2015.
10. Poggi, A., Lembo, L., Calvanese, D., De Giacomo, G., Lenzerini, M., Rosati, R.: Linking Data to Ontologies. J. Data Semantics. 10 (2008) 133-173.
11. Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., Rosati, R.: Tractable reasoning and efficient query answering in description logics: The DL-Lite family. J. Autom. Reasoning. 39 (3) (2007).
12. Grau, B. C., Horrocks, I., Kazakov, Y., Sattler, U.: Modular reuse of ontologies: theory and practice. J. Artif. Intell. Res. 31 (2008).
13. Botoeva, E., Konev, B., Lutz, C., Ryzhikov, V., Wolter, F., Zakharyaschev, M.: Inseparability and conservative extensions of description logic ontologies: A survey. In: Reasoning Wb, 2016.
14. Xu, J., Shironoshita, P., Visser, U., John, N., Kabuka, M.: Module Extraction for Efficient Object Queries over Ontologies with Large ABoxes. Artif Intell Appl. 2 (1) (2015).
15. Harbi, R., Abdelaziz, I., Kalnis, P., Mamoulis, N.: Evaluating SPARQL queries on massive RDF datasets. Proceedings of the VLDB Endowment, 8 (12) (2015).
16. Potter, A., Motik, B., Horrocks, I.: Querying Distributed RDF Graphs: The Effects of Partitioning. In: SSWS@ISWC, 2014.
17. Yang, S., Yan, X., Zong, B., Khan, A.: Towards effective partition management for large graphs. In: SIGMOD Conference, 2012.
18. Rosati, R., Almatelli, A.: Improving Query Answering over DL-Lite Ontologies. In: KR, 2010.
19. Venetis, T., Stoilos, G., Stamou, G. B.: Query extensions and incremental query rewriting for OWL 2 QL ontologies. J. Data Semantics, 3 (1) (2014).
20. Bursztyn, D., Goasdoué, F., Manolescu, I.: Teaching an RDBMS about ontological constraints. Proceedings of the VLDB Endowment, 9 (12) (2016).
21. Gottlob, G., Orsi, G., Pieris, A.: Query rewriting and optimization for ontological databases. ACM Trans. Database Syst. 39 (3) (2014).
22. Paulheim, H., Stuckenschmidt, H.: Fast Approximate A-Box Consistency Checking Using Machine Learning. In: ESWC, 2016.
23. Fokoue, A., Kershenbaum, A., Ma, L., Schonberg, E., Srinivas, K.: The Summary Abox: Cutting Ontologies Down to Size. In: ISWC, 2006.