ISLANDS AND QUERY ANSWERING FOR ALCHI-ONTOLOGIES

Sebastian Wandelt and Ralf Möller

Hamburg University of Technology, Institute for Software Systems,
Schwarzenbergstr. 95, 21073 Hamburg, Germany
`wandelt@tuhh.de,r.f.moeller@tuhh.de`
`http://www.sts.tu-harburg.de`

Abstract. The vision of the Semantic Web fostered the interest in reasoning over ever larger sets of assertional statements in ontologies. Today, real-world ontologies do not fit into main memory anymore and therefore tableaux-based reasoning systems cannot handle these large ontologies any longer.
We propose strategies to overcome this problem by performing query answering for an ontology over (usually small) relevant subsets of assertional axioms, called islands. These islands are computed based on a partitioning-criteria. We propose a way to preserve the partitions while updating an ontology and thus enable stream like reasoning for description logic ontologies. Furthermore, we explain how islands can be used to answer grounded conjunctive queries for description logic ontologies. We think that our proposal can support description logic systems to deal with the upcoming large amounts of fluctuant assertional data.

Key words: Description Logics, Reasoning, Scalability, Partitioning

1 Introduction

As the Semantic Web evolves, scalability of inference techniques becomes increasingly important. Even for basic description logic-based inference techniques, e.g. instance checking, it is only recently understood on how to perform reasoning on large ABoxes in an efficient way. This is not yet the case for problems that are too large to fit into main memory.

In this paper we present an approach to execute efficient retrieval tests on ontologies, which do not fit into main memory. Existing tableau-based description logic reasoning systems, e.g. Racer [HM01], do not perform well in such scenarios since the implementation of tableau-algorithms is usually built based on efficient in-memory structures. Our contribution is concerned with the following main objective: we want to partition the assertional part of an $\mathcal{ALCHI}$-ontology to more efficiently answer queries over partitions, instead of the complete ABox. The idea is to split up redundant/unimportant role assertions and then partition the ABox based on individual connectedness.

Moreover, we focus on the problem of updating ontologies. The idea is that a partitioning does not need to be computed from the scratch whenever the

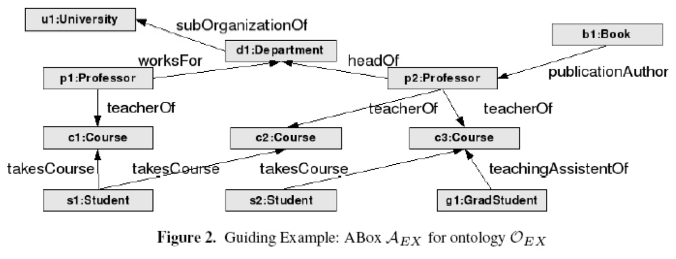

Fig. 1. Guiding Example: ABox $\mathcal{A}_{EX}$ for ontology $\mathcal{O}_{EX}$

underlying ontology is changed. To solve that, we propose partitioning-preserving transformations for each possible syntactic update of an ontology (terminological and assertional updates). We are convinced that such an incremental approach is crucial to enable stream-like processing of ontologies.

We propose was to handle common kinds of queries over description logic ontologies, i.e. instance checking, instance retrieval and grounded conjunctive queries.

The remaining parts of the paper are structured as follows. Section 2 introduces necessary formal notions and gives an overview over Related Work. In Section 3 we introduce the underlying partitioning algorithm, and propose our partitioning-preserving transformations in Section 4 (assertional updates) and in Section 5 (terminological updates). We present our preliminary implementation and evaluation in Section 6. In Section **??**, we give insights on query answering over partitionings. The paper is concluded in Section 8.

2 Foundations

2.1 Description Logic $\mathcal{ALCHI}$

We briefly recall syntax and semantics of the description logic $\mathcal{ALCHI}$. For the details, please refer to [BCM+07]. We assume a collection of disjoint sets: a set of *concept names* N_{CN} , a set of *role names* N_{RN} and a set of *individual names* N_I. The *set of roles* N_R is $N_{RN} \cup \{R^- | R \in N_{RN}\}$. The set of $\mathcal{ALCHI}$*-concept descriptions* is given by the following grammar:

$$C, D ::= \top | \bot | A | \neg C | C \sqcap D | C \sqcup D | \forall R.C | \exists R.C$$

where $A \in N_{CN}$ and $R \in N_R$. With N_C we denote all *atomic concepts*, i.e. concept descriptions which are concept names. For the semantics please refer to [BCM+07].

A *TBox* is a set of so-called *generalized concept inclusions*(GCIs) $C \sqsubseteq D$. A *RBox* is a set of so-called *role inclusions* $R \sqsubseteq S$. An *ABox* is a set of so-called *concept and role assertions* $a : C$ and $R(a, b)$. A *ontology* $\mathcal{O}$ consists of a 3-tuple $\langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle$, where $\mathcal{T}$ is a TBox, $\mathcal{R}$ is a RBox and $\mathcal{A}$ is a ABox. We restrict the concept assertions in $\mathcal{A}$ in such a way that each concept description is an atomic concept or a negated atomic concept. This is a common assumption, e.g. in [GH06], when dealing with large assertional datasets in ontologies.

In the following we define an example ontology, which is used throughout the remaining part of the paper. The ontology is inspired by LUBM [GPH05], a benchmark-ontology in the setting of universities. Although this is a synthetic benchmark, several (if not most) papers on scalability of ontological reasoning consider it as a base reference. We take a particular snapshot from the LUBM-ontology (TBox, RBox and ABox) and adapt it for presentation purposes. Please note that we do not claim that our snapshot is representative for LUBM.

Example 21 *Let* $\mathcal{O}_{EX} = \langle \mathcal{T}_{EX}, \mathcal{R}_{EX}, \mathcal{A}_{EX} \rangle$*, s.t.*

$$
\begin{aligned}
\mathcal{T}_{EX} = \{ & \\
& Chair \equiv \exists headOf.Department \sqcap Person, Professor \sqsubseteq Faculty, \\
& Book \sqsubseteq Publication, \\
& GraduateStudent \sqsubseteq Student, Student \equiv Person \sqcap \exists takesCourse.Course, \\
& \top \sqsubseteq \forall teacherOf.Course, \exists teacherOf.\top \sqsubseteq Faculty, Faculty \sqsubseteq Person, \\
& \top \sqsubseteq \forall publicationAuthor^{-}.(Book \sqcup ConferencePaper) \\
& \} \\
\mathcal{R}_{EX} = & \{headOf \sqsubseteq worksFor, worksFor \sqsubseteq memberOf, memberOf \doteq member^{-}\} \\
\mathcal{A}_{EX} = & \texttt{see Figure 1}
\end{aligned}
$$

2.2 Related Work

Referring to Example 21, different kinds of partitionings can be, informally, summarized as follows:

- Naive partitioning: This partitioning is done in existing reasoning systems. The idea is that individuals end up in the same partition, if there is a path of role assertions connecting them. Usually many individuals are connected to most other individuals in an ontology. This basic partitioning strategy is often not enough. In our LUBM-example there is only one partition, since each named individual is connected via a path to each other named individual.
- Extension in [GH06]: Since $suborganizationOf$ and $teachingAssistentOf$ are the only roles, which are not bound in a ∀-constraint in $\mathcal{T}_{EX}$ (please note that $takesCourse$ occurs indirectly in a ∀-constraint when the definition of student is split up into two inclusions), there are three partitions:
  1. one partition containing university $u1$,
  2. one partition containing graduate student $g1$ and
  3. one partition containing all remaining individuals

– Our proposal: a more fine-grained partitioning (details see below). For example, the only sub-concepts, which can be propagated over the role $teacherOf$ are $\bot$ and $Course$. Now, since for role assertion $teacherOf(p1, c1)$, $c1$ is an explicit instance of $Course$, i.e. the propagation is redundant, we can informally speaking "split up" the assertion to further increase granularity of connectedness-based partitioning.

There exists further related work on scalable reasoning. In [FKM+06], the authors suggest a scalable way to check consistency of ABoxes. The idea is to merge edges in an ABox whenever consistency is preserved. Their approach is query dependent and, informally speaking, orthogonal to partitioning approaches.

Several papers discuss the transformation of an ontology into datalog, e.g. [MOS+02], or the use of novel less-deterministic hypertableau algorithms[MSH07], to perform scalable reasoning. Furthermore, [SK04] suggests to partition the terminological part of an ontology, while we focus on the assertional part.

After all, we think that our work can be seen as complementary to other work, since it can be easily incorporated into existing algorithms. Furthermore we are unique in focusing on updating partitions to support stream-like processing.

3 Ontology Partitioning

We have initially proposed a method for role assertion separability checking in [WM08]. For completeness we start with one definition from [WM08]. The definition of $\mathcal{O}$-separability is used to determine the importance of role assertions in a given ABox. Informally speaking, the idea is that $\mathcal{O}$-separable assertions will never be used to propagate "complex and new information" (see below) via role assertions.

Definition 1. *Given an ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle$, a role assertion $R(a, b)$ is called* $\mathcal{O}$-separable*, if we have $\mathcal{O}$ is inconsistent $\iff \langle \mathcal{T}, \mathcal{R}, \mathcal{A}_2 \}\rangle$ is inconsistent, where*

$$\begin{aligned}\mathcal{A}_2 = \mathcal{A} \setminus \{R(a, b)\} \cup \{R(a, i_1), R(i_2, b)\} \cup \\ \{i_1 : C | b : C \in \mathcal{A}\} \cup \{i_2 : C | a : C \in \mathcal{A}\},\end{aligned}$$

s.t. i_1 and i_2 are fresh individual names.

Now, we further extend our proposal by partitioning-preserving update transformations. To do so, we define a notion of ABox and Ontology partitioning, which will be used in our update transformations below.

Definition 2. *Given an ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle$, an* ABox Partition *for $\mathcal{A}$ is a tuple $AP = \langle IN, S \rangle$ such that*

– *$IN \subseteq Inds(\mathcal{A})$ and*
– *$S = \{a : C | a \in M \wedge a : C \in \mathcal{A}\} \cup \{R(a, b) | (a \in IN \vee b \in IN) \wedge R(a, b) \in \mathcal{A}\}$, where $M = \{a | b \in IN \wedge (R(a, b) \in \mathcal{A} \vee R(b, a) \in \mathcal{A})\} \cup IN$*

We define two projection functions to obtain the first and the second element in a partition-pair: let $\pi_{IN}(AP) = IN$, and $\pi_S(AP) = S$. Informally speaking, an *ABox Partition* is composed of two components. The individual set IN, which contains the core individuals of the partition, and the assertion set S containing all the assertions needed in the partition. If a is an individual in IN, then S contains all the assertions involving a and all the concept assertions involving all direct neighbours of a.

Definition 3. *Given an ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle$, an* ABox Individual Partitioning *for $\mathcal{A}$ is a set $P = \{ap_1, .., ap_n\}$, such that each ap_i is an* ABox Partition *for $\mathcal{A}$ and*

1. *For each ap_i, ap_j, $(i \neq j)$ we have $\pi_{IN}(ap_i) \cap \pi_{IN}(ap_j) = \emptyset$*
2. *$Ind(\mathcal{A}) = \bigcup_{i=1..n} \pi_{IN}(ap_i)$*
3. *$\mathcal{A} = \bigcup_{i=1..n} \pi_S(ap_i)$*

The definition states that all the partitions have distinct core individual sets, the union of all the core individual sets of all the partitions is exactly the individual set of $\mathcal{A}$, and the union of all the assertion sets of all the partitions is the assertion set of $\mathcal{A}$.
Since each individual is assigned to only one ABox partition as a core individual, we define a function $\phi_P : Ind(\mathcal{A}) \rightarrow P$ that returns the partition for a given individual a. If $a \notin Ind(\mathcal{A})$, then $\phi_P(a) = \emptyset$. Next we will define the partitioning for the ontology.

Definition 4. *Given a consistent ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle$, an* Ontology Partitioning *for $\mathcal{O}$ is a structure $OP_{\mathcal{O}} = \langle \mathcal{T}, \mathcal{R}, P \rangle$, where P is an ABox Partitioning for $\mathcal{A}$ such that for each individual $a \in Ind(\mathcal{A})$ and each atomic concept C we have $\mathcal{O} \vDash a : C$ iff $\langle \mathcal{T}, \mathcal{R}, \pi_S(\phi_P(a)) \rangle \vDash a : C$.*

We use the $\mathcal{O}$-separability, see [WM08], of role assertions to determine the partitioning of $\mathcal{A}$. From the previous section, it holds that with the partitioning an ABox based on the $\mathcal{O}$-separability of role assertions, the instance checking problem can be solved with only one partition.

4 Updating the ABox

In this section, we will introduce means to preserve a partitioning of an ontology under Syntactic ABox Updates[HwPS06]. With syntactic updates, there is no consistency checking when adding a new assertion, and neither an enforcement of non-entailment when removing. However, syntactic updates are computationally easier to handle.

The general scenario for updating an ABox is as follows: We assume to start with an empty ontology (which has no assertions in the ABox), and its corresponding partitioning. Then we build up step by step the partitioned ontology by use of our update transformations.

For an empty ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{R}, \{\} \rangle$, the corresponding partitioning is $OP_{\mathcal{O}} = \langle \mathcal{T}, \mathcal{R}, P \rangle$ where $P = \{\langle \{\}, \{\} \rangle\}$. In the following we will use two update functions, *merge* and *reduce*, to implement our update transformations:

Definition 5. *The result of the* merge operation *on a set of ABox Partitions for $\mathcal{A}$, $Merge(\{ap_1, .., ap_n\})$, is defined as the ABox Partition ap for $\mathcal{A}$, s.t.*

$$ap = \langle \bigcup_{i \leq n} \pi_{IN}(ap_i), \bigcup_{i \leq n} \pi_S(ap_i) \rangle$$

Definition 6. *The result of the* reduce operation *on an ABox Partition for $\mathcal{A}$, $Reduce(pa)$, is defined as a set of ABox Partition $\{ap_1, .., ap_n\}$ built as follows:*

1. *For each $R(a,b) \in \pi_S(ap)$ do: if $R(a,b)$ is $\mathcal{O}$-separable, then replace $R(a,b)$ with $\{R(a,b*), R(a*,b)\} \cup \{a* : C | a : C \in \pi_S(ap)\} \cup \{b* : C | b : C \in \pi_S(ap)\}$, where $a*$ and $b*$ are fresh individual names for a and b.*
2. *Let $\{ap_1, .., ap_n\}$ be the disconnected partitions in ap.*
3. *Replace each $a*$ in each ap_i by a.*
4. *Replace each $b*$ in each ap_i by b.*

The *merge* operation simply merges all the core individual sets and the assertion sets of all the partitions. The *reduce* operation, in the other hand, divides an ABox Partition into smaller partitions based on $\mathcal{O}$-separability of role assertions.

The algorithm for updating ABoxes is illustrated in Figure 2. It can be informally summarized as follows:
Adding a role assertion $R(a,b)$: first we ensure that partitions exist for both a and b (if not, create a new partition). If a and b are in the same partition, then the role assertion is just simply added to the partition. If a and b are in two distinct partitions, and $R(a,b)$ is not $\mathcal{O}$-separable, then the two partitions are merged.
Removing a role assertion $R(a,b)$: if a and b are in different partitions, then the role assertion is just simply removed from both partitions. If a and b are in the same partition, then after removing the role assertion the partition needs to be rechecked to see if the removal of the role assertion causes the partition to be reduce-able.
Adding a concept assertion $C(a)$: first we ensure that partition exists for individual a. Then we add concept assertion $C(a)$ to the partition of a ($\phi_P(a)$), and all the partitions that contain any role assertion for a, to maintain the data consistency between partitions.
Removing a concept assertion $C(a)$: remove the concept assertion from all the partitions containing it. After that, all the role assertion involving a need to be $\mathcal{O}$-separability checked. If any of the role assertions becomes $\mathcal{O}$-inseparable due to the removal, then the corresponding partitions need to be merged.

5 Updating the TBox

In the following, we give a rough sketch of the update transformations. For details please refer to our technical report [Ngu09]. We extend the definition of

```
Adding a role assertion $R(a,b)$:

1. If $\phi_P(a) = \emptyset$ then add $\langle\{a\},\{R(a,b)\}\rangle$ to $P$
2. If $\phi_P(b) = \emptyset$ then add $\langle\{b\},\{R(a,b)\}\rangle$ to $P$
3. If $\phi_P(a) = \phi_P(b)$ then $\pi_S(\phi_P(a)) = \pi_S(\phi_P(a)) \cup \{R(a,b)\}$
4. Else If $R(a,b)$ is $\mathcal{O}$-separable w.r.t. $\pi_S(\phi_P(a))$ then
   (a) Add $R(a,b)$ to $\pi_S(\phi_P(a))$ and to $\pi_S(\phi_P(b))$
   (b) Add $\{b : C \mid b : C \in \pi_S(\phi_P(b))\}$ to $\pi_S(\phi_P(a))$
   (c) Add $\{a : C \mid a : C \in \pi_S(\phi_P(a))\}$ to $\pi_S(\phi_P(b))$
5. Else
   (a) Add $R(a,b)$ to $\pi_S(\phi_P(a))$
   (b) $P = P \setminus \{\phi_P(a), \phi_P(b)\} \cup Merge(\phi_P(a), \phi_P(b))$

Removing a role assertion $R(a,b)$:

1. If $\phi_P(a) \neq \phi_P(b)$ then
   (a) $\pi_S(\phi_P(a)) = \pi_S(\phi_P(a)) \setminus \{R(a,b)\}$
   (b) $\pi_S(\phi_P(b)) = \pi_S(\phi_P(b)) \setminus \{R(a,b)\}$
2. Else
   (a) If $R(a,b)$ was $\mathcal{O}$-separable w.r.t. $\pi_S(\phi_P(a))$ then
         $\pi_S(\phi_P(a)) = \pi_S(\phi_P(a)) \setminus \{R(a,b)\}$
         $\pi_S(\phi_P(b)) = \pi_S(\phi_P(b)) \setminus \{R(a,b)\}$
   (b) Else $P = P \setminus \{\phi_P(a), \phi_P(b)\} \cup Reduce(Merge(\phi_P(a), \phi_P(b)))$

Adding a concept assertion $a : C$:

1. If $\phi_P(a) = \emptyset$ then add $\langle\{a\},\{\}\rangle$ to $P$
2. $\pi_S(\phi_P(a)) = \pi_S(\phi_P(a)) \cup \{a : C\}$
3. For each $ap_i \in P$ do
     If $a \in Ind(\pi_S(ap_i))$ then $\pi_S(ap_i) = \pi_S(ap_i) \cup \{a : C\}$
4. $P = P \setminus \{\phi_P(a)\} \cup Reduce(\phi_P(a))$

Removing a concept assertion $a : C$:

1. $\pi_S(\phi_P(a)) = \pi_S(\phi_P(a)) \setminus \{a : C\}$
2. For each $ap_i \in P$ do
     – If $a \in Ind(\pi_S(ap_i))$ then $\pi_S(ap_i) = \pi_S(ap_i) \setminus \{a : C\}$
3. For each $R(a,b) \in \phi_P(a)$ do
     – If R(a,b) is not $\mathcal{O}$-separable, then $P = P \setminus \{\phi_P(a), \phi_P(b)\} \cup \{Merge(\phi_P(a), \phi_P(b))\}$
4. For each $R(b,a) \in \phi_P(a)$ do
     – If R(b,a) is not $\mathcal{O}$-separable, then $P = P \setminus \{\phi_P(b), \phi_P(a)\} \cup \{Merge(\phi_P(b), \phi_P(a))\}$
```

Fig. 2. Updating ABox

the ∀-info structure from [WM08], by introducing a *reduced* ∀-info structure and an *extended* ∀-info structure.

Definition 7. *A* reduced *∀-info structure for ontology $\mathcal{O}$ is a function $e^{\forall}_{\mathcal{O}}$ which is extend from ∀-info structure $f^{\forall}_{\mathcal{O}}$ such that for every role R:*

$$e^{\forall}_{\mathcal{O}}(R) = f^{\forall}_{\mathcal{O}}(R) \backslash \{C_k | \exists C \in f^{\forall}_{\mathcal{O}} : C \sqsubset C_k\}$$

Definition 8. *An* extended *∀-info structure for ontology $\mathcal{O}$ is a function $g^{\forall}_{\mathcal{O}}$ which is extended from reduced ∀-info structure $e^{\forall}_{\mathcal{O}}$ as following:*

– *If $e^{\forall}_{\mathcal{O}}(R) = *$ then $g^{\forall}_{\mathcal{O}}(R) = \{\langle *, * \rangle\}$*
– *Else If $e^{\forall}_{\mathcal{O}}(R) = \emptyset$ then $g^{\forall}_{\mathcal{O}}(R) = \{\langle \emptyset, \emptyset \rangle\}$*
– *Else $g^{\forall}_{\mathcal{O}}(R) = \{\langle C_i, Sub(C_i) \rangle\}$, with $C_i \in e^{\forall}_{\mathcal{O}}(R)$, and $Sub(C_i)$ is the set of all the concepts that C_i subsumes in the simple concept hierarchy H_S.*

We also denote $\pi_C(g^{\forall}_{\mathcal{O}}(R)) \equiv \{C_i\}$, the set of all C_i appears in $\{\langle C_i, Sub(C_i) \rangle\}$ (which is $e^{\forall}_{\mathcal{O}}(R)$); and $\pi_{Sub,C_i}(g^{\forall}_{\mathcal{O}}(R)) \equiv Sub(C_i)$.

Informally speaking, the reduced ∀-info structure contains only the bottom-most concepts of the concept hierarchy branches that appears in $f^{\forall}_{\mathcal{O}}$, w.r.t. the simple concept hierarchy. On the other hand, an entry in the extended ∀-info structure is a set, each element of which is a tuples of a concept in $e^{\forall}_{\mathcal{O}}$ and the set of all the children of that concept, w.r.t. the concept hierarchy.

Updating ABox assertions can lead to the merging/reducing involving one or two specific partitions identified by the individuals in the updated assertions, while updating in TBox and RBox rather causes the merging/reducing in many pairs of partitions involving a certain set of role names. More formally speaking, updating w.r.t TBox and RBox can affects a set of role U_R, such that for each $R \in U_R$, and all individual pairs $\{a, b\}, s.t.R(a, b) \in \mathcal{A}$, the status of the role assertion $R(a, b)$ might be changed ($\mathcal{O}$-separable to $\mathcal{O}$-inseparable or vice versa). We call this role set U_R the *changeable role set*, and each $R \in U_R$ *changeable role*.

We have derived the following algorithm for updating a TBox and a RBox:

- For each role R in new terminology $\mathcal{T}*$, calculate $g^{\forall}_{\mathcal{O}}(R)$ before updating and $g^{\forall}_{\mathcal{O}*}(R)$ after updating.
  - If$(g^{\forall}_{\mathcal{O}}(R) \neq g^{\forall}_{\mathcal{O}} * (R))$ then $U_R = U_R \cup R$
- For each $R \in U_R$, and for each $R(a, b)$:
  - If $R(a, b)$ is $\mathcal{O}$-separable but not $\mathcal{O}*$-separable then $P = P \backslash \{\phi_P(a), \phi_P(b)\} \cup Merge(\phi_P(a), \phi_P(b))$
  - If $R(a, b)$ is not $\mathcal{O}$-separable but $\mathcal{O}*$-separable then $P = P \backslash \phi_P(a) \cup Reduce(\phi_P(a))$

(*) $\mathcal{O}*$-separable is denoted for separable with respect to the new ontology (after update), while $\mathcal{O}$-separable is denoted for separable with respect to the old ontology.

In the following, we will consider specific cases of updating TBox, and the effects they make to the extended $\forall$-info structure, and by this, compute the changeable role set. Then, in case of a terminological update, we have to check all role assertions, whose role is an element of the changeable role set, for $\mathcal{O}$-separability.

5.1 Updating TBox - concept inclusions

Updating TBox by adding/removing a concept inclusion might causes changes to $g^{\forall}_{\mathcal{O}}$ because

- if the concept inclusion adds $A \sqsubseteq B$ to the Concept Hierarchy H_S, and since the extended $\forall$-info structure $g^{\forall}_{\mathcal{O}}$ is built based on H_S, there probably have changes in $g^{\forall}_{\mathcal{O}}$.
- if the SNF, see [WM08] for details, of the added concept inclusion contains one or more $\forall$-bound for a role R that did not exist in the old terminology (or does not exist in updated terminology in case of removing concept inclusion), then there is changes in the $\forall$-info structure of the terminology, which also probably causes changes in the extended $\forall$-info structure.

Thus, instead of recalculating the extend $\forall$-info structure, if we know that the update is of a concept inclusion, then we just need to extract the information from the added/removed concept inclusion itself to check if it will cause changes

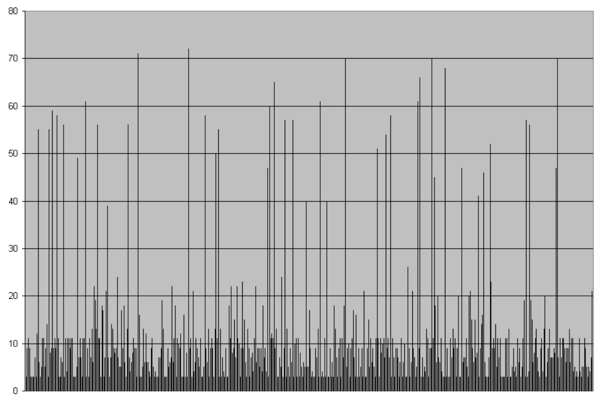

Fig. 3. Assertion distribution among partitions in node 1 (3 nodes)

in the $g^{\forall}_{\mathcal{O}}$.
Before go into details how to decide the update role set from the added/ removed concept inclusion, we introduce some useful definitions.

Definition 9. *A $\forall$-info structure for a concept inclusion $C \sqsubseteq D$ w.r.t $\mathcal{O}$, written as $f^{\forall}_{C\sqsubseteq D,\mathcal{O}}$, is a function that assigns to each role name R in $SNF(C \sqsubseteq D)$ one of the following entries:*

- *$\emptyset$ if we know that there is no $\forall$ constraint for R in $SNF(C \sqsubseteq D)$.*
- *a set S of atomic concept or negation atomic concept, s.t. there is no other than those in S that occurs $\forall$-bound on R in $SNF(C \sqsubseteq D)$.*
- *$*$, if there are arbitrary complex $\forall$ constraints on role R in $SNF(C \sqsubseteq D)$.*

This definition is literally similar to the definition of the $\forall$-info structure stated before, but for only one axiom. From this, we also define the *reduced $\forall$-info structure for a concept inclusion w.r.t. ontology $\mathcal{O}$* and *extended $\forall$-info structure for a concept inclusion w.r.t. ontology $\mathcal{O}$* in the same manner

Definition 10. *A* reduced *$\forall$-info structure for a concept inclusion $C \sqsubseteq D$ w.r.t. ontology $\mathcal{O}$ is a function $e^{\forall}_{C\sqsubseteq D,\mathcal{O}}$ which is extend from $\forall$-info structure $f^{\forall}_{C\sqsubseteq D,\mathcal{O}}$ such that for every role R:*

$$e^{\forall}_{C\sqsubseteq D,\mathcal{O}}(R) = f^{\forall}_{C\sqsubseteq D,\mathcal{O}}(R) \backslash \{C_k | \exists C \in f^{\forall}_{C\sqsubseteq D,\mathcal{O}} : C \sqsubset C_k\}$$

Definition 11. *An* extended *$\forall$-info structure for a concept inclusion $C \sqsubseteq D$ w.r.t. ontology $\mathcal{O}$ is a function $g^{\forall}_{C\sqsubseteq D,\mathcal{O}}$ which is extended from reduced $\forall$-info structure $e^{\forall}_{C\sqsubseteq D,\mathcal{O}}$ as following:*

- *If $e^{\forall}_{C\sqsubseteq D,\mathcal{O}}(R) = *$ then $g^{\forall}_{C\sqsubseteq D,\mathcal{O}}(R) = \{\langle *, * \rangle\}$*
- *Else If $e^{\forall}_{C\sqsubseteq D,\mathcal{O}}(R) = \emptyset$ then $g^{\forall}_{C\sqsubseteq D,\mathcal{O}}(R) = \{\langle \emptyset, \emptyset \rangle\}$*

Node	Total Partitions	Total Assertions	Assertions/partition	min	max
1	518	6089	11.7548	3	72
2	518	6822	13.1699	3	1596
3	518	5702	11.0077	3	77

Table 1. Partitions and assertions distribution among 3 nodes

– *Else* $g^{\forall}_{C\sqsubseteq D,\mathcal{O}}(R) = \{\langle C_i, Sub(C_i)\rangle\}$*, with* $C_i \in e^{\forall}_{C\sqsubseteq D,\mathcal{O}}(R)$*, and* $Sub(C_i)$ *is the set of all the concepts that* C_i *subsumes in the simple concept hierarchy* H_S.

And we have the following detailed algorithm for calculating the update role set in case of adding/removing a concept inclusion:

- Adding a concept inclusion $C \sqsubseteq D$
  - For each $A \sqsubseteq B$ that is added to the concept hierarchy:
    - for any role R that $B \in g^{\forall}_{\mathcal{O}}(R)$, $U_R = U_R \cup R$
  - For each R s.t. $g^{\forall}_{C\sqsubseteq D,\mathcal{O}*}(R) \neq \emptyset \wedge g^{\forall}_{C\sqsubseteq D,\mathcal{O}*}(R) \nsubseteq g^{\forall}_{\mathcal{O}}(R)$, $U_R = U_R \cup R$
- Removing a concept inclusion $C \sqsubseteq D$
  - For each $A \sqsubseteq B$ that is removed to the concept hierarchy:
    - for any role R that $B \in g^{\forall}_{\mathcal{O}}(R)$, $U_R = U_R \cup R$
  - For each R s.t. $g^{\forall}_{C\sqsubseteq D,\mathcal{O}*}(R) \neq \emptyset \wedge g^{\forall}_{C\sqsubseteq D,\mathcal{O}*}(R) \nsubseteq g^{\forall}_{\mathcal{O}*}(R)$, $U_R = U_R \cup R$

Here, we denote with $\mathcal{O}$ the ontology before updating and with $\mathcal{O}*$ the ontology after updating.

5.2 Updating RBox - role inclusions

Adding/removing a role inclusion has a quite obvious effect: it might change the role hierarchy. Since the ∀-info structure of the ontology is calculated using role taxonomy, this will change the ∀-info structure, and also the extended ∀-info structure. In the following, we present a way to determine the update role set

- Adding a role inclusion $R \sqsubseteq S$
  - if $g^{\forall}_{\mathcal{O}}(S) \nsubseteq g^{\forall}_{\mathcal{O}}(R)$ then for all sub role V of R ($V \sqsubseteq R$), $U_R = U_R \cup V$
- Removing a role inclusion $R \sqsubseteq S$
  - if $g^{\forall}_{\mathcal{O}}(S) \nsubseteq g^{\forall}_{\mathcal{O}*}(R)$ then for all sub role V of R ($V \sqsubseteq R$), $U_R = U_R \cup V$

5.3 Updating RBox - role inverses

Adding/removing a role inverse, on the other hand, might change the ∀-bound for both roles involving the inverse role. This causes the changes for the ∀-info structure of both roles, which also alters their extend ∀-info structure, thus we have following algorithm for calculating update role set:

Node	Total Partition	Total Assertion	Assertion/partition	min	max
1	260	2989	11.4962	3	70
2	259	4129	15.9421	3	1596
3	259	2864	11.0579	3	77
4	258	3100	12.0155	3	72
5	259	2693	10.3977	3	76
6	259	2838	10.9575	3	74

Table 2. Partitions and assertions distribution among 6 nodes

– Adding a role inverse pair $R = Inv(S)$
  - for all role $V \sqsubseteq R$, $U_R = U_R \cup V$
  - for all role $W \sqsubseteq S$, $U_R = U_R \cup W$

– Removing a role inverse pair $R = Inv(S)$
  - for all role $V \sqsubseteq R$, $U_R = U_R \cup V$
  - for all role $W \sqsubseteq S$, $U_R = U_R \cup W$

6 Distributed Storage System and Preliminary Evaluation

We have implemented the above algorithms in a Java program and performed initial tests on LUBM. The first test is composed of a server and 3 nodes. For the system performance, our test program was able to load 400-500 LUBM-ABox/TBox assertions per second. This is just an average value. From our experience, ABox assertions turn out to be loaded much faster, while TBox assertions slow the system down. The reasons for that behaviour have already been indicated above.

Besides system performance, another factor we want to evaluate is the distribution of the data among nodes. The data collected using three nodes is shown in Table 5.2. It is easy to see that the number of partitions in the 3 nodes are somehow equally distributed.

Figure 3 illustrates the distribution of the assertions in the partitions on the first node. As shown in the figure, the number of assertions is quite different between partitions. These differences actually illustrate the structure of the test data.

We also ran the testing with four, five and six nodes to collect distribution data. The distribution is somehow similar to the case of 3 nodes. Table 5.2 listed the data collected for six nodes. The data distribution in our test is somehow nice, with the equally distribution of the partitions among nodes. However, this is the result of some synthetic benchmark data, which does not introduce many merging between partitions. Running our algorithm on more complex data, the partition allocation policy can be a critical factor deciding the system performance.

7 Query Answering

In the following section we investigate the problem of query answering over ontologies and in how far our proposal of island partitionings can help to solve problems locally. Solving the problem of *instance checking*, finding out whether $\mathcal{O} \models a : C$, is immediate from our proposal of island partitionings. Since we have

$$\mathcal{O} \models a : C \iff \langle \mathcal{T}, \mathcal{R}, \pi_S(\phi_P(a))\rangle \models a : C,$$

we run a tableaux algorithm on the ontology $\langle \mathcal{T}, \mathcal{R}, \pi_S(\phi_P(a)) \cup \{a : \neg C\}\rangle$ and check, whether it is consistent. If the ontology is inconsistent, then we proved that $\mathcal{O} \models a : C$, and non-entailment otherwise. Thus, instance checking can be performed locally on one node.

To solve the problem of *relation checking* for $\mathcal{ALCHI}$, i.e. find out whether $\mathcal{O} \models R(a, b)$, we can look at ontology $\langle \mathcal{T}, \mathcal{R}, \pi_S(\phi_P(a))\rangle$ and see, whether there exists a $R_2(a, b) \in \pi_S(\phi_P(a)$ (or a $R_3(b, a) \in \pi_S(\phi_P(a))$, such that R_2 is a subrole of R (or R_3 is a subrole of R^-). Thus, relation checking can be performed locally on one node again.

An extended decision problem is instance retrieval for a concept C, i.e. we want to find all named individuals a, such that $\mathcal{O} \models a : C$. The idea is that we determine first local solutions on each node, and then use a chosen master node to combine the results. More formally, given nodes $node_1, ..., node_n$ let $\{ap_{i,1}, ..., ap_{i,m}\}$ denote the ABox partitions associated to node i. We set $localresults_i = \{a \mid \exists j.ap_{i,j} \in node_i \wedge \langle \mathcal{T}, \mathcal{R}, \pi_S(\phi_P(a))\rangle \models a : C\}$. Then the result of instance retrieval is the union of all the local results, i.e. $\bigcup_{1\leq i\leq n} localresults_i$. Relation retrieval, i.e. finding all pairs of individuals connected by a role R can be handled in a similar fashion.

Last, we want to look into answering grounded conjunctive queries, without giving a formal definition. We rather want to provide the intuition and leave concrete results for future work. For $\mathcal{ALCHI}$ grounded conjunctive queries can be answered in the following way:

1. Retrieve the results for all concept query atoms in the query by instance retrieval
2. Retrieve the results for all role query atoms in the query by relation retrieval
3. Combine results from 1) and 2) to answer the grounded conjunctive query

We have provided modular solutions for instance retrieval and and relations retrieval above. For the combination of the results, we propose to use a centralized relational database system, such that we have

– one table for each concept query atom (one column corresponding to the individuals which match) and
– one table for each role query atom (two columns corresponding to the pairs of individuals which match) in the conjunctive query.

The idea is that the nodes fill the tables with their local information obtained from local instance and relation retrieval. Then, in the centralized system, the grounded conjunctive query is translated to a SQL query and executed. The result is then a table with individual tuples representing solutions for the conjunctive query. An example is given in Example 71.

Example 71 *Let* $q = Student(X) \wedge takesCourse(X, Y) \wedge GraduateCourse(X)$ *be a grounded conjunctive query. In the centralized database system we would create the three relations* $Student(X : TEXT)$*,* $takesCourse(X : TEXT, Y : TEXT)$ *and* $GraduateCourse(Y : TEXT)$*. The distributed partitioning systems will fill all three tables with their locally obtained results. After all local results are added, the following SQL-query is used to determine all results for the grounded conjunctive query:*

```
SELECT X,Y
FROM Student c1, takesCourse r1, GraduateStudent c2
WHERE
  c1.X=r1.X AND
  c2.Y=r1.Y AND
```

Please note that this approach can be further improved. For instance, if we use a stream based relational database system, then we don't have to wait until all local results are available in the centralized database, but we can evaluate the SQL-query incrementally, and thus, decrease intial query answering latency.

8 Conclusions

We have introduced means to reason over $\mathcal{ALCHI}$-ontologies, which have large amounts of assertional information. Our updatable partitioning approach allows state-of-the-art description logic reasoner to load only relevant subsets of the ABox to perform sound and complete reasoning. In particular, we have proposed a set of partitioning-preserving update transformations, which can be run on demand. Our techniques can be incorporated into the description logic reasoner RACER[HM01], to enable more scalable reasoning in the future.

In future work, we will investigate the applicability of our proposal to more expressive description logics, e.g. SHIQ. The extension for transitive roles is straightforward. The incorporation of min/max-cardinality constraints in a naive way can be done as well. However, it has to be investigated, whether the average partition size with these naive extensions is still small enough to be feasible in practice. Furthermore, we intend to perform more evaluation on real-world ontologies to provide detailed timing statistics. Especially the case of boot strapping the assertional part of an ontology needs further investigation.

References

[BCM$^+$07] Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider. *The Description Logic Handbook.* Cambridge University Press, New York, NY, USA, 2007.

[FKM$^+$06] Achille Fokoue, Aaron Kershenbaum, Li Ma, Chintan Patel, Edith Schonberg, and Kavitha Srinivas. Using Abstract Evaluation in ABox Reasoning. In *SSWS 2006*, pages 61–74, Athens, GA, USA, November 2006.

[GH06] Yuanbo Guo and Jeff Heflin. A Scalable Approach for Partitioning OWL Knowledge Bases. In *SSWS 2006*, Athens, GA, USA, November 2006.

[GPH05] Yuanbo Guo, Zhengxiang Pan, and Jeff Heflin. Lubm: A benchmark for owl knowledge base systems. *J. Web Sem.*, 3(2-3):158–182, 2005.

[HM01] V. Haarslev and R. Möller. Description of the racer system and its applications. In *Proceedings International Workshop on Description Logics (DL-2001), Stanford, USA, 1.-3. August*, pages 131–141, 2001.

[HwPS06] Christian Halashek-wiener, Bijan Parsia, and Evren Sirin. Description logics reasoning with syntactic updates. In *In Proc. of the 5th Int. Conf. on Ontologies, Databases, and Applications of Semantics (ODBASE 2006*. Sringer Verlag, 2006.

[MOS$^+$02] Boris Motik, Daniel Oberle, Steffen Staab, Rudi Studer, and Raphael Volz. Kaon server architecture. WonderWeb Deliverable D5, 2002. http://wonderweb.semanticweb.org.

[MSH07] Boris Motik, Rob Shearer, and Ian Horrocks. Optimized reasoning in description logics using hypertableaux. In Frank Pfenning, editor, *CADE*, volume 4603 of *Lecture Notes in Computer Science*, pages 67–83. Springer, 2007.

[Ngu09] Anh Ngoc Nguyen. Distributed storage system for description logic knowledge bases. In *Technical Report*, 2009. `http://www.sts.tu-harburg.de/~wandelt/research/NgocThesis.pdf`.

[SK04] H. Stuckenschmidt and M. Klein. Structure-based partitioning of large class hierarchies. In *International Semantic Web Conference*, 2004.

[WM08] Sebastian Wandelt and Ralf Moeller. Island reasoning for alchi ontologies. In Carola Eschenbach and Michael Grninger, editors, *FOIS*, volume 183 of *Frontiers in Artificial Intelligence and Applications*, pages 164–177. IOS Press, 2008.