Large-Scale Reasoning on Expressive Horn Ontologies

Carlo Allocca[3], Francesco Calimeri[1,2], Cristina Civili[3], Roberta Costabile[1], Bernardo Cuteri[1], Alessio Fiorentino[1], Davide Fuscà[1], Stefano Germano[2], Giovanni Laboccetta[2], Marco Manna[1], Simona Perri[1], Kristian Reale[2], Francesco Ricca[1], Pierfrancesco Veltri[2], and Jessica Zangari[1,2]

[1] Department of Mathematics and Computer Science, University of Calabria, Italy
{calimeri,r.costabile,cuteri,fiorentino,fusca,
manna,perri,ricca,zangari}@mat.unical.it
[2] DLVSystem L.T.D., Polo Tecnologico Unical, Rende, Italy
{calimeri,germano,laboccetta,reale,veltri}@dlvsystem.com
[3] Samsung R&D Institute, Staines, UK
{c.allocca,c.civili}@samsung.com

Abstract. Efficient large-scale reasoning is a fundamental prerequisite for the development of the Semantic Web. In this scenario, it is convenient to reduce standard reasoning tasks to query evaluation over (deductive) databases. From a theoretical viewpoint much has been done. Conversely, from a practical point of view, only a few reasoning services have been developed, which however typically can only deal with lightweight ontologies. To fill the gap, the paper presents OWL2DLV, a novel and modern Datalog system for evaluating SPARQL queries over very large OWL 2 knowledge bases. OWL2DLV builds on the well-known ASP system DLV by incorporating novel optimizations sensibly reducing memory consumption and a server-like behavior to support multiple-query scenarios. The high potential of OWL2DLV for large-scale reasoning is outlined by the results of an experiment on data-intensive benchmarks, and confirmed by the direct interest of a major international industrial player, which has stimulated and partially supported this work.

1 Introduction

Datalog is a powerful, yet simple and elegant rule-based language originally designed in the context of *deductive databases* for querying relational data. After almost 40 years, however, its scope of applicability and its extensions go definitely beyond the initial target, so much that now they range from optimization and constraint satisfaction problems [19] to even ontological design and reasoning in the Semantic Web [20,30]. Indeed, in the development of the Semantic Web, efficient large-scale reasoning is a fundamental prerequisite. In this scenario, it is convenient to reduce standard reasoning tasks to query evaluation over (deductive) databases. From a theoretical viewpoint much has been done: in many ontological settings, the problem of evaluating a conjunctive query over a *knowledge base* (KB) consisting of an *extensional dataset* (ABox) paired with

an *ontology* (TBox) can be reduced to the evaluation of a Datalog query (i.e., a Datalog program, possibly nonrecursive and including strong constrains, paired with a union of conjunctive queries, both constructed only from the original query and the TBox) over the same ABox [25,28,35,42,45]. Conversely, from a practical viewpoint the situation is not so rosy. Many classical Datalog reasoners, such as CLINGO [29] and DLV [39], are based on one-shot executions performing heavy operations (e.g., loading and indexing) multiple times and hence are rather unsuited. Also, only a few reasoning services with a server-like behavior, such as MASTRO [24], ONTOP [23], and RDFOX [41], have been developed, which however can only deal with lightweight TBoxes. To fill the gap, the paper presents OWL2DLV, a modern Datalog system, based on the aforementioned rewriting approach, for evaluating SPARQL conjunctive queries [44] over very large OWL 2 knowledge bases [26].

Reasoning over OWL 2 is generally a very expensive task: fact entailment (i.e., checking whether an individual is an instance of a concept) is already 2NEXPTIME-hard, while decidability of conjunctive query answering is even an open problem. To balance expressiveness and scalability, the W3C identified three tractable profiles —OWL 2 EL, OWL 2 QL, and OWL 2 RL— exhibiting good computational properties: the evaluation of conjunctive queries over KBs falling in these fragments is in PTIME in data complexity (query and TBox are considered fixed) and in PSPACE in combined complexity (nothing is fixed) [43]. To deal with a wide variety of ontologies, OWL2DLV implements the Horn-$\mathcal{SHIQ}$ fragment of OWL 2, which enjoys good computational properties: conjunctive queries are evaluated in PTIME (resp., EXPTIME) in data (resp., combined) complexity. Moreover, it is also quite expressive: it generalizes both OWL 2 QL and OWL 2 RL, while capturing all OWL 2 EL constructs except role chain [38].

From the technical side, OWL2DLV builds on the well-known ASP system DLV [39], and in particular its most recent incarnation DLV2 [5], by incorporating a server-like modality, which is able to keep the main process alive, receive and process multiple user's requests on demand, and restore its status thanks to an embedded persistency layer. Following the approach proposed by Eiter et al. [28], a Horn-$\mathcal{SHIQ}$ TBox paired with a SPARQL query are rewritten, independently from the ABox, into an equivalent Datalog query.

The high potential of OWL2DLV for large-scale ontological reasoning is outlined by the results of an experiment on data-intensive benchmarks, and confirmed by the direct interest of a big international industrial player, which has partially supported this work and also stimulated the evolution of the system with a major *challenge*: "deal with LUBM-8000 —the well-known LUBM [31] standard benchmark for ontological reasoning collecting about 1 billion factual assertions upon 8,000 universities— over machines equipped with 256GB RAM and with an average query evaluation time of at most 10 minutes". Eventually, not only the system was able to widely win the general challenge as reported in Figure 1; but, amazingly, the average time taken by OWL2DLV on the ten (out of fourteen) *bound* queries —i.e., queries containing at least one constant— of LUBM-8000 was eventually less than one second (see Section 7 for details).

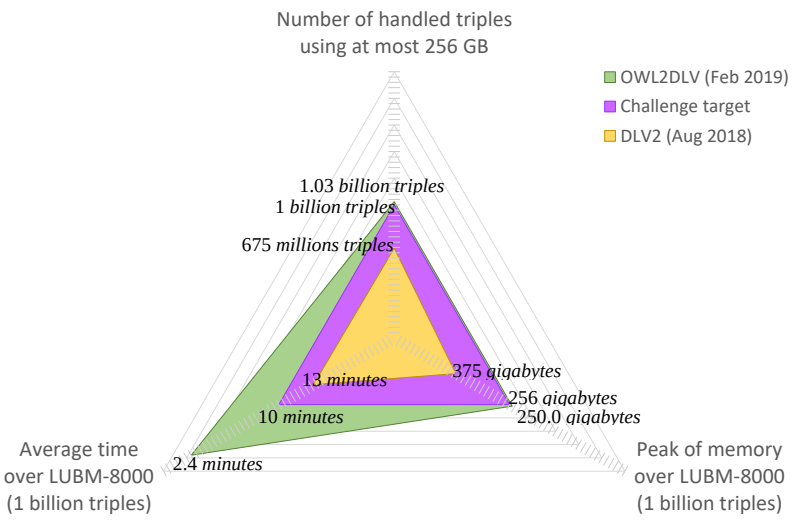

Fig. 1: OWL2DLV – performance and enhancements.

2 Background

OWL 2. As said, we focus on the Horn-$\mathcal{SHIQ}$ [33] fragment of OWL 2. In Description Logic (DL) terminology and notation, a *knowledge base* (KB) $\mathcal{K}$ is a pair $(\mathcal{A}, \mathcal{T})$, where $\mathcal{A}$ is a set of *(factual) assertions* representing extensional knowledge about individuals and $\mathcal{T}$ is a set of *axioms* representing intensional knowledge about the domain of interest. Let $\mathsf{N_I}$ (*individuals*), $\mathsf{N_C} \supset \{\top, \bot\}$ (*atomic concepts*) and $\mathsf{N_R}$ (*role names*) be pairwise disjoint discrete sets. A *role* r is either a role name s or its *inverse* s^-. A *concept* is either an atomic concept or an expression of the form $C \sqcap D$, $C \sqcup D$, $\neg C$, $\forall r.C$, $\exists r.C$, $\geqslant nr.C$ or $\leqslant nr.C$, where C and D are concepts, r is a role, and $n \geq 1$. *General concept inclusions* (GCIs), *role inclusions* (RIs), and *transitive axioms* (TAs) are respectively of the form $C_1 \sqsubseteq C_2$, $r_1 \sqsubseteq r_2$, and $Tr(r)$, where: $\sqcup$ is disallowed in C_2, $\geqslant nr$ and $\leqslant nr$ are disallowed in C_1, and they are disallowed also in C_2 in case r is transitive. A Horn-$\mathcal{SHIQ}$ TBox is a finite set of GCIs, RIs and TAs satisfying some non-restrictive global conditions [34,36]. An instance I is a set of assertions of the form $C(a)$ and $r(a, b)$, where $C \in \mathsf{N_C}$, $r \in \mathsf{N_R}$, and $a, b \in \mathsf{N_I}$. An ABox is any finite instance. Common formats for OWL 2 KBs are both Turtle [17] and RDF/XML [26]. For example, assertions $Woman(Ann)$, $Person(Tom)$ and $likes(Tom, Ann)$ are encoded in Turtle as:

```
:Ann rdf:type :Woman .     :Tom rdf:type :Person ;  :likes :Ann .
```

while the GCI $Woman \sqsubseteq Person$ is encoded in RDF/XML as:

```
<owl:Class rdf:about="#Person"/>
<owl:Class rdf:about="#Woman">
  <rdfs:subClassOf rdf:resource="#Person"/>
</owl:Class>
```

SPARQL. It is the standard language in the Semantic Web for querying OWL 2 knowledge bases [44]. As in databases, the most important class of SPARQL queries are the *conjunctive* ones, which syntactically are quite similar to SQL queries. For example, to select the individuals that are instances of *Person* we write `SELECT ?X WHERE { ?X rdf:type :Person }`. According to the example given in the previous section, however, the answer to this query is different when executed over the ABox only (giving just *Tom* as answer) or by also taking into account the TBox (giving as answer also *Ann*). More generally, when querying OWL 2 knowledge bases the TBox plays the role of a fist-order theory and it has to be taken into account properly, as described next.

OBQA. A *model* of a KB $\mathcal{K} = (\mathcal{A}, \mathcal{T})$ is typically any instance $I \supseteq \mathcal{A}$ satisfying all the axioms of $\mathcal{T}$, written $I \models \mathcal{T}$, where GCIs, RIs and TAs can be regarded as first-order expressions [15]. For example, inclusion $C \sqcap D \sqsubseteq E$ over atomic concepts is equivalent to $\forall x\,(E(x) \leftarrow C(x) \wedge D(x))$. (For a comprehensive picture, we refer the reader to [38].) The set of all models of $\mathcal{K}$ is denoted by $\mathsf{mods}(\mathcal{K})$. To comply with the so-called *open world assumption* (OWA), note that I might contain individuals that do not occur in $\mathcal{K}$. The *answers* to a query $q(\bar{x})$ over an instance I is the set $q(I) = \{\bar{a} \in \mathsf{N_I}^{|\bar{x}|} \mid I \models q(\bar{a})\}$ of $|\bar{x}|$-tuples of individuals obtained by evaluating q over I. Accordingly, the *certain answers* to q under OWA is the set $\mathsf{cert}(\mathcal{K}, q) = \bigcap_{I \in \mathsf{mods}(D,\Sigma)} q(I)$. Finally, ontology-based query answering (OBQA) is the problem of computing $\mathsf{cert}(\mathcal{K}, q)$.

DLV. It is one of the most used logic programming systems based on answer set semantics[39], a well-known formalism extended by many expressive constructs [3] useful in several application domains [4,8,9,10,11]. Recently, it has been redesigned and reengineered to version 2.0, called DLV2 [5], to enjoy modern evaluation techniques together with development platforms fully complying with the ASP-Core-2 language. Nowadays, it integrates two sub-systems: I-DLV [21], handling the deductive databases and program grounding, and WASP [6] for the model search phase. DLV2 has been the basis for the development of OWL2DLV.

3 Ontological Reasoning via Datalog

As said, to perform OBQA, OWL2DLV follows the approach of Eiter et al. [28]. From an OWL 2 Horn-$\mathcal{SHIQ}$ TBox $\mathcal{T}$ and a SPARQL conjunctive query $q(\bar{x})$, OWL2DLV runs Algorithm 1 to build a Datalog program $P_{\mathcal{T}}$ and a union of conjunctive queries $Q_{q,\mathcal{T}}(\bar{x})$ such that, for each ABox $\mathcal{A}$, the evaluation of $Q_{q,\mathcal{T}}(\bar{x})$ over $\mathcal{A} \cup P_{\mathcal{T}}$ produces the same answers as the evaluation of $q(\bar{x})$ over $\mathcal{A} \cup \mathcal{T}$.

Algorithm 1: TBox and Query Rewriting

Input: An OWL 2 Horn-$\mathcal{SHIQ}$ TBox $\mathcal{T}$ together with a query $q(\bar{x})$
Output: The Datalog program $P_{\mathcal{T}}$ together with the query $Q_{q,\mathcal{T}}(\bar{x})$

1. $\mathcal{T}' \leftarrow \mathsf{Normalize}(\mathcal{T})$;
2. $\mathcal{T}^* \leftarrow \mathsf{EmbedTransitivity}(\mathcal{T}')$;
3. $\Xi(\mathcal{T}^*) \leftarrow \mathsf{Saturate}(\mathcal{T}^*)$;
4. $P_{\mathcal{T}} \leftarrow \mathsf{RewriteTBox}(\Xi(\mathcal{T}^*))$;
5. $Q_{q,\mathcal{T}}(\bar{x}) \leftarrow \mathsf{RewriteQuery}(q(\bar{x}), \Xi(\mathcal{T}^*))$;

As an example, consider a TBox $\mathcal{T}$ consisting of the GCIs $\mathit{CommutingArea} \sqsubseteq \exists \mathit{linked}.\mathit{Capital}$, $\exists \mathit{linked}.\mathit{Capital} \sqsubseteq \mathit{DesirableArea}$, and $\mathit{Capital} \sqsubseteq \mathit{DesirableArea}$, together with the TA $\mathit{Tr}(\mathit{linkedViaTrain})$ and the RI $\mathit{linkedViaTrain} \sqsubseteq \mathit{linked}$. According to Algorithm 1 $P_{\mathcal{T}}$ is as follows:

```
linked(X,Y) :-  linkedViaTrain(X,Y).
desirableArea(Y) :- capital(X), linked(Y,X).
desirableArea(X) :- capital(X).
capital*(Y) :- commutingArea(X), linkedViaTrain(X,Y).
capital*(Y) :- capital*(X), linkedViaTrain(X,Y).
capital(X) :- capital*(X).
desirableArea(X) :- commutingArea(X).
```

Moreover, starting from the SPARQL query $q(\bar{x})$ reported below

```
SELECT ?X WHERE { ?X :linkedViaTrain ?Y. ?Y rdf:type :DesirableArea }
```

we obtain the following UCQ $Q_{q,\mathcal{T}}(\bar{x})$, also encoded as a set of Datalog rules:

```
q(X) :- linkedViaTrain(X,Y), desirableArea(Y).
q(X) :- linkedViaTrain(X,Y), capital(Y).
q(X) :- linkedViaTrain(X,Y), commutingArea(Y).
q(X) :- linkedViaTrain(X,Y), capital*(Y).
q(X) :- commutingArea(X).
q(X) :- capital*(X).
```

4 Query Optimization

The pair $(Q_{q,\mathcal{T}}(\bar{x}), P_{\mathcal{T}})$ returned by Algorithm 1 is further optimized by a *pruning strategy* followed by the so-called *Magic Sets rewriting* —the latter is already in use in DLV2 but it has been further improved due to the specific nature of $P_{\mathcal{T}}$. The result of this phase consists of the pair $(\mathsf{opt}(Q_{q,\mathcal{T}}(\bar{x})), \mathsf{opt}(P_{\mathcal{T}}))$.

Pruning Strategy. Pairs of GCIs of the form $C_1 \sqsubseteq C_2$ and $C_2 \sqsubseteq C_1$ give rise to Datalog queries containing rules with multiple predicates having the same extensions: $C_1(X) :- C_2(X)$ and $C_2(X) :- C_1(X)$. The same happens with RIs. During the evaluation of the query, however, this can be considerably expensive. Hence, we adopt the following pruning strategy. Let $\mathcal{E} = \{E_1, ..., E_k\}$ be a set of equivalent concepts or roles. First, we remove from $P_{\mathcal{T}}$ all the rules of the form $E_i(X) :- E_j(X)$ with $1 < i \leq k$ and $1 \leq j \leq k$. Second, let $P^1_{\mathcal{T}}$ be the subset of $P_{\mathcal{T}}$ containing only rules of the form $E_1(X) :- E_j(X)$ with $2 \leq j \leq k$, for each $i \in \{2, ..., k\}$, we replace each occurrence of E_i by E_1 both in $Q_{q,\mathcal{T}}(\bar{x})$ and in each rule of $P_{\mathcal{T}}$ that does not belong to $P^1_{\mathcal{T}}$. An analogous technique is applied over RIs of the form $r \sqsubseteq s^-$ and $s^- \sqsubseteq r$ by taking into account, in this case, that the first argument of r (resp., s) maps the second one of s (resp., r).

Magic Sets Rewriting. Datalog systems usually implement a bottom-up algorithm that iteratively derives new facts by matching bodies with already known facts. Queries are answered on the fixpoint of the algorithm producing the canonical model of the program. In contrast, a typical top-down algorithm for query answering looks for a rule from which some answers to the input query might be derived; if this kind of rule is found, its body atoms are considered as subqueries

and the procedure is iterated. This way, only parts of the program that are relevant for answering the query are evaluated. Magic Sets [7] aim at combining the benefits of the two algorithms; in fact the program is rewritten to simulate a top-down query evaluation via a bottom-up algorithm. Basically, Magic Sets introduce rules defining additional atoms, named *magic atoms*, to identify *relevant atoms* for answering the query, namely atoms reachable by a top-down query evaluation. The bottom-up evaluation is then limited by adding magic atoms in the bodies of the original rules. Consider the Datalog query

```
ancestor(X,Y) :- parent(X,Y).
ancestor(X,Y) :- parent(X,Z), ancestor(Z,Y).
ancestor(mario,X)?
```

The Magic Sets rewriting starts with the query seed `m#ancestor#bf(mario)`, modifies the rules defining the intentional predicate ancestor, and introduces magic rules for every occurrence of intentional predicates in the modified rules:

```
m#ancestor#bf(mario).
ancestor(X,Y) :- m#ancestor#bf(X), parent(X,Y).
ancestor(X,Y) :- m#ancestor#bf(X),parent(X,Z),ancestor(Z,Y).
m#ancestor#bf(Z) :- m#ancestor#bf(X), parent(X,Z).
ancestor(mario,X)?
```

The new program is specialized for answering the original query as its bottom-up evaluation only materializes descendants of *mario*, rather than the full *ancestor* relation. For large programs, however, many "irrelevant" rules may be introduced, which in turn unavoidably overload the reasoning. To optimize the rewriting, OWL2DLV performs two novel steps: (1) Eliminate rules that have a magic atom with predicate `m#p#`α if $\alpha \neq$ `f`$\cdots$`f` and `m#p#f`$\cdots$`f` also occurs in the rewritten program; and (2) Remove every rule r_1 whenever it is subsumed by some other rule r_2 ($r_1 \sqsubseteq r_2$), namely there is a variable substitution that maps the head (resp., body) of r_2 to the head (resp., body) of r_1. To avoid the quadratic number of checks, OWL2DLV associates each rule with a suitable hash value of size 64 bits. Then, $r_1 \sqsubseteq r_2$ is checked only if the bit-a-bit equation $hash(r_1)$ & $hash(r_2) == hash(r_2)$ is satisfied.

5 OWL2DLV: Design and Implementation

The OWL2DLV architecture is depicted in Figure 2. The system features four main modules: Loading, Rewriting, Query Answering, and Command Interpreter. Clients interact with the system through the latter one, which takes user commands and requests to the internal modules to execute the corresponding behavior, and provide the output to the client. This module allows to "keep alive" the system, and execute multiple commands (e.g., loading, warmup, data updates, and query evaluations) without having to instantiate a new process for each client request. The Command Interpreter can be controlled either via command line (e.g., bash shell in Linux) or from external applications through a Java API. The remaining three modules are discussed below.

Loading. It handles the input of the system. In particular, it processes an OWL 2 ABox encoded in Turtle format via the ABox Loader. To guarantee a

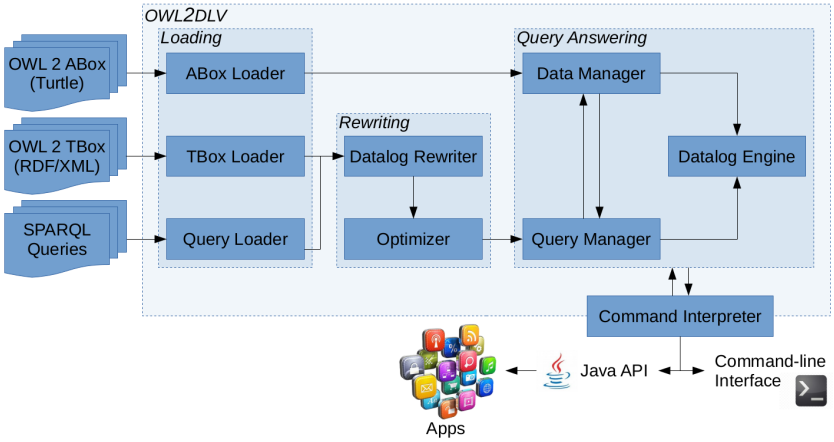

Fig. 2: System Architecture

high performance both in the scanner and in the parser phase, OWL2DLV employs *Flex & Bison [1,2]* in the implementation of the official Turtle grammar and the parsing procedure. The RDFS datatypes `xsd:double`, `xsd:integer`, and `xsd:string` are efficiently and effectively handled; nevertheless, the datatypes check is optional. The result of the parsing phase are Datalog-like facts; with all the prefixed names expanded, *predicate names* encoded using `IRI`s and *terms* encoded using strings (and eventually integers). They are then stored in the OWL2DLV data structures handled by the Data Manager. Finally, it worth noting that a scalability test over the LUBM dataset [31] shows that this module evolves linearly both in time and memory. Concerning the TBox, OWL2DLV supports OWL 2 Horn-$\mathcal{SHIQ}$ ontologies encoded in RDF/XML. The input is parsed by the TBox Loader using the well-known OWL API [32] and loaded in DL-like data structures that are suitable for the purposes of Algorithm 1. During the parsing, every range restriction on one of the aforementioned datatype properties is also kept in a suitable data structure, later exploited by other modules. Finally, the system supports a set of SPARQL conjunctive queries via the Query Loader. Also in this case, the input is parsed via the OWL API.

Rewriting. It is responsible to implement Algorithm 1 via the Datalog Rewriter submodule and optimize —as described in Section 4 by applying the *pruning strategy* and the *Magic Sets rewriting*— its output $(Q_{q,\mathcal{T}}(\bar{x}), P_{\mathcal{T}})$ via the Optimizer. The Datalog Rewriter is also in charge of producing the *datatypes directives*: for each data property enforcing a datatype d in the range of a role r, the directive `#datatypes('r/2',{2:d})` is added to $P_{\mathcal{T}}$. This syntax is inherited from DLV2, where `r/2` says that r is of arity 2, and `2:d` that the second argument of r must be of type d. Moreover, for each range assertion for a role r_2 over a datatype d and for each role inclusion $r_1 \sqsubseteq r_2$, a range assertion for r_1 over d is added to $P_{\mathcal{T}}$, until a fixpoint is reached.

Query Answering. This module, which is in charge of answering to Datalog queries over the input ABox, consists of three submodules. The Data Manager

handles the extensional part of each input predicate and organizes it efficiently by means of indexed data structures; as data may undergo through updates, the module is also responsible for managing data additions and deletions. The Query Manager handles query rewritings, along with meta-information about the scheduling, guiding the evaluation of each input query. These two submodules may need to exchange info: the Data Manager may ask for information about the structure of the Datalog queries to single out an optimal indexing policy, while the Query Manager may need statistics about data distribution to define an optimal scheduling. Query answering is carried out by the Datalog Engine, which represents an extension of I-DLV [21] (the grounder of DLV2). The overall evaluation procedure is based on a bottom-up process based on a semi-naïve approach empowered with optimizations working in synergy [21,22] and extended via techniques specifically devised to manage efficiently large sizes of data as we briefly summarize below. The evaluation process has been endowed with a fruitful *memory-releasing policy* that, on the basis of structural information over the program at hand, anticipates the release of memory occupied by internal data structures as soon as these are no longer needed. Specifically, data structures intended to represent the intensional part of a predicate p, are removed once all rules depending on p have been fully grounded. Moreover, we devised an *optimized data retrieval strategy* to reduce both the memory needed to store data and the time required to retrieve them. In particular, performance improvements have been achieved by re-implementing internal data structures and by optimizing crucial points of the retrieval task that, when frequently executed on large sizes, may negatively affect the performance. The system features also a persistence mechanism allowing to serialize input data handled by the Data Manager on the disk in order to enable a faster reloading of input data.

6 OWL2DLV: Functionalities and Application Scenarios

When dealing with query answering in large-scale contexts, it is fairly common that the system at hand is required to repeatedly query on demand a certain KB where the ABox slowly changes over time by preserving, however, its structural properties. This features a number of *scenarios* that might significantly differ, depending on what is known before the reasoning starts. To maximize the performance, each scenario requires an appropriate query answering strategy, or *setting*, acting as a tuning mechanism that determines an effective indexing schema, along with a compatible body-ordering (when possible) for all rules. This is performed by the Query Answering module during the so-called *warmup* phase, executed just after the loading phase has been completed. The resulting overhead, although not negligible, is paid only once. We now summarize some of the most common *scenarios* while assuming that an initial ABox $\mathcal{A}$ is known.
Informed. Both the TBox $\mathcal{T}$ and some *template queries* (i.e., prototypical conjunctive queries where some arguments bounded by constants are marked in order to indicate that such constants might change at query time) are known. In this case, for each template query $q(\bar{x})$, the system performs a preliminary run of $\mathsf{opt}(Q_{q,\mathcal{T}}(\bar{x}))$ over $\mathcal{A} \cup \mathsf{opt}(P_{\mathcal{T}})$. This step pre-computes indices and body

orderings that will be of use when the system will be actually queried; roughly, it saves the choices that DLV2 would make in case of a "one-shot" execution.
Responsive. The TBox $\mathcal{T}$ is known, while no information is available about the incoming queries. Here, a more general strategy is adopted: the system performs a single preliminary run over $\mathcal{A} \cup P_{\mathcal{T}}$ (without any query) and, similarly to the previous case, it stores information about body ordering and indexing for all rules of $P_{\mathcal{T}}$ that DLV2 would choose in a one-shot execution. Then, by predicting how $P_{\mathcal{T}}$ is generally modified by the pruning and the Magic Sets rewriting, OWL2DLV enriches the set of created indices accordingly.
Dynamic. Nothing is available. Clearly, no preliminary run for pre-computing body orderings can be performed. Depending on the memory availability, this drives the system to opt either for an *aggressive* indexing policy where all possible indices are computed over input data or a *parsimonious* (yet "blind") indexing policy where only the first attribute of each predicate is indexed, while delegating to the query answering phase the creation of possibly needed further indices. This setting is also used as the default whenever the user tries to specify a setting not compatible with the actual scenario.

7 OWL2DLV: Performance

We report the results of an experimental evaluation of OWL2DLV over LUBM [31] and DBpedia [14]. Note that, by focusing on the few ready-to-use OWL 2 reasoning services with a server-like behavior, neither MASTRO [24] nor ONTOP [23] nor RDFOX [41] fully support query answering in both domains; in particular all of them do not process some of the axioms in the LUBM TBox. Further experiments with computationally intensive benchmarks, such as LUBM$^{\exists}$ [40] and UOBM [37], will be part of an extended version of this paper. Moreover, a comparison against MASTRO, ONTOP and RDFOX on lightweight ontologies as well as a comparison against modern Datalog-based systems like VADALOG [18] and GRAAL [16] for query existential rules [12,13] is also in our agenda.
Benchmarks. LUBM is the prime choice of our industrial partner for the challenge. It describes a very-large real-world application domain encoded in OWL 2 Horn-$\mathcal{SHIQ}$ with customizable and repeatable synthetic data. The benchmark incorporates 14 SPARQL queries, 10 of which are *bound* (i.e., containing at least a constant). When rewritten together with the TBox, each LUBM query gives rise to a Datalog query consisting of about 130 rules. Data generation is carried out by the LUBM data generator tool (UBA) whose main parameter is the number of universities to consider: 8,000 in our case, for a total number of about 1 billion triples. This dataset is next referred to as LUBM-8000. Concerning DBpedia, it is a well-known KB created with the aim of sharing on the Web the multilingual knowledge collected by Wikimedia projects in a machine-readable format. For this benchmarks, we inherited a set of queries from an application conceived to query DBpedia in natural language applying the approach [27]. When rewritten together with the TBox, each DBpedia query gives rise to a Datalog query of almost 5400 rules. The latest release of the official DBpedia dataset consists of 13 billion pieces of multilingual information (RDF triples). Here, we focus on the

	LUBM-8000				DBpedia			
	CQ name	*informed*	*responsive*	*dynamic*	CQ name	*informed*	*responsive*	*dynamic*
	q01	0.00	0.00	71.77	**q01**	0.18	0.32	2.51
	q02	194.48	179.65	295.74	**q02**	0.17	0.29	2.15
	q03	0.00	0.00	160.95	**q03**	0.22	0.32	2.29
	q04	0.01	0.01	379.22	**q04**	0.21	0.30	0.33
	q05	0.03	0.03	19.60	**q05**	0.20	0.29	7.52
	q06	844.65	854.35	978.46	**q06**	0.19	0.29	0.38
	q07	0.01	0.01	5.07	**q07**	0.19	0.38	0.79
	q08	0.40	0.32	0.50	**q08**	0.18	0.32	0.31
	q09	972.63	1,008.43	1,053.82				
	q10	0.00	0.01	4.66				
	q11	0.00	0.00	0.00				
	q12	0.03	0.03	0.03				
	q13	6.32	6.69	8.13				
	q14	14.64	14.50	14.12				
Max Memory		250.0	251.0	249.3		63.6	96.4	106.2
Loading		5,226.8	5,196.4	5,186.8		3,845.3	3,719.0	3,811.2
Warmup		4,144.4	3,102.3	1,303.8		226.5	1,136.3	595.5
Query Answering (all)		145.2	147.4	238.9		0.2	0.3	2.0
Query Answering (bound)		0.7	0.7	106.2		0.2	0.3	2.0

Table 1: Experimental evaluation of OWL2DLV in different scenarios. Bound queries are reported in bold. The rows "Query Answering (all)" and "Query Answering (bound)" show, respectively, the average evaluation time computed over all queries and bound queries only. Times are expressed in seconds and memory peaks in GB.

information extracted from the English edition of Wikipedia that is composed by about half a billion triples (https://wiki.dbpedia.org/public-sparql-endpoint).

Results and Discussion. The machine used for testing is a Dell Linux server with an Intel Xeon Gold 6140 CPU composed of 8 physical CPUs clocked at 2.30 GHz, with 297GB of RAM. According to the challenge, a memory limit of 256GB has been set during all tests. Table 1 shows the results of our analysis; bound queries are reported in bold. The upper part of the table reports times needed by OWL2DLV to answer queries under the three scenarios discussed in Section 6; the second part shows extra statistics about peaks of memory, loading and warmup times, and average answering times computed over all queries and over bound queries only. In the *informed* scenario –where we assume that the TBox and the template queries are known in advance– we obtain the best performance. Despite the large ABox, on LUBM almost all bound queries are answered in less than 0.1 seconds with an average time of about 0.7 seconds, while on DBpedia the average evaluation time over all queries is about 0.2 seconds. These results confirm the effectiveness of all enhancements herein discussed and of the Magic Sets technique. In the *responsive* scenario –where only the TBox is known in advance– the system performance is comparable with the one obtained in the *informed* scenario, although in general the evaluation time is a bit higher. These results confirm that the warmup policy of this setting has a positive impact on the system performance although queries are not known. Finally, in the *dynamic* scenario –where nothing is known– the parsimonious indexing policy is adopted since 256GB are not enough to use the aggressive one. Uniformly, the same policy is also used for DBpedia although not expressly needed. This produces a general gain in the warmup phase later unavoidably paid during the query evaluation due to some missing index that has to be computed on-the-fly.

Acknowledgments. This work has been partially supported by Samsung under project "Enhancing the DLV system for large-scale ontology reasoning", by MISE under project "S2BDW" (F/050389/01-03/X32) – "Horizon2020" PON I&C2014-20 and by Regione Calabria under project "DLV LargeScale" (CUP J28C17000220006) – POR Calabria 2014-20.

References

1. The fast lexical analyzer. URL https://github.com/westes/flex
2. Gnu bison. URL https://www.gnu.org/software/bison
3. Adrian, W.T., Alviano, M., Calimeri, F., Cuteri, B., Dodaro, C., Faber, W., Fuscà, D., Leone, N., Manna, M., Perri, S., Ricca, F., Veltri, P., Zangari, J.: The ASP system DLV: advancements and applications. KI **32**(2-3), 177–179 (2018)
4. Adrian, W.T., Manna, M., Leone, N., Amendola, G., Adrian, M.: Entity set expansion from the web via ASP. In: ICLP TCs, pp. 1:1–1:5 (2017)
5. Alviano, M., Calimeri, F., Dodaro, C., Fuscà, D., Leone, N., Perri, S., Ricca, F., Veltri, P., Zangari, J.: The ASP system DLV2. In: LPNMR (2017)
6. Alviano, M., Dodaro, C., Leone, N., Ricca, F.: Advances in WASP. In: LPNMR, *LNCS*, vol. 9345, pp. 40–54 (2015)
7. Alviano, M., Faber, W., Greco, G., Leone, N.: Magic sets for disjunctive datalog programs. Artif. Intell. **187**, 156–192 (2012)
8. Amendola, G.: Preliminary results on modeling interdependent scheduling games via answer set programming. In: RiCeRcA@AI*IA, vol. 2272 (2018)
9. Amendola, G.: Solving the stable roommates problem using incoherent answer set programs. In: RiCeRcA@AI*IA, vol. 2272 (2018)
10. Amendola, G., Dodaro, C., Leone, N., Ricca, F.: On the application of answer set programming to the conference paper assignment problem. In: AI*IA (2016)
11. Amendola, G., Greco, G., Leone, N., Veltri, P.: Modeling and reasoning about NTU games via answer set programming. In: IJCAI, pp. 38–45 (2016)
12. Amendola, G., Leone, N., Manna, M.: Finite model reasoning over existential rules. TPLP **17**(5-6), 726–743 (2017)
13. Amendola, G., Leone, N., Manna, M., Veltri, P.: Enhancing existential rules by closed-world variables. In: IJCAI, pp. 1676–1682 (2018)
14. Auer, S., Bizer, C., Kobilarov, G., Lehmann, J., Cyganiak, R., Ives, Z.G.: Dbpedia: A nucleus for a web of open data. In: ISWC, LNCS, pp. 722–735 (2007)
15. Baader, F., Calvanese, D., McGuinness, D.L., Nardi, D., Patel-Schneider, P.F. (eds.): The Description Logic Handbook. C.U.P. (2003)
16. Baget, J., Leclère, M., Mugnier, M., Rocher, S., Sipieter, C.: Graal: A toolkit for query answering with existential rules. In: RuleML, pp. 328–344 (2015)
17. Beckett, D., Berners-Lee, T., Prud'hommeaux, E., Carothers, G., Machina, L.: Rdf 1.1 turtle – terse rdf triple language. W3C Recommendation (2014)
18. Bellomarini, L., Sallinger, E., Gottlob, G.: The vadalog system: Datalog-based reasoning for knowledge graphs. PVLDB **11**(9), 975–987 (2018)
19. Bodirsky, M., Dalmau, V.: Datalog and constraint satisfaction with infinite templates. J. Comput. Syst. Sci. **79**(1), 79–100 (2013). DOI 10.1016/j.jcss.2012.05.012
20. Calì, A., Gottlob, G., Lukasiewicz, T.: A general datalog-based framework for tractable query answering over ontologies. J. Web Semant. **14**, 57–83 (2012)
21. Calimeri, F., Fuscà, D., Perri, S., Zangari, J.: I-DLV: the new intelligent grounder of DLV. Intelligenza Artificiale **11**(1), 5–20 (2017). DOI 10.3233/IA-170104

22. Calimeri, F., Perri, S., Zangari, J.: Optimizing answer set computation via heuristic-based decomposition. TPLP p. 1–26 (2019)
23. Calvanese, D., Cogrel, B., Komla-Ebri, S., Kontchakov, R., Lanti, D., Rezk, M., Rodriguez-Muro, M., Xiao, G.: Ontop: Answering SPARQL queries over relational databases. Semantic Web **8**(3), 471–487 (2017)
24. Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., Poggi, A., Rodriguez-Muro, M., Rosati, R., Ruzzi, M., Savo, D.F.: The MASTRO system for ontology-based data access. Semantic Web **2**(1), 43–53 (2011). DOI 10.3233/SW-2011-0029
25. Carral, D., Dragoste, I., Krötzsch, M.: The combined approach to query answering in horn-alchoiq. In: KR, pp. 339–348. AAAI Press (2018)
26. Carroll, J., Herman, I., Patel-Schneider, P.F.: Owl 2 web ontology language rdf-based semantics (second edition) (2012)
27. Cuteri, B., Reale, K., Ricca, F.: A logic-based question answering system for cultural heritage. In: JELIA, LNCS. Springer (to appear) (2019)
28. Eiter, T., Ortiz, M., Simkus, M., Tran, T., Xiao, G.: Query rewriting for horn-shiq plus rules. In: AAAI (2012)
29. Gebser, M., Kaminski, R., Kaufmann, B., Ostrowski, M., Schaub, T., Wanko, P.: Theory solving made easy with clingo 5. In: ICLP TCs, pp. 2:1–2:15 (2016)
30. Grau, B.C., Kharlamov, E., Kostylev, E.V., Zheleznyakov, D.: Controlled query evaluation for datalog and OWL 2 profile ontologies. In: IJCAI (2015)
31. Guo, Y., Pan, Z., Heflin, J.: LUBM: A benchmark for OWL knowledge base systems. J. Web Semant. **3**(2-3), 158–182 (2005)
32. Horridge, M., Bechhofer, S.: The OWL API: A java API for OWL ontologies. Semantic Web **2**(1), 11–21 (2011). DOI 10.3233/SW-2011-0025
33. Hustadt, U., Motik, B., Sattler, U.: Data complexity of reasoning in very expressive description logics. In: IJCAI, pp. 466–471 (2005)
34. Kazakov, Y.: Consequence-driven reasoning for horn SHIQ ontologies. In: IJCAI, pp. 2040–2045 (2009)
35. Kontchakov, R., Lutz, C., Toman, D., Wolter, F., Zakharyaschev, M.: The combined approach to ontology-based data access. In: IJCAI (2011)
36. Krötzsch, M., Rudolph, S., Hitzler, P.: Complexities of horn description logics. ACM Trans. Comput. Log. **14**(1), 2:1–2:36 (2013)
37. Ledvinka, M., Kremen, P.: Object-uobm: An ontological benchmark for object-oriented access. In: KESW, vol. 518, pp. 132–146. Springer (2015)
38. Leone, N.: The AI system DLV: ontologies, reasoning, and more. In: IC3K, pp. 5–16 (2018)
39. Leone, N., Pfeifer, G., Faber, W., Eiter, T., Gottlob, G., Perri, S., Scarcello, F.: The DLV system for knowledge representation and reasoning. TOCL (2006)
40. Lutz, C., Seylan, I., Toman, D., Wolter, F.: The combined approach to OBDA: taming role hierarchies using filters. In: ISWC, pp. 314–330 (2013)
41. Nenov, Y., Piro, R., Motik, B., Horrocks, I., Wu, Z., Banerjee, J.: Rdfox: A highly-scalable RDF store. In: ISWC, vol. 9367, pp. 3–20 (2015)
42. Stefanoni, G., Motik, B., Horrocks, I.: Small datalog query rewritings for EL. In: DL, *CEUR Workshop Proceedings*, vol. 846 (2012)
43. Stefanoni, G., Motik, B., Krötzsch, M., Rudolph, S.: The complexity of answering conjunctive and navigational queries over OWL 2 EL knowledge bases. J. Artif. Intell. Res. **51**, 645–705 (2014)
44. W3C: SPARQL 1.1 entailment regimes. https://www.w3.org/TR/sparql11-entailment/ (2013)
45. Xiao, G., Calvanese, D., Kontchakov, R., Lembo, D., Poggi, A., Rosati, R., Zakharyaschev, M.: Ontology-based data access: A survey. In: IJCAI (2018)